

Faculdade de Engenharia da Universidade do Porto



Alien Express

Laboratório de Computadores

Turma 2LEIC16

Grupo 04

Membros do grupo:

Haochang Fu - 202108730

Isabel Moutinho - 202108767

João Cardoso - 202108732

Pedro Paixão - 202008467

Índice

1. Instruções do Utilizador.....	4
1.1 Objetivo.....	4
1.2 Jogador.....	4
1.2.1 Inventário.....	4
1.2.2 Equipamento.....	4
1.3 Teleportes e Encomendas.....	5
1.3.1 Itens.....	5
1.4 Inimigos.....	5
1.4.1 Predadores.....	5
1.4.2 Inofensivos.....	6
1.4.3 “Keep your enemies close”.....	6
1.5 Resumo de Comandos.....	6
2. Estado do Projeto.....	7
2.1 Funcionalidades.....	7
2.1 Dispositivos.....	7
2.1.1 Timer.....	8
2.1.2 Placa Gráfica.....	8
2.1.3 Teclado.....	9
2.1.4 Rato.....	10
2.1.5 RTC.....	11
3. Organização e estrutura do código.....	11
3.1 graphics.c (3%).....	11
3.2 keyboard.c (2%).....	11
3.3 mouse.c (3%).....	11
3.4 rtc.c (1%).....	11
3.5 timer_proj.c (1%).....	11
3.6 game.c (5%).....	12
3.7 menu.c (3%).....	12
3.8 level.c (4%).....	12
3.9 pause.c (3%).....	12
3.10 gameover.c (3%).....	12
3.11 view.c (19%).....	12
3.12 spawner.c (5%).....	12
3.13 bullet.c (1%).....	12
3.14 enemies_lvl.c (2%).....	12
3.15 inventory.c (2%).....	12
3.16 player.c (11%).....	12
3.17 teleport.c (2%).....	13


3.18 wolf.c (6%).....	13
3.19 workers.c (6%).....	13
3.20 explosion.c (1%).....	13
3.21 ammo.c (1%).....	13
3.22 backpack.c (1%).....	13
3.23 bone.c (3%).....	13
3.24 boots.c (1%).....	13
3.25 grenade.c (5%).....	13
3.26 package.c (3%).....	13
3.27 pistol.c (1%).....	13
3.28 pizza.c (1%).....	13
3.29 watch.c (1%).....	13
3.30 Gráfico de chamada de funções.....	14
4. Detalhes da Implementação.....	15
4.1 Colisões.....	15
4.2 Gestão de inventário.....	15
4.3 Spawner.....	15
4.4 Game state.....	15
5. Conclusão.....	16

1. Instruções do Utilizador

1.1 Objetivo

No nosso jogo, o utilizador controla um astronauta, cujo objetivo é coletar encomendas que aparecem pela galáxia, e colocá-las em teleportes distribuídos pelo jogo, enquanto evita inimigos que o perseguem ou tentam sabotá-lo de outras formas. Enquanto tenta sobreviver o máximo de tempo possível, o jogador pode aumentar a sua pontuação no jogo ao colocar encomendas nos teleportes.

1.2 Jogador

O jogador tem a escolha de se deslocar em todas as direções, usando as teclas  , no entanto o número de encomendas que tem na sua posse irá afetar a velocidade a que se desloca. Também tem a habilidade de atacar os seus inimigos, ora com os seus punhos, ora uma arma que encontre. Para isso usa a tecla 'L' do teclado.

1.2.1 Inventário

De forma a gerir todos itens na sua posse, o jogador possui um inventário com capacidade máxima base de 3 itens. Quer o rato, quer as teclas '1', '2' e '3' do teclado, podem ser usadas para selecionar um item do inventário. Selecioná-lo significa que o jogador pode interagir com ele.

1.2.2 Equipamento

Certos itens podem ser equipados pelo jogador, dando diferentes tipos de bônus, itens equipados não contam para a capacidade do inventário.



Mochila - aumenta a capacidade máxima do inventário para 9 itens. Ainda assim, apenas se pode selecionar itens nas posições 1, 2 e 3, itens noutras posições podem ser movidos para essas com o rato.



