FanTale

Projeto realizado para a disciplina de Laboratório de Computadores.

Feito por:

- Diogo Sousa, up201706409 – Turma 2, Grupo 8

Índice

Secção 1 - Instruções para o Utilizador pg. 3

Secção 2 – Estado do Projeto pg. 11

Secção 3 – Estrutura e Organização do Código pg. 15

Secção 4 – Detalhes de Implementação pg. 21

Secção 5 - Conclusões pg. 23

Instruções para o Utilizador

Após correr o programa, o Utilizador começa no menu inicial:

Neste menu existem 3 opções:

- Play, que inicia o jogo;
- How to Play, que leva o utilizador para um menu com instruções sobre como jogar;
- Exit, que termina o programa.

O utilizador usa o rato para mover o cursor no ecrã, para escolher qualquer uma das opções, com o botão esquerdo.

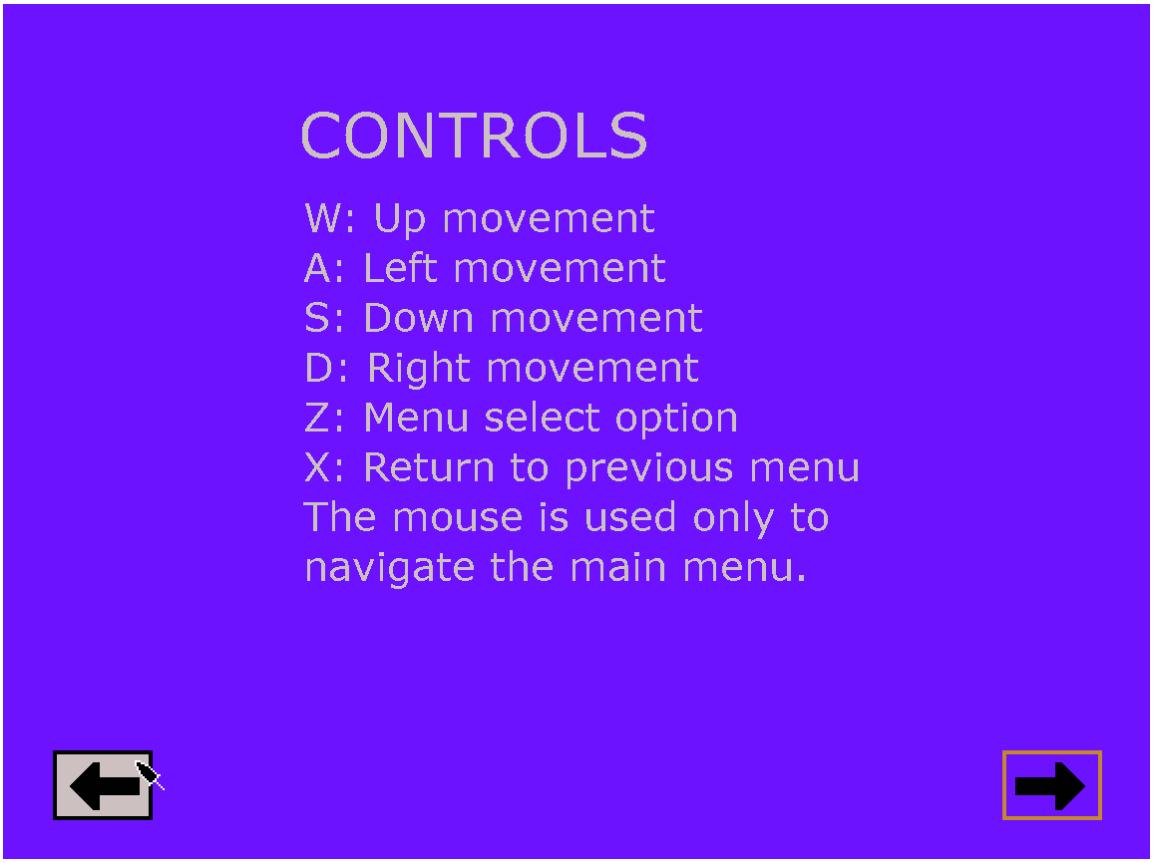
How to Play:

O menu How to Play tem 3 páginas:

- a primeira mostra os controlos do jogo;
- a segunda explica as opções de combate do jogador;
- a terceira explica o objetivo do jogo.

