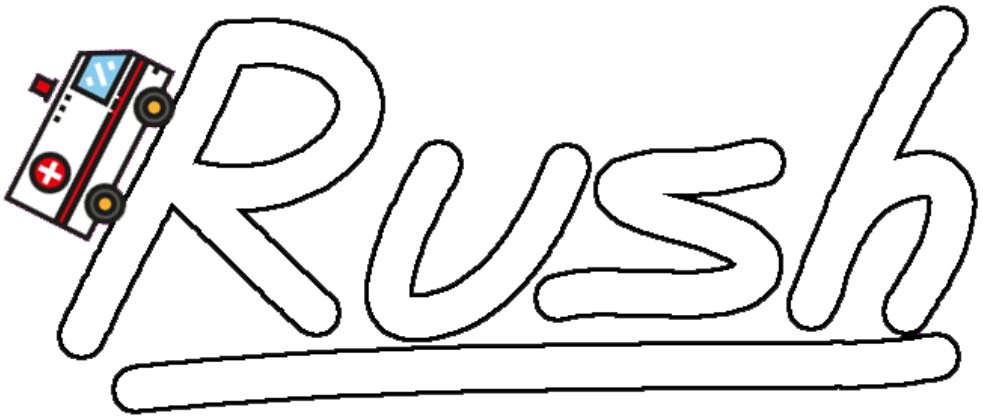


RUSH

Projeto LCOM 17/18

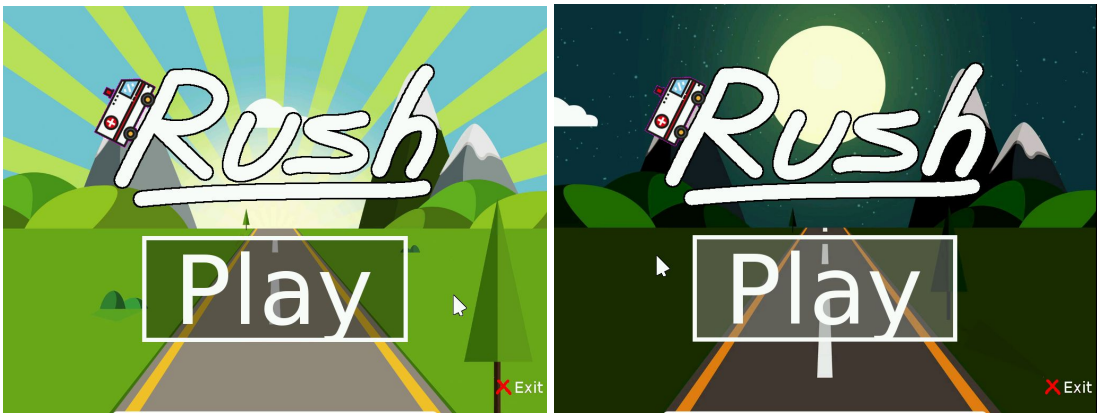
Ângelo Teixeira - up201606516
Henrique Lima - up201606525