Botas - aumentam a velocidade base do jogador.

Estes itens, depois de coletados pelo jogador, podem ser equipados usando o rato para o passar do inventário para o equipamento, de capacidade máxima de 3 equipamentos.

1.3 Teleportes e Encomendas







Teleportes aparecem no jogo em lugares aleatórios e possuem um temporizador, se nesse intervalo de tempo não receberem uma encomenda, retiram pontos ao jogador, e o inverso se receberem, sendo que, em ambos os casos desaparecem.

Para coletar uma encomenda utiliza-se a tecla 'P' do teclado, para largar num teleporte ou no chão utiliza-se a tecla 'O' do teclado.

Além de colocar as encomendas coletadas nos teleportes, o jogador pode escolher abrir a encomenda com a tecla 'I' do teclado e uma série de itens pode ser obtida de dentro dela, esta ação inevitavelmente descarta a encomenda.

1.3.1 Itens

Os itens obtidos podem ser usados, largados e coletados da mesma forma que as encomendas e garantem diversas vantagens:



-  Pizza - fornece vida ao jogador quando consumida
-  Uranium - uma simples barra de uranium, não aparenta ter nenhum uso.. (ver [1.4.3](#))
-  Granada - pode ser lançada na direção para a qual o jogador está orientado. Explode passado uns segundos, dando dano a ambos o jogador e inimigos na área à sua volta.
-  Munição - podem ser colocadas na pistola.
-  Pistola - quando provida de munições, dispara na direção para a qual o jogador está orientado.
-  Relógio - dispositivo extremamente raro, permite ver as horas.
- Mochila e Botas - ver [1.2.2](#)

1.4 Inimigos

No mundo do Alien Express há vários tipos de monstros que põem em risco a vida do jogador de formas diferentes.

1.4.1 Predadores

Seguem o jogador, contacto direto com eles tira vida ao jogador.

-  Zombies - inimigo mais comum no jogo, aparecem mais frequentemente e em maior quantidade. Poder ofensivo e resistência médios.
-  Lobos - inimigo raro, tem probabilidade de aparecer mais tarde no jogo. Tem o mesmo poder e resistência dos Zombies mas desloca-se mais rapidamente.

1.4.2 Inofensivos

Não dão dano direto ao jogador, mas ainda assim são perigosos.



Workers - andam extremamente rápido, dirigem-se às encomendas com a intenção de as roubar para fora do jogo, impedindo o jogador de concretizar o seu objetivo.

Não têm poder ofensivo e a sua resistência é muito reduzida.

Tip: apenas fogem quando perseguidos, caso o jogador pare perto deles, irão até ele.

1.4.3 “Keep your enemies close”

Os lobos do universo do Alien Express têm duas vulnerabilidades, a primeira é que gosta de jogar ao ‘lançar o pau’, se lançado algo semelhante a um osso para o chão estes irão atrás dele em vez do jogador, a segunda é que ingerir uranium que é fatal. Sabendo isto, e na posse de uranium, o jogador pode optar por deixar o lobo roê-lo, causando a morte do inimigo, ou jogar ao ‘lançar o pau’ com ele até ganhar a sua confiança. Se conseguir alcançar isto, o lobo juntar-se-á ao jogador na sua missão de distribuir encomendas, não só levando-as até aos teleportes mas também matando qualquer worker que tente agarrar a sua encomenda.

Tip: Quando tirar o uranium ao lobo, para voltar a lançar, segure o uranium (hold I), se apenas pegar nele será morte certa.

Se só lançar o uranium para perto do lobo, pode não ser divertido o suficiente para ganhar a amizade dele.

1.5 Resumo de Comandos

Rato - movimentar itens no inventário, equipamento e entre eles.

A - deslocar o jogador para a esquerda.

D - deslocar o jogador para a direita.

W - deslocar o jogador para cima.

S - deslocar o jogador para baixo.

P - pegar em qualquer item.

O - pousar qualquer item.

I - interagir com qualquer item (abrir, consumir, lançar, etc.).

I pressionado - no caso de itens lançáveis, lança-os mais longe.

L - atacar.