As páginas são navegadas através de 2 botões com imagens de seta no fundo do ecrã. (Clicar na seta de trás na primeira página retorna o jogador ao menu inicial).

Como exemplo, esta é a primeira página:

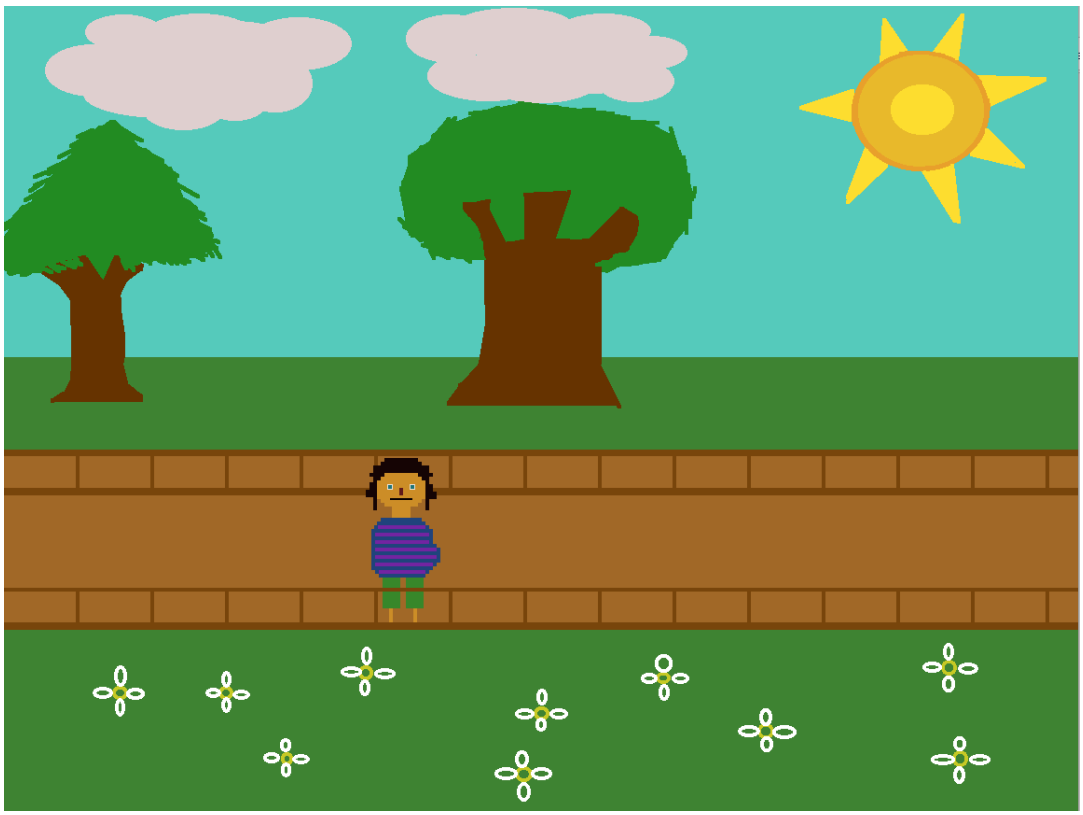


Play:

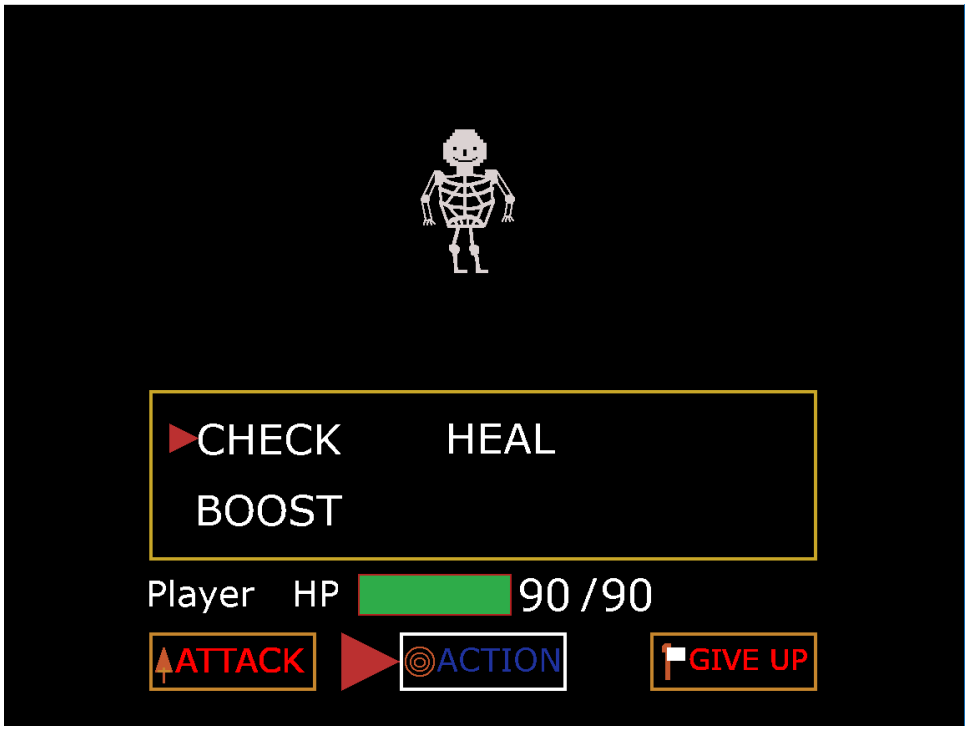
No início do jogo, a personagem é colocada num pequeno ambiente antes da batalha, o jogador tem que movimentar a personagem para o fundo direito do ecrã para começar a batalha. Aqui, o único movimento permitido é horizontal (tecla “A” para mover para a esquerda, e tecla “D” para mover para a direita).

A imagem de fundo muda consoante a hora do dia (através do RTC):

- Se o jogo for iniciado entre as 8:00 e 20:00, a imagem de fundo de dia é carregada;
- Se o jogo for iniciado entre as 20:00 e 8:00, a imagem de fundo de noite é carregada.



Quando a personagem se encontra no final do caminho, o modo de combate é carregado:



Para navegar no menu de combate, muda-se a posição da seta vermelha com as teclas WASD, que respetivamente, movem a seta para cima, esquerda, baixo e direita (quando possível), para mudar a opção escolhida. Existe uma seta para navegar nos menus principais (Attack, Action, e Give Up), e uma seta mais pequena para navegar nos sub-menus (no caso do menu Action, existem 3 opções para escolher no sub-menu).

A tecla Z escolhe uma opção, e a tecla X volta para a opção anterior. Então, por exemplo, se quiser escolher a opção “Heal” em “Action”, tem que se navegar entre os menus principais até a seta principal estar em “Action”, escolher a opção (com a tecla Z), e navegar entre o sub-menu até chegar a opção “Heal”, e escolher a opção. Se dentro do menu “Action” é preciso voltar para trás para ir ao menu “Attack”, usa-se a tecla X.

Opções de combate:

Existem 5 opções que o jogador pode fazer no seu turno:

Attack – Enemy : O jogador ataca o inimigo, e o inimigo faz o seu turno a seguir.

Action – Check: É mostrada a vida do inimigo ao jogador. Esta opção não dá a vez ao inimigo após ser usada.

Action – Heal: O jogador recupera 30 HP, e o inimigo faz o seu turno a seguir.

Action – Boost: O jogador aumenta o dano do seu ataque permanentemente por 2, e o inimigo faz o seu turno a seguir. Só pode ser usada uma vez.

Give Up – Confirm: O jogador desiste da batalha e perde o jogo.