Índice

1. Instruções de Utilização

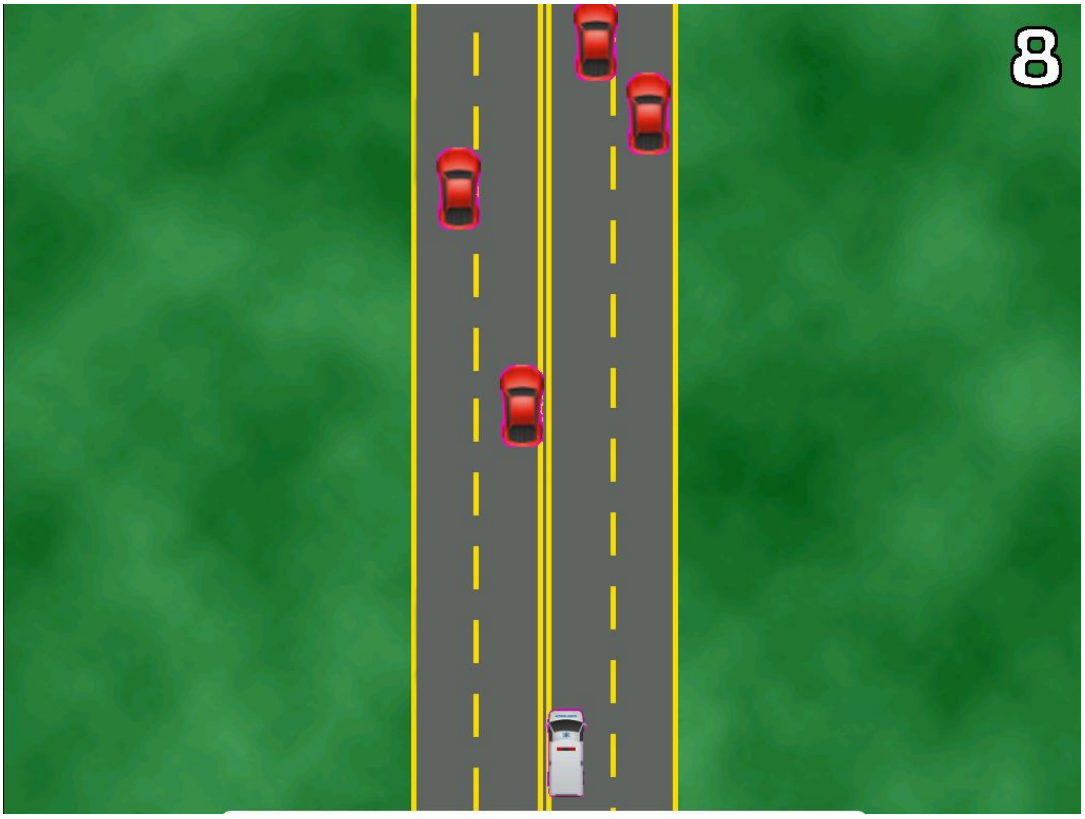
1.1. Menu Inicial

Ao iniciar o programa, é mostrado um menu inicial, onde é possível, através do cursor, selecionar uma de duas opções: Começar o jogo (PLAY) ou sair do programa (EXIT).