1/2/3 - seleccionar o slot respetivo do inventário.

2. Estado do Projeto

2.1 Funcionalidades

Funcionalidade	Dispositivo(s)	Completação
Relógio do jogo	Timer	100%
xpm e xpm animados	Gráfica, Timer	85%
Interação do rato com o jogo	Rato	60%
Tempo real a aparecer	RTC	95%
Diversidade de itens	Teclado	70%
Mecânicas de monstros	Teclado, Timer	80%
Interface do jogo	Teclado, Rato	90%

2.1 Dispositivos

Dispositivos	Finalidade	Interrupções
Timer	Aplicar a frequência em que tudo é desenhado. Temporizador do jogo.	Y
Gráfica	Gestão dos buffers. Responsável por desenhar no ecrã. Controlo do modo de jogo.	N
Teclado	Movimentar e controlar a personagem principal. Selecionar as opções de menu, pausa, fim de jogo.	Y
Rato	Selecionar as opções de menu, pause, fim de jogo. Controlar inventário do jogador	Y
RTC	Mostrar as horas ao jogador	Y

2.1.1 Timer

O timer foi usado para controlar a frequência em que os xpm's são desenhados. Em termos de funcionalidades no jogo, o timer também gere o tempo que o jogador demora no nível de jogo, aumentando o valor do temporizador, assim permitindo uma espécie de progresso em que o jogador tenta obter um melhor tempo de jogo cada vez que entra num nível. O aumento do temporizador também altera outras funcionalidades, tal como o nascimento dos monstros, encomendas e teleports que aparecem mais frequentemente ao longo do tempo.

O timer é usado no ficheiro `timer_proj.h` e `timer_proj.c`:

Para controlar a frequência em que os xpm's são desenhados, utiliza-se a função `timer_set_frequency_proj`, Para aumentar o temporizador utiliza-se a função `timer_int_handler_proj`.

2.1.2 Placa Gráfica

A placa gráfica foi usada para desenhar as entidades do jogo para o ecrã, onde o utilizador os pode visualizar. Para inicializar foi usado as funções `set_frame_buffer` e `set_graphics_mode` do ficheiro de `graphics.c` e `graphics.h`. Essas duas funções vão inicializar a placa gráfica num certo modo, que no nosso caso usamos o modo `0x14C` que dispõe 4 bytes por pixel a ser desenhado no modo direto numa resolução de `1152x864` com número de cores total de 2 elevado a número de bits por pixel, ou seja, 4294967296 cores diferentes.

A forma como as entidades estão a ser desenhadas, são através de load de xpm's e armazenamento destes mal que o programa começa. Desta forma, os xpm's não estão a sempre a ser carregados em cada interrupção do timer e a cache do sistema não se sobrecarrega. As funções que criam os xpm's e os armazenam estão nos ficheiros `view.c/view.h`. A função `make_img` é responsável pela conversão do xpm para a struct `img_t` e as funções com prefixo *init* são responsáveis pela seleção dos seus xpm's identificadores para os posteriormente criar.

Algumas entidades são animadas tal como os monstros, o jogador e até o cursor do rato, logo para essas entidades são guardadas nas structs `AnimatedImg_t`, `simple_animation_t` através das funções criadoras dessas animações `make_animated_img`, `make_simple_animation` respetivamente.

Algumas entidades colidem umas com as outras tal como alguns monstros, o jogador e bem como as encomendas e teleports. Para isso utilizámos duas funções para verificar as colisões, que se encontram no ficheiro `view.c/view.h`: `is_occupied`, `manage_collision`. A função `is_occupied` verifica se na posição para onde a entidade se quer mover no `prev_buffer` já está ocupada e a função `manage_collision` verifica o mesmo no `frame_buffer` e movimenta se não estiver ocupada.

Estas duas funções são usadas para colisões entre monstros, paredes e o jogador, no entanto para colisões para apanhar encomendas e itens do chão, bem como largar encomendas nos teleports e matar monstros, foram utilizados métodos de coordenadas, em que se verifica se as coordenadas de uma entidade se encontra dentro das coordenadas de outra. Isto é feito nas funções `hit_monster`, `kill_monster_radius`, `is_on_teleport`, `manage_hitbox`, `remove_package`, `remove_item_from_ground`, `kill_workers`.