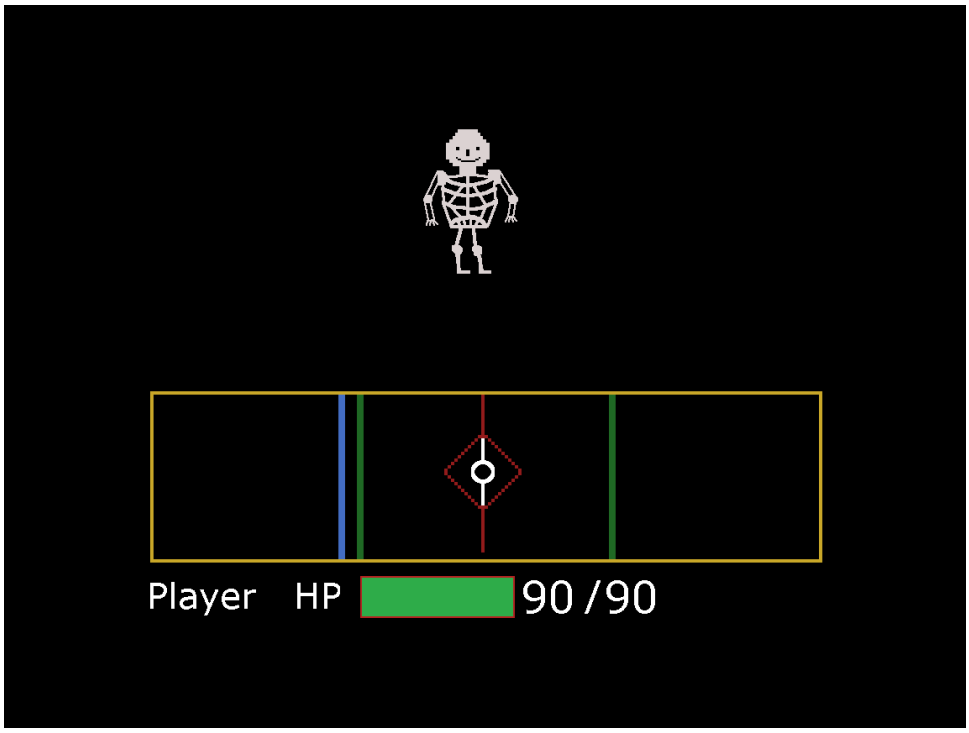
Objetivo do jogo:

O jogo é estilo “bullet-hell” e turn-based, sendo o principal objetivo desviar dos ataques que o inimigo usa no seu turno, e atacando o inimigo no turno do jogador, tentando fazer com que a vida do inimigo chegue a 0 antes que a vida do jogador chegue a 0. O jogo é ganho quando o jogador retira completamente a vida do inimigo, e tem três condições de game over:

- O jogador perde completamente a vida;
- Após o inimigo atacar 20 vezes;
- O jogador desiste da batalha.

Ataques do jogador/inimigo:

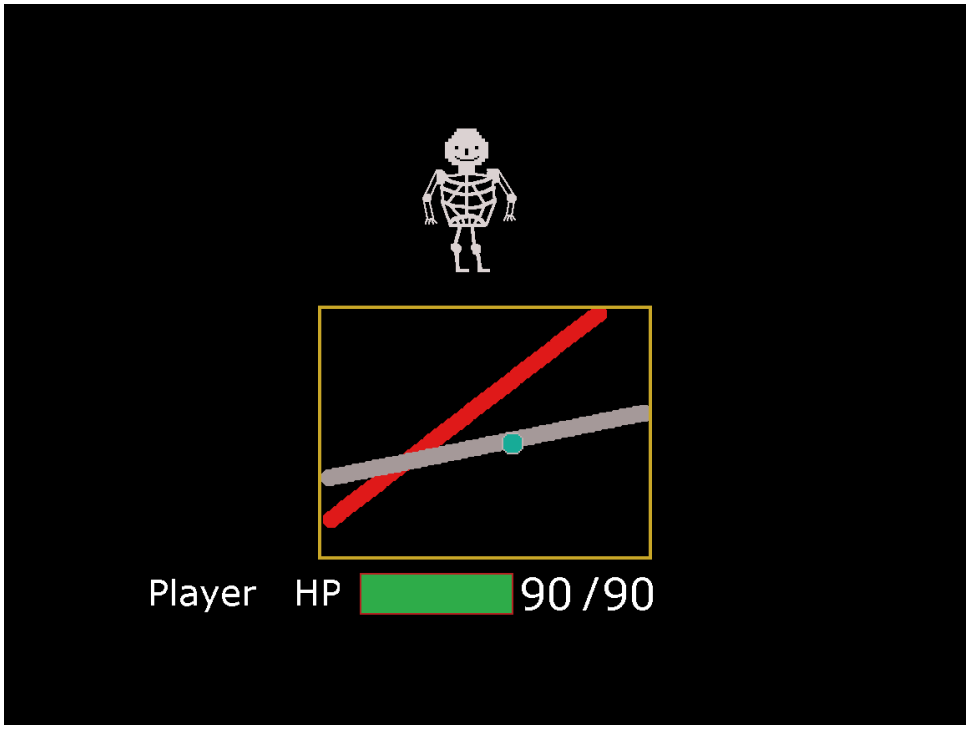
O ataque do jogador consiste numa barra em movimento (que o jogador pode parar a qualquer momento com a tecla Z), a passar entre vários alvos. Quanto mais perto do meio a barra é parada, maior será o dano infligido pelo jogador.