1.2. Game Play

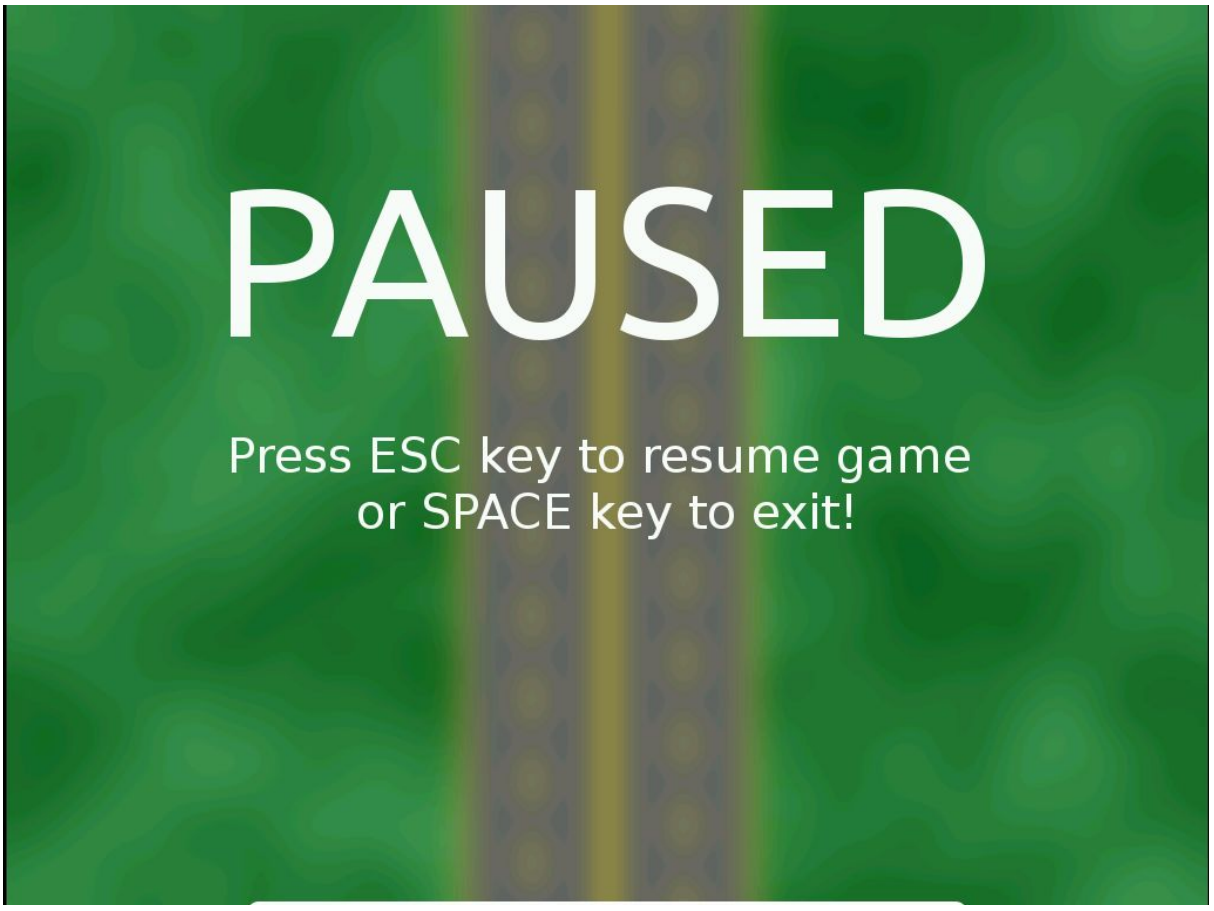
Tendo carregado no botão de “PLAY” no Menu Inicial, o jogo inicia-se. O objetivo é desviar-se de outros veículos em trânsito através do movimento do rato (Horizontal). Quanto mais tempo ficar até colidir no máximo 3 vezes com outros veículos, maior será a pontuação final. Por vezes é necessário acelerar a ambulância para ultrapassar outros veículos e evitar acidentes, para tal, é possível pressionar a tecla ESPAÇO.
Para colocar o jogo em pausa basta carregar na tecla ESC.