Foram utilizados 4 buffers, o buffer `vm_buffer` atualiza tudo que estiver nele para o ecrã, o buffer `frame_buffer` é o que em cada interrupt do timer, todas as entidades são desenhadas nele e depois, através da função `pass_to_vm_buffer`, os valores do `frame_buffer` são todos passados para a `vm_buffer`, após isto o `frame_buffer` é reiniciado. O buffer `prev_buffer` controla aspetos das colisões entre entidades guardando nele em cada interrupt do timer o valor do `frame_buffer` antes deste se reiniciar, isto é feito na função `pass_to_vm_buffer`. Finalmente, o buffer chamado `map` é utilizado para ter lá dentro o background do jogo, de forma a não estar sempre a desenhar o background que é algo estático no jogo inteiro. Este buffer em cada interrupt do timer, passa os valores de lá dentro para o `frame_buffer` através da função `pass_map`.

Por fim, para desenhar as entidades, utiliza-se 3 funções de desenho que se encontram nos ficheiros `graphics.c/graphics.h`: `draw_pixel` desenha um pixel no `frame_buffer` os valores passados como argumento, `draw_pixel_map` desenha num pixel do buffer chamado `map` os valores passados como argumento e o `draw_pixel_number` que desenha num pixel do `frame_buffer` entidades como números, letras e símbolos. Estas funções por sua vez são chamadas nos ficheiros `view.c/view.h` através das funções `draw`, `draw_map`, `draw_numbers` respetivamente.

Os xpm's são todos guardados na pasta `Assets` final.

2.1.3 Teclado

O teclado é usado para controlar a maior parte do jogo, como o movimento, ações, navegação de menu, pausa e final de jogo.

Na navegação de menu, pause e final de jogo, o usuário clica nas teclas `W` e `S` e `ENTER` para efetuar decisões e também `ESC` para entrar e sair da pausa.

No nível o jogador já pode movimentar com `W,A,S,D` e com as teclas `P`, `O`, `I`, `L` pode efetuar diversas ações previamente explicadas.

Com isto dito, a maior parte da gestão do teclado encontra-se nos ficheiros `keyboard.c/keyboard.h`, onde em cada interrupção do teclado é lido um `scancode` através da função `kbc_read_output`, e para restaurar os valores do teclado utiliza-se o `kbc_write_output`, `keyboard_restore`.

Já no ficheiro `game.c/game.h`, a função `keyboard_ih` chama a respetiva função de controlo dependendo do estado do jogo. Os diferentes estados são: Menu, Level, Pause, Game Over, e nos ficheiros `menu.c/menu.h`, `level.c/level.h`, `pause.c/pause.h`, `game_over.h/gameover.c` encontram-se as funções que controlam o teclado em cada estado.

Além disso dependendo na tecla que se pressiona, o estado atual muda através da função `control_state` situada no `game.c/game.h`

2.1.4 Rato

Faz o mesmo género de coisas que o teclado faz, no entanto não controla o movimento nem as ações do jogador, mas sim só é responsável pela navegação do menu, pause, fim de jogo e a gestão do inventário do jogador.

No menu, pausa e fim de jogo, o jogador pode colocar o rato por cima de cada opção, a sua imagem muda e as letras tornam-se verde de forma a comunicar com o user que o programa está a registar o que ele pretende fazer, se clicar provoca uma ação que muda de estado do jogo.

Para o inventário basta clicar numa caixa do inventário com o botão de esquerda do rato e a deslizar para outra caixa para trocar de lugares, ou simplesmente deslizar para fora do inventário para o item nela ser removido do inventário ou só apenas clicar no botão direito para o largar, também o botão do meio interage com o item do inventário.

A maior parte da gestão do rato se encontra nos ficheiros `mouse.c/mouse.h`. Para ler os valores que o user usou no rato, utiliza-se a função `mouse_read_output` e para os gerir e guardar usa-se a função `mouse_control` e `fill_mouse_packet` que os guarda numa struct packet. Para movimentar o cursor do rato usa-se a função `move_mouse` e para registar as colisões do rato com entidades usa-se `mouse_position`.