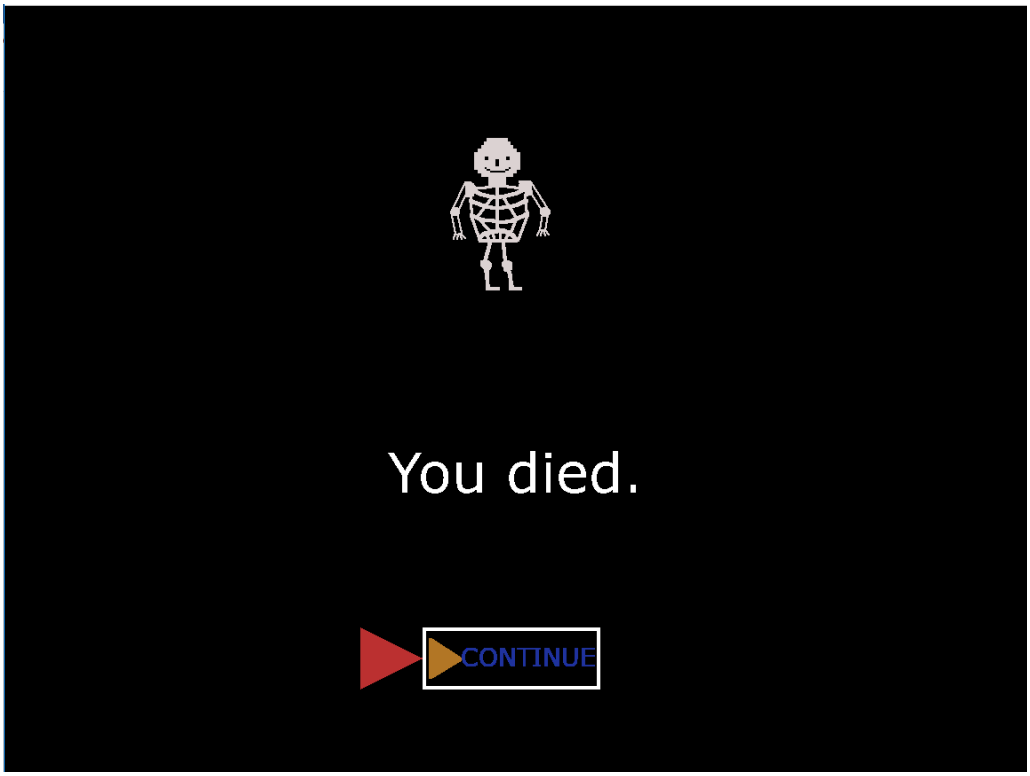
Existem 3 diferentes números de dano (6, 8 e 10):

- 6 de dano: Quando a barra de movimento (azul) é parada entre uma barra verde e uma parede, ou quando a tecla Z não é premida;
- 8 de dano: Quando a barra de movimento é parada entre uma barra verde e o alvo central;
- 10 de dano: Quando a barra é parada no alvo central.

No ataque do inimigo, o jogador controla um objeto azul, limitado a mover-se entre a caixa amarela (usando WASD), e tem que se desviar do padrão de ataque que o inimigo usa (existem 8 padrões diferentes). Certos padrões podem ter 2 cores nos ataques (cinzento e vermelho), O ataque cinzento é um aviso a indicar que o ataque vai cair naquela área, portanto não retira vida ao jogador se ele estiver na área cinzenta. Mas se o jogador colidir com a cor vermelha, ele perde vida.



Game Over:



No final do jogo (após ganhar/perder), o jogador é levado a um ecrã final, que diz o resultado do jogo, e leva o jogador de volta ao menu inicial após confirmação (tecla Z).

Estado do Projeto

A tabela seguinte indica os periféricos usados no projeto, qual o seu objetivo, e se a sua utilização baseia-se em interrupções ou não.