1.3. Menu de Pausa

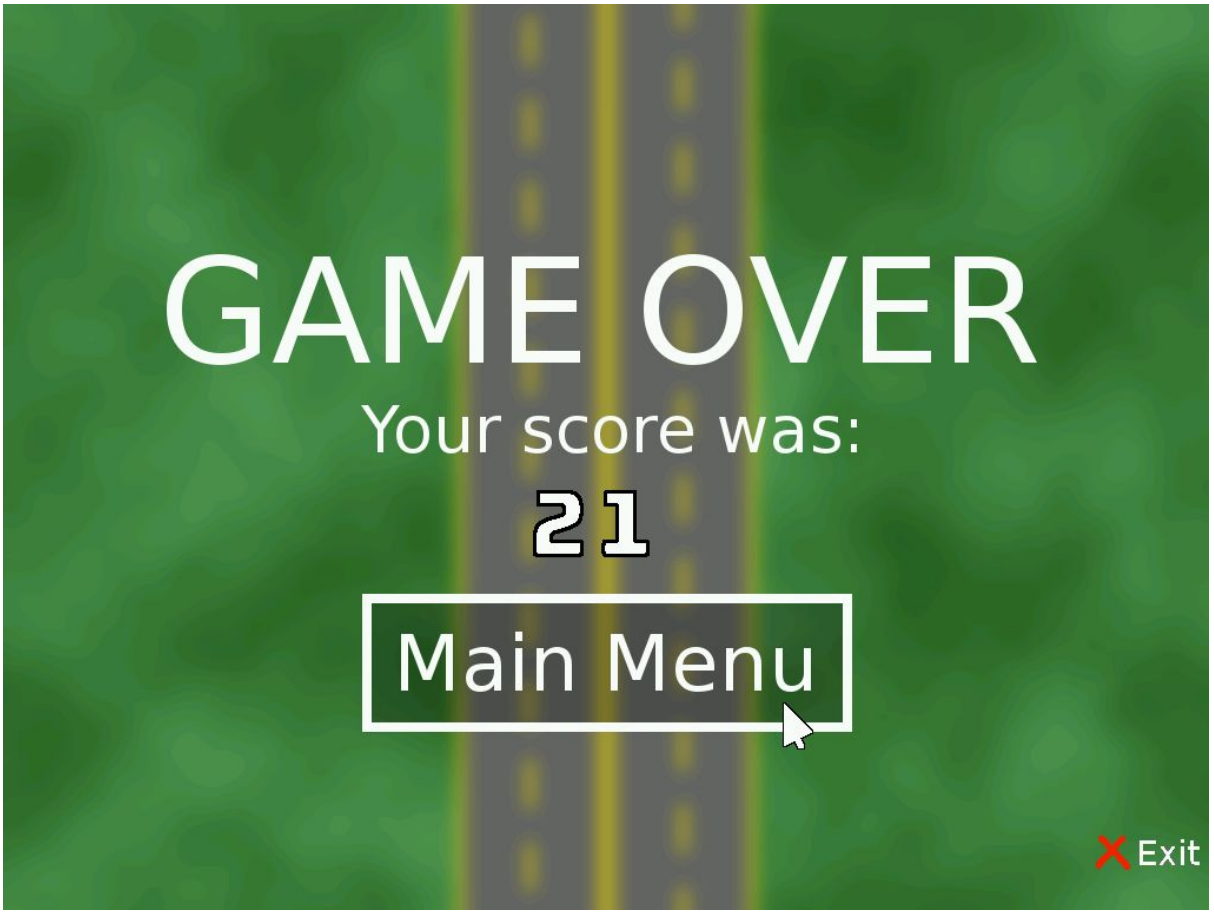
Estando no menu de pausa, é possível retomar o jogo atual, ou voltar ao menu inicial, descartando o progresso atual.

Para voltar ao jogo, basta carregar ESC novamente, para ir para o menu inicial, basta carregar ESPAÇO.



1.4. Game Over

Quando o utilizador colide com outros veículos mais de 3 vezes, perde o jogo e é mostrado o ecrã de “Game Over” onde pode consultar a pontuação alcançada. Para voltar ao menu inicial, basta carregar em Main Menu. Para sair do jogo, clicar em Exit.



2. Estado do Projeto

2.1. Dispositivos Implementados

Dispositivo	Funcionalidade	Usa Interrupções
Timer	Atualizar o ecrã com uma determinada frequência	Sim
Keyboard	Navegar entre estados de jogo (Sair de pausa, de Game Over) e, em Game Play, acelerar a ambulância	Sim
Mouse	Selecionar opção no Menu Inicial. Mover horizontalmente a ambulância em Game Play	Sim
Video Card	Representar o jogo visualmente	Não
RTC	Obter o tempo atual para modificar o design “dia/noite”	Não

2.1.1. Timer

No programa foram usadas as interrupções geradas pelo Timer 0 do MINIX. Por definição, este gera uma interrupção a cada 1/60 segundos, pelo que esta frequência foi mantida para ter um jogo com 60 frames por segundo (cada frame é gerado a cada interrupção do timer).

A abstração desde dispositivo encontra-se em timer.c, contendo funções de subscrição / cancelamento de subscrição de interrupções.

Estas são usadas no módulo gameEngine.c, que contém a função mainGameLoop() cujo objetivo é ter um ciclo que verifica interrupções dos dispositivos usados, incluindo o timer, executando diferentes ações dependendo das interrupções recebidas.

2.1.2. Keyboard

O teclado é usado para obter input do jogador/utilizador, nomeadamente cliques de teclas específicas. As interrupções geradas pelo mesmo são processadas no módulo gameEngine.c, similarmente ao Timer, e tem a sua abstração definida em keyboard.c, que, para além das funções de subscrição/cancelamento de subscrição de interrupções, tem também uma

função kbd_get_pressed_key(), que processa os códigos (scancodes) das teclas premidas e retorna esses mesmos scancodes.
No ciclo principal (em mainGameLoop()), assim que é recebida uma interrupção do teclado, é chamada uma outra função da abstração: a keyboardHandler(), que recebe a lista de eventos do jogo e um scancode e cria um evento do tipo “KEY_PRESS” associado ao código (scancode) da tecla premida, para processamento futuro desse evento associado ao clique da tecla.