Já no ficheiro `game.c/game.h`, é usado a função `mouse_proj_ih` para gerir o input do user com base no estado do jogo. Dependendo do estado do jogo esta função irá chamar outras funções específicas desse estado nos ficheiros `menu.c/menu.h`, `pause.c/pause.h`, `level.c/level.h` `game_over.h/gameover.c`.

Além disso, dependendo do botão que o user utilizou no rato, o estado atual muda através da função `control_state` situada no `game.c/game.h`.

2.1.5 RTC

Mostra um temporizador que se iguala ao tempo atual. Guarda num array os valores dos segundos, minutos, horas, dia, mês e ano, no entanto para ser mais agradável, decidimos apenas mostrar os segundos, minutos e as horas. Isto apenas é visível após apanhar um item especial de uma encomenda, chamado de relógio, e quando a seleccionar o relógio no inventário, aparece ao lado deste o tempo real.

A maior parte do código do RTC se encontra no ficheiro `rtc.c/rtc.h`. A função `update`, serve para autorizar updates ou não autorizar updates do rtc, a função `read` lê os valores do rtc e a função `wait`, espera até esses valores puderem ser lidos. Enquanto a função `rtc_ih` é chamada em cada interrupção do rtc, para atualizar o array `rtc_time`, lendo assim os valores dos segundos, minutos, horas, dia, mês e ano e guardando-os.

3. Organização e estrutura do código

3.1 `graphics.c` (3%)

Neste módulo, encontra-se toda a informação sobre a placa gráfica, inicialização desta, controlo dos buffers e bem como os desenhos neles.

3.2 `keyboard.c` (2%)

Neste módulo, encontra-se toda a informação sobre a gestão do teclado, desde as subscrições, leitura de scancodes, bem como a restauração do teclado na terminação do programa.

3.3 `mouse.c` (3%)

Neste módulo, encontra-se toda a informação sobre a gestão do rato, desde as subscrições, leitura de scancodes, restauração do rato após terminação do programa e inicialização também, bem como o controlo do da struct `packet`, que contém todas as informações sobre o rato. Também é feito o movimento do cursor do rato bem como as colisões dele.

3.4 `rtc.c` (1%)

Neste módulo, encontra-se toda a informação sobre a gestão do rtc, desde as subscrições, `update`, espera e leitura do rtc, bem como o handle da interrupção.

3.5 `timer_proj.c` (1%)

Neste módulo, encontra-se toda a informação sobre a gestão do timer, desde as subscrições, busca da configuração, controlo da frequência das frames e incremento de um contador(handle das interrupções).

3.6 game.c (5%)

Neste módulo, gera-se a inicialização, terminação e o game loop principal do jogo. Controla-se o estado e as ações do jogo, bem como as inputs que o user faz. Essencialmente, age como um módulo controlador de todos os outros módulos que representam o estado do jogo.

3.7 menu.c (3%)

Neste módulo, controla-se o desenho e os inputs do rato e teclado do estado de menu.

3.8 level.c (4%)

Neste módulo, controla-se o desenho, movimentos e os inputs do rato e teclado do estado de level.

3.9 pause.c (3%)

Neste módulo, controla-se o desenho e os inputs do rato e teclado do estado de pause.

3.10 gameover.c (3%)

Neste módulo, controla-se o desenho e os inputs do rato e teclado do estado de gameover.

3.11 view.c (19%)

Neste módulo, cria-se todos os xpm's, guardando-os em structs identificadores. Controla-se as colisões e gera-se a forma como as entidades serão desenhadas.

3.12 spawner.c (5%)

Neste módulo, gera-se o aparecimento de certas entidades no nível, monstros, encomendas e teleports dependendo do tempo que já passou e da probabilidade que cada um tem de aparecer.

3.13 bullet.c (1%)

Neste módulo, controla-se o desenho e o movimento das balas.

3.14 enemies_lvl.c (2%)

Neste módulo, controla-se o desenho e o movimento dos zombies.

3.15 inventory.c (2%)

Neste módulo, controla-se o desenho e gestão do inventário.