Periféricos	**Objetivos**	**Usa interrupções**
Timer	• Responsável pelo frame rate • Atualiza o desenho/execução do jogo	→ Sim
Keyboard	• Mover a personagem • Navegar no menu de combate	→ Sim
Mouse	• Navegar no menu inicial	→ Sim
Video card	• Desenhar os objetos do jogo no ecrã	→ Não
RTC	• Obter a hora atual	→ Sim (periodic)

Timer:

O timer é utilizado para manter a frame rate do jogo, responsável por atualizar o desenho do ecrã e execução de algumas funções.

Para detetar cada avanço de um frame, são utilizadas interrupções. O jogo corre a 60 frames/segundo (igual a frequência base do timer). A implementação é bastante simples: funções para subscrição de interrupts: *timer_subscribe_int* e *timer_unsubscribe_int*, e um interrupt handler, *timer_int_handler*, que lança um evento à state machine para atualizar o jogo.

Keyboard:

O keyboard é utilizado para mover a personagem, quando possível, e para navegar nas opções do menu de combate. Os controlos são WASD para o movimento da personagem/movimento entre opções, Z para confirmar uma opção no menu, e X para voltar a opção anterior.

Para detetar uma tecla premida, são utilizadas interrupções, com as funções respetivas: *kbc_subscribe_int* e *kbc_unsubscribe_int*. O interrupt handler, *keyboard_int_handler*, usa uma mistura de funções mais simples para ler os bytes do keyboard, e se um byte correspondente aos controlos for detetado, lança um evento à state machine, dependendo do byte encontrado, para processamento.

Mouse:

O mouse é utilizado para navegar no menu inicial, usando o movimento e botão esquerdo, para escolher as opções do menu.

Quando é detetado algum movimento do rato, ou o clique dum botão, é gerada uma interrupção. As funções de subscrição de interrupções são: *mouse_subscribe_int* e *mouse_unsubscribe_int*. Também é necessário ativar o data reporting do mouse após a subscrição de interrupções, sendo usadas s funções *mouse_command* e *read_mouse_ack.* As mesmas funções são também usadas ao terminar o programa, para desativar o data reporting. A função *mouse_ih* usa uma mistura de funções mais simples para ler bytes vindo dos interrupts, construir os packets de mouse de 3 bytes, e, se o packet estiver pronto, lançar o evento relevante à state machine, se o packet conter algum movimento/mudança de left click.

Video card:

O video card é utilizado para desenhar os objetos no ecrã. No programa, a resolução utilizada é **1152x864,** e o modo de cor é de **32** bits per pixel, 8 para cada cor (RGB), e 8 para o alpha channel. O modo que permite esta configuração é o modo **0x14C.**

O programa usa double buffering, objetos que se movimentam, objetos animados, fonts, e deteção de colisões.

A video card é inicializada (mudança de modo) no início do programa, com a função *vg_init,* e para o uso de double buffering, a mesma função também aloca memória para um segundo buffer. No final do programa, é usada a função *vg_exit* para retornar ao modo de texto.

Durante a execução do programa, a cada frame (quando aplicável), por ordem:

- O buffer secundário é limpo através da função *clear_buffer*, ou *clear_combat_buffer* (em combate);
- Os objetos são desenhados no buffer secundário, através da função *draw_sprite* implementada na class Bitmap;
- Os objetos são copiados do buffer secundário para o buffer primário, através da função *buffercopy*;

Para desenhar objetos que se movimentam, é apenas necessário mudar a posição do objeto a cada frame. Como o buffer é limpo a cada frame, não existirão artefactos indesejáveis. Mesma coisa para os objetos animados, só que em vez de mudar a posição, muda–se a imagem.

Em termos de fonts, existe uma imagem para cada número (de 0 a 9), usadas para desenhar a quantidade numérica de vida que o inimigo/jogador tem.

Para deteção de colisões, a função *draw_sprite*, da class Bitmap, verifica se existia algo desenhado na localização onde o novo objeto vai ser desenhado. Isto é usado no modo de combate, se a cor que estava desenhada na localização era vermelha, o jogador perde vida. Também é verificado a colisão do do cursor com os botões nos menus, comparando os valores da posição do cursor com os valores da posição dos botões.

RTC:

O RTC é utilizado para ler a hora (apenas a hora mesmo, não são necessários minutos/segundos), e do RTC é também utilizada a funcionalidade de interrupts periódicos.