2.1.3. Mouse

O rato é também usado para obter input do jogador/utilizador, nomeadamente cliques e movimento, sendo os cliques e o movimento usados no menu inicial, movendo um cursor, e sendo usado apenas o movimento no Game Play para deslizar a ambulância horizontalmente. As interrupções geradas pelo mesmo são processadas no módulo gameEngine.c, similarmente ao Timer e ao Keyboard, e tem a sua abstração definida em mouse.c, que, para além das funções de subscrição/cancelamento de subscrição de interrupções, tem também uma função mouse_movement_handler(), que processa as interrupções geradas pelo rato, quer seja pelos cliques ou por movimento, retornando um array de 3 bytes, sendo este o packet enviado pelo rato.
No ciclo principal (em mainGameLoop()), assim que é recebida uma interrupção do rato, é chamada uma outra função da abstração: a mouseHandler(), que recebe a lista de eventos do jogo e um packet e cria um evento do tipo “MOUSE_CHANGE” associado ao movimento em x ou y ou ao clique do botão esquerdo/direito, para processamento futuro desse evento.

2.1.4. Video Card

A placa de video é usada para representar graficamente o jogo, através de alterações à cor dos píxeis representados no ecrã.
Neste jogo, utilizamos o modo VBE 0x117, que usa códigos de 16 bits (RGB565) para identificar as cores, isto é, 5 bits para representar a “quantidade” de vermelho (R), isto é 6 bits para representar a “quantidade” de verde (G) e 5 bits para representar a “quantidade” de azul (B), resultando num total de 2^16 = 65536 cores possíveis para cada pixel. Com este modo consegue-se também uma resolução maior que a normal (800x600), sendo possível representar em 1024x768.
A abstração deste dispositivo está definida em video_gr.c sendo o módulo vbe.c auxiliar à mesma.
Nela estão contidas as funções de inicializar o modo gráfico (vg_init()), bem como sair do mesmo (vg_exit()) (voltar para o modo de texto), onde são usadas as funções VBE 0x02 para mudar o modo gráfico e 0x00 para retornar ao modo de texto. Em relação à implementação no jogo, usamos double buffering, e temos objetos (sprites) animados com deteção de

colisões. Temos também uma font, que é o conjunto de 10 algarismos para mostrar a pontuação do jogador.

2.1.5. RTC

O RTC é usado para obter o tempo atual, mais concretamente a hora. Com esta informação modificamos o jogo visualmente, de forma a que a partir das 19 horas e até às 7, o jogo fique em modo noturno, com gráficos alusivos a esse efeito.

3. Organização do Código

3.1. Fila de Eventos (ev_queue.c) (10%)

Este módulo contém a abstração para uma fila de eventos, que é usada para guardar os eventos gerados pelos periféricos ou por ações no jogo. Em termos de implementação, é uma lista ligada, com elementos do tipo ev_queue_elem_st que contêm a informação do evento e um apontador para o elemento seguinte. O módulo disponibiliza as funções de push (ev_queue_push()) e pop (ev_queue_pop()) que adicionam um dado elemento ao fim da fila e retiram o primeiro elemento da fila, respetivamente. Há também um "construtor" de lista e "destrutor" (ev_queue_create() e ev_queue_free()).

Intervenção dos membros

Ângelo Teixeira - Implementação da abstração inicial realizando a maior parte das funções e correção de erros posteriormente (70%).
Henrique Lima - Correção de erros (30%).