3.16 player.c (11%)

Neste módulo, controla-se o desenho e o movimento do jogador, bem como as ações que ele pode tomar.

3.17 teleport.c (2%)

Neste módulo, controla-se o desenho, interação e animação dos teleports.

3.18 wolf.c (6%)

Neste módulo, controla-se o desenho e o movimento do lobo.

3.19 workers.c (6%)

Neste módulo, controla-se o desenho e o movimento dos workers.

3.20 explosion.c (1%)

Neste módulo, posiciona-se e anima-se/desenha-se as explosões.

3.21 ammo.c (1%)

Neste módulo, busca-se e desenha-se as munições.

3.22 backpack.c (1%)

Neste módulo, busca-se, desenha-se e usa-se a mochila.

3.23 bone.c (3%)

Neste módulo, busca-se, desenha-se e anima-se o osso de urânio.

3.24 boots.c (1%)

Neste módulo, busca-se, desenha-se e usa-se as botas.

3.25 grenade.c (5%)

Neste módulo, busca-se, desenha-se, anima-se e usa-se as granadas.

3.26 package.c (3%)

Neste módulo, desenha-se e usa-se as encomendas.

3.27 pistol.c (1%)

Neste módulo, busca-se, desenha-se e usa-se as pistolas.

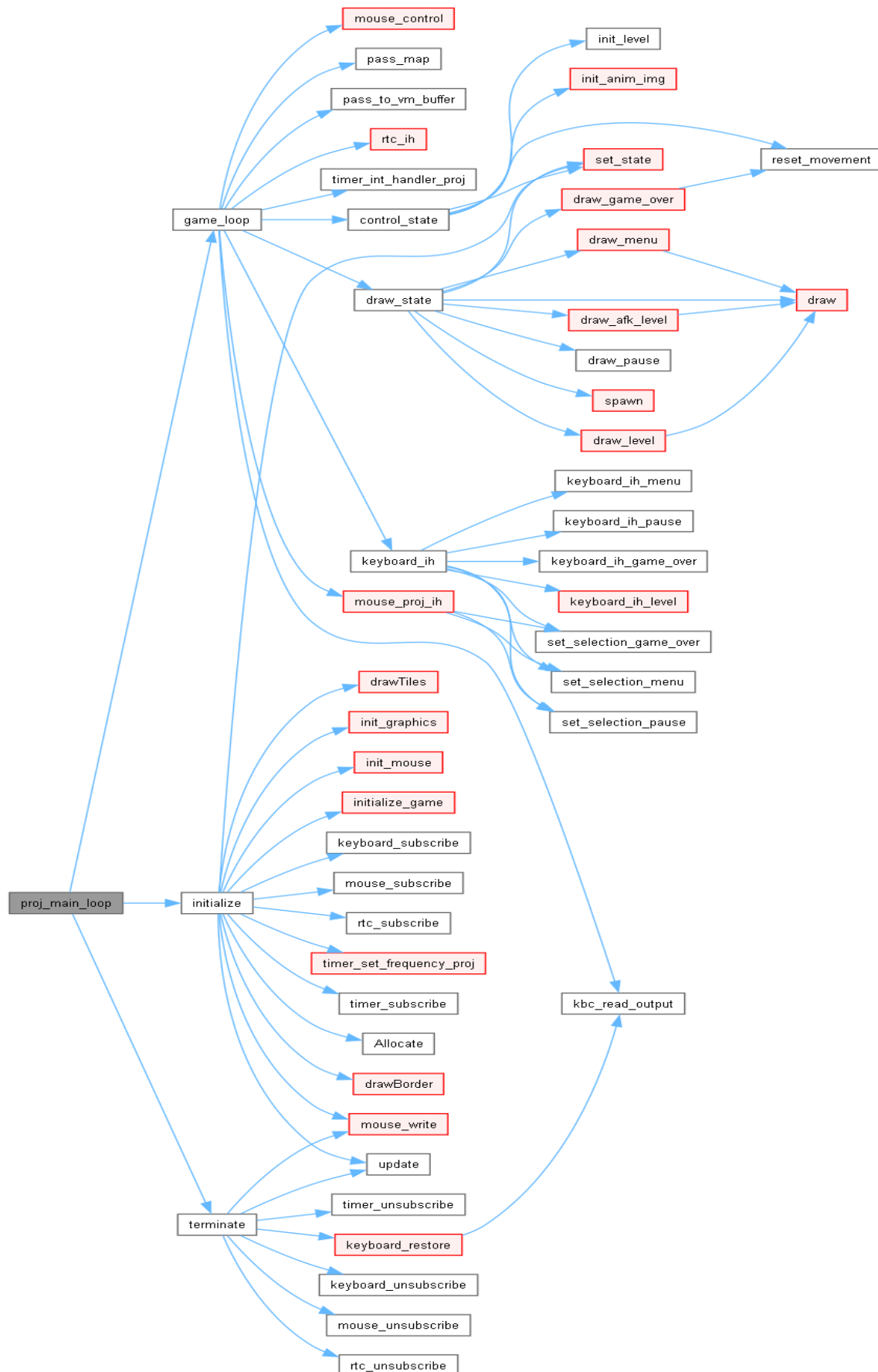
3.28 pizza.c (1%)

Neste módulo, busca-se, desenha-se e usa-se as pizzas.

3.29 watch.c (1%)

Neste módulo, busca-se, desenha-se o relógio e os valores de tempo real do rtc.

3.30 Gráfico de chamada de funções



4. Detalhes da Implementação

4.1 Colisões

Para as colisões foram utilizados 2 buffers, um para registar o conteúdo das entidades desenhadas anteriormente (`prev_buffer`) e outro para registar as entidades a serem desenhadas atualmente. Isto foi feito, porque cada entidade tem uma hitbox e, ao movimentar cada entidade, elas irão verificar se na posição onde ela quer ir se já estava desenhado no buffer anteriormente e também vai verificar se atualmente está lá desenhado algo. Só após estas duas condições serem satisfeitas é que a entidade é desenhada.

4.2 Gestão de inventário

Para gerir o inventário, utilizamos uma array que guarda as posições do inventário, sempre que se apanha um item, ele é colocado no array de index mais pequeno e que não esteja a ser usado atualmente. Desta forma, em vez de estar a mexer o item para o local do inventário apenas é necessário passar a cor do item para o inventário para que seja lá assim desenhado.

4.3 Spawner

Para fazer as entidades aparecerem numa determinada parte do nível e numa certa quantidade, foi utilizado na classe de spawner, um controlo do aparecimento destes com base no tempo que já passou do jogo e a probabilidade específica de cada entidade. Tendo em atenção que os monstros aparecem nos cantos do jogo e as encomendas e teleports podem aparecer em qualquer parte do mapa.

4.4 Game state

Para mudar de estado o user tem de fazer uma ação, quer com o teclado quer com o rato. Cada classe de um estado tem nela uma função que controla os inputs do teclado e outra que controla as do rato. E estes input são próprios do estado, e dependendo de o input feito é retornado para uma classe controladora de estados (`game.c`) o resultado da ação efetuada, e com base nessa ação é alterado o estado do jogo.

5. Conclusão

Devido a falta de tempo no projeto, algumas funcionalidades foram sacrificadas:

- Um final para o jogo.
- Um sistema de loja onde se pode comprar e vender itens.
- Movimento de câmara cada vez que o jogador anda e mapa do nível.
- Personagem de boss que aparece num certo tempo e que deixa cair recompensas.
- Juntar itens para formar novos itens.
- Uso de paredes para impedir monstros de se aproximarem.

Acerca das funcionalidades implementadas, estamos bastante orgulhosos, pois conseguimos trazer estas entidades para o ecrã e animá-las, bem como criar um jogo desafiante mas divertido ao mesmo tempo.

Para concluir, este projeto foi muito interessante e conseguimos aprender muito em termos de não só como cada dispositivo funciona e como os programar, mas também como criar um jogo e estruturar, organizar o código de forma a facilitar o progresso. Também a criação de um relatório final tão extenso e informativo ajudou-nos muito a perceber que passos a tomar a seguir, bem como melhorar o que temos presentemente.