3.2. Game Objects (gameObjects.c) (10%)

Este módulo contém a abstração para uma "camada" de objetos e para um objeto de jogo.
A "camada" de objetos (z-index) é uma abstração feita para lidar com sobreposição de objetos na mesma posição e é feita através de um array de ObjLayer, sendo cada ObjLayer uma "camada" que contém um array de gameObject. Um gameObject é um elemento do jogo que tem uma posição x,y, velocidade vx,vy, um Bitmap associado (o sprite), a cor transparente desse objeto (cor que não será desenhada), uma label (para identificar o objeto), o tamanho (sizeX, sizeY), e o zIndex (numero da "camada" à qual pertence). Tendo isto, a abstração permite adicionar objetos a qualquer camada, ou removê-los através de realloc (zIndexPush(), zIndexRemove(), zIndexRemoveByIndex()). É possível desenhar um dado objeto (drawObject()), dependendo da posição do dado objeto. Similarmente à Fila de Eventos, tem também os "construtores" e "destrutores" dos objetos (ptr_createObjLayer(), createObjLayer(), createGameObject(), createActiveObjectsList(), freeGameObject()).
Por fim, disponibiliza também uma função de "pesquisa" de objeto por label (getObjectFromLabel()), que retorna o primeiro objeto encontrado com a label pretendida.

Intervenção dos membros

Henrique Lima - Implementação inicial da abstração realizando a maior parte das funções e correção de erros posteriormente (80%).
Ângelo Teixeira - Correção de erros (20%).

3.3. Maquina de Estados (stateMachine.c) (10%)

Este módulo contém a abstração para a Maquina de Estados do jogo.
Foi realizada de forma a integrar o processamento de eventos contidos na fila de eventos. A máquina de estados é formada por vários gameState_st, que são os "estados" da máquina. Este contêm informação sobre o de estado de jogo (MAIN_MENU, GAME_PLAY, PAUSE_MENU, GAME_OVER, GAME_EXIT) bem como os objetos de cada estado (p.ex. os objetos de MAIN_MENU sao diferentes de GAME_PLAY) e a pontuação atual.
Contém funções para criar estados (newGameState()), inicializar a máquina (startStateMachine() - coloca o estado dado como estado inicial) e outra função para iniciar a máquina e, consequentemente, o jogo (startStates()), que chama a função startup(), que é ligada ao jogo em si e à inicialização dos objetos e lógica de jogo.

Intervenção dos membros

Ângelo Teixeira - Implementação da abstração inicial realizando a maior parte das funções e correção de erros posteriormente (80%).
Henrique Lima - Correção de erros (20%).

3.4. Timer (timer.c) (10%)

Este módulo contém a abstração para uso do timer do minix, incluindo a subscrição e anulamento da mesma de interrupções do timer, para uso no jogo.

Intervenção dos membros

Esta abstração foi realizada no Lab 2, pelo que a distribuição do trabalho foi equilibrada. Cada membro realizou 50%.

3.5. Keyboard (keyboard.c) (10%)

Este módulo contém a abstração para uso do teclado, incluindo a subscrição e anulamento da mesma de interrupções do mesmo, e processamento destas (kbd_get_pressed_key() e keyboardHandler()). Esta última gera um evento do tipo KEY_PRESS associado à tecla premida e adiciona-o à fila de eventos para ser processado posteriormente.

Intervenção dos membros

Esta abstração foi realizada no Lab 3, pelo que a distribuição do trabalho foi equilibrada. Cada membro realizou 50%.

3.6. Mouse (mouse.c) (10%)

Este módulo contém a abstração para uso do rato, incluindo a subscrição e anulamento da mesma de interrupções do mesmo, e processamento destas (mouse_movement_handler() e mouseHandler()). Esta última gera um evento do tipo MOUSE_CHANGE associado à ação realizada (movimento ou clique) e adiciona-o à fila de eventos para ser processado posteriormente.

Intervenção dos membros

Esta abstração foi realizada no Lab 4, pelo que a distribuição do trabalho foi equilibrada. Cada membro realizou 50%.

3.7. Video Card (video_gr.c e vbe.c) (10%)

Este módulo contém a abstração para uso da placa gráfica, incluindo a mudança para modo gráfico (neste caso é usado o modo vbe 0x117 (1024x768)) e retorno ao modo de texto (vg_init() e vg_exit()), mapeamento da VRAM para mudança da mesma (A VRAM é modificada ao mudar a cor dos píxeis a apresentar)

Intervenção dos membros

Esta abstração foi realizada no Lab 5, pelo que a distribuição do trabalho foi equilibrada. Cada membro realizou 50%.

3.8. Bitmaps (bitmap.c) (5%)

Este módulo contém a abstração para o uso de bitmaps como sprites de objetos. Permite desenhar imagens .bmp no ecrã.
Foi desenvolvido por um ex-aluno de LCOM, Henrique Ferrolho, e nós modificámos algumas partes para garantir a compatibilidade com o nosso projeto, nomeadamente o uso de short para a identificação de cores (no modo 0x117 são usados endereços de 16 bits), e a possibilidade de ter uma cor “invisivel” no .bmp, isto é, uma cor que não é desenhada, permitindo ter sprites não retangulares.

Intervenção dos membros

Henrique Ferrolho - Realização da abstração (http://difusal.blogspot.pt/2014/09/minixtutorial-8-loading-bmp-images.html)
Ângelo Teixeira - Inclusão inicial e modificações (70%)
Henrique Lima - Correção de erros (30%)

3.9. Motor de Jogo (gameEngine.c) (10%)

Neste módulo é onde estão as bases do jogo, enquanto aplicação. Tem uma função inicial, que muda a aplicação para modo gráfico (0x117) e chama a função principal do motor: mainGameLoop(). Nesta função é feito um loop onde a cada ciclo se verificam as interrupções dos periféricos. A cada interrupção do timer são processados os eventos, podendo haver alteração do estado do jogo (gameState_st). Contém também a função processEvents, que processa os eventos, chama a função update(), que realiza operações específicas do jogo desenha o frame resultante que depende do estado do jogo, através da função drawFrame(), também aqui implementada.

Intervenção dos membros

Henrique Lima - Realização da estrutura principal e das funções mainGameLoop(), drawFrame(), e processEvents(). Correção de erros. (80%)
Ângelo Teixeira - Correção de erros e atualizações relativas ao processamento de eventos (20%)

3.10. RTC (rtc.c) (5%)

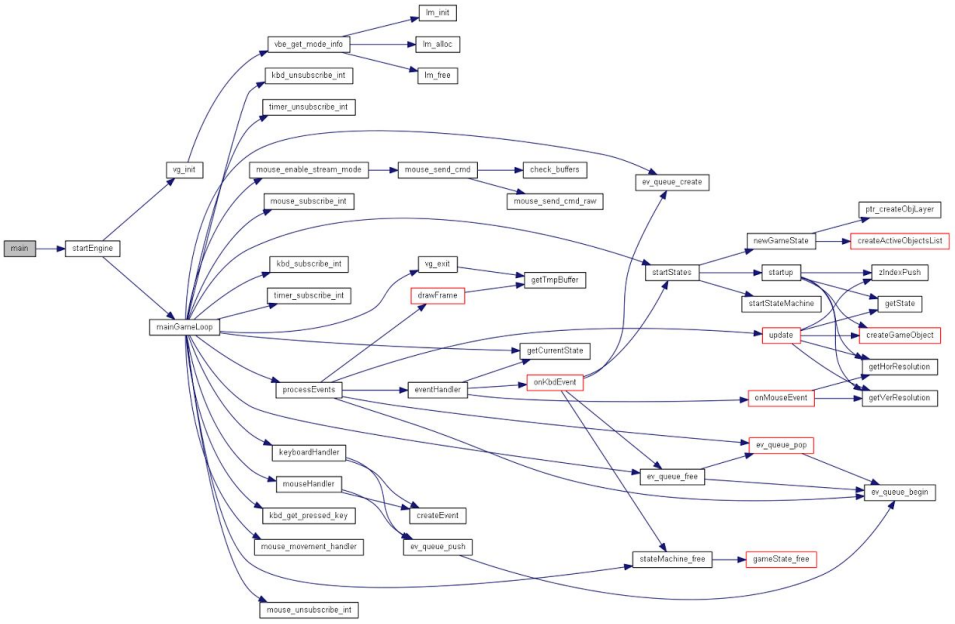
Neste módulo está definida a abstração para obter informações do RTC, nomeadamente o tempo atual. Tem funções para saber as horas, minutos e segundos, embora só seja usada a primeira.

Ângelo Teixeira - Desenvolvimento da Abstração (100%)

3.11. Lógica do Jogo (mainGame.c) (10%)

Neste módulo estão definidas as “regras do jogo”, ou seja, é onde ocorre a atualização das posições dos gameObjects, verificação de colisões, processamento de eventos de teclado e rato específicos (p.ex. O que acontece quando o utilizador pressiona ESC num certo estado). Este módulo contém também a função update(), chamada a cada frame, que atualiza alguma propriedades dos objetos e do jogo.

3.12. Gráfico das Funções



mainGameLoop() - Loop que verifica interrupções e chama os handlers necessários. Processamento de eventos.
startStates() - Inicia a máquina de estados, iniciando a lógica de jogo
startup() - chamada no inicio do jogo (para incializar objetos)
update() - chamada a cada frame, atualiza o jogo.

4. Detalhes da Implementação

No nosso programa usamos um “event driven design”, associado a uma máquina de estados principal, que alterna entre os estados de jogo, consoante os eventos que recebe. Para facilitar a parte gráfica e interativa, temos também objetos de jogo (GameObject) que são guardados em “layers”, sendo mais fácil mostrá-los por ordem de visibilidade no final (objetos na layer mais inferior, são desenhados no inicio, sendo-lhes sobrepostos objetos de outras layers). Este “desenho dos objetos” é feito a cada frame (a cada 1/60 segundos) com a ajuda do timer 0 do Minix, que gera interrupções com essa frequencia, e portanto processamos as interrupções e geramos o frame através de double buffering para minimizar o atraso no display.

Em termos de design do jogo como uma aplicação, é composto por um “engine” e uma “lógica de jogo”. No engine, são recebidas as interrupções, chamam-se os handlers dessas interrupções e processam-se os eventos criados a partir das interrupções. No fim de cada processamento de eventos (cada ciclo) é chamada função update() (da lógica de jogo - pois está relacionada com o jogo em si) e a

função drawFrame(), para desenhar os objetos do jogo no estado em que estão atualmente, depois de terem sido atualizados.
Na lógica de jogo, há uma função startup(), e a função update(), já referida. A função startup() é chamada no início do jogo (mais propriamente ao iniciar a máquina de estados) e cria o ambiente de jogo (os objetos).

Cada estado tem o seu apontador para os objetos, pois os objetos são diferentes em cada estado (no MAIN_MENU vai haver fundo, cursor e botões ao passo que no GAME_PLAY vai haver um outro fundo, a ambulância (principal) e outros carros, por exemplo). Desta forma, evitamos processamento desnecessário.
Os objetos são reutilizados. Por exemplo, a estrada (fundo do GAME_PLAY) são apenas 2 imagens seguidas, que, ao sairem do ecrã, voltam a reaparecer na parte oposta, o mesmo acontece com os carros (obstáculos).

No MAIN_MENU, com o uso do RTC, conseguimos saber as horas atuais, e das 7h às 19h tem um aspeto “diário”, ao passo que a partir das 19h até às 7h, tem um aspeto “noturno”.

5. Conclusões

Este projeto permitiu-nos perceber aprofundadamente o funcionamento do hardware mais comum, bem como formas de utilizar esse conhecimento para criar uma aplicação interativa. Todo este processo demonstrou-nos a utilidade da criação de abstrações para possibilitar um ambiente de desenvolvimento de mais alto nível. Contudo, consideramos necessário salientar que algumas diferenças nas instruções fornecidas para os laboratórios da cadeira relativamente aos restante material da mesma dificultou o processo de desenvolvimento nos mesmos.

6. Apêndice: Instruções de Instalação

Para instalar o programa, com privilégios root no minix, basta executar o script install.sh, seguido de compile.sh, e para correr: run.sh. O primeiro vai colocar a imagens e tudo o que é necessário num local “neutro” e independente da máquina que corre o programa, de forma a que o jogo saiba sempre onde estão, o segundo compila o programa, e o terceiro inicia-o.