As funções de subscrição de interrupts são *rtc_subscribe_int* e *rtc_unsubscribe_int* e para ativar/desativar os interrupts periódicos, é usada a função *rtc_command.* A função *rtc_int_handler* trata do interrupt recebido, e gera um evento à state machine, para atualizar a hora do jogo, através da função *rtc_get_time.* A hora é usada para mudar a imagem de fundo, dependendo se é dia ou noite.

Estrutura e Organização do Código

O código do programa é organizado em diferentes módulos, de acordo com a funcionalidade do código. Existe um módulo para cada periférico, e um módulo para cada classe do jogo.

Nota adicional: Como fiz o projeto sozinho, não vou incluir o segundo parágrafo a dizer qual membro fez o módulo (redundante visto que só estou eu no grupo).

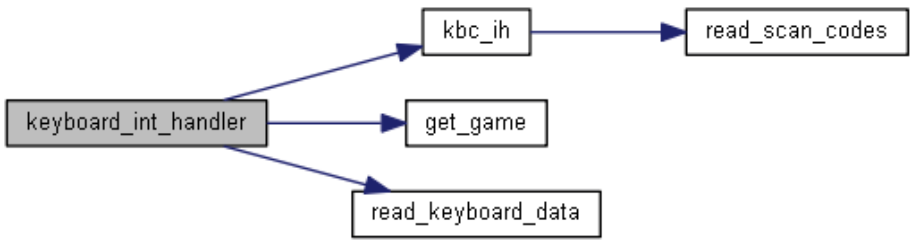
Módulos de Periféricos:

Timer:

O módulo do timer tem funções para lidar com interrupts (subscrição e handler). Módulo muito simples, peso relativo apenas de 2%.

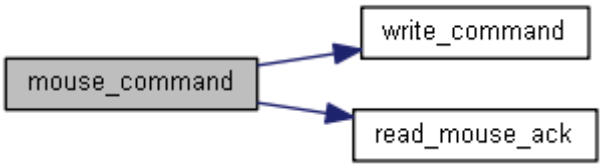
Keyboard:

O módulo do keyboard tem funções para lidar com interrupts (subscrição e handler, que lê o byte proveniente do keyboard) e funções de escrita e leitura do KBC. Como exemplo, aqui está o call graph do interrupt handler chamado no interrupt loop. Peso relativo de 4%.



Mouse:

O módulo do rato tem funções para lidar com interrupts (subscrição e handler, que lê o byte proveniente do rato) e funções de escrita e leitura do KBC (tem alguma reutilização do módulo do keyboard). Exemplo de call graph de uma escrita no KBC (inclui uma função do módulo do keyboard). Peso relativo de 4%.



Video card:

O módulo da placa gráfica tem funções para lidar com o ecrã do MINIX, através de funções para mudar o modo de vídeo, alterar o buffer secundário, copiar os conteúdos do buffer secundário para o primário, e tem funções úteis para o desenho de imagens. Peso relativo de 4%.

RTC:

O módulo do RTC tem funções para lidar com interrupts periódicos (subscrição e handler, que lê a hora atual e atualiza a hora gravada no jogo a cada interrupt), e funções de escrita e leitura dos endereços de memória do RTC. Exemplo de call graph do interrupt handler. Peso relativo de 4%.

Módulos do Jogo

Bitmap:

O módulo bitmap tem funções que permitem ler bitmaps, e copiar o seu contéudo para um objeto do tipo Sprite, que também se encontra definido neste módulo. A classe Sprite contém a imagem de um bitmap, e a posição onde se encontra (nos eixos x e y). O módulo também tem funções para manipular objetos do tipo Sprite. Peso relativo de 5%.

O código para carregar os bitmaps foi encontrado na internet, no seguinte link: https://stackoverflow.com/questions/14279242/read-bitmap-file-into-structure

Este código em questão abre um ficheiro bitmap, e copia o conteúdo relevante para uma struct gravada no programa. No código original, apenas copiava a imagem em si, mas adicionei a funcionalidade de copiar a largura e a altura também. O código original também tinha algumas coisas extra desnecessárias no final, que foram removidas. Finalmente, mudei os nomes de algumas das variáveis e dos seus tipos também.

Button:

O módulo button tem funções que permitem criar objetos com funcionalidade de botões, e manipular objetos desta classe, Button. A classe Button consiste em 2 imagens (uma imagem para o botão em estado normal, outra para mostrar quando o botão está seleccionado), posição x/y, e um valor que indica se o botão está seleccionado ou não. Peso relativo de 4%.

Character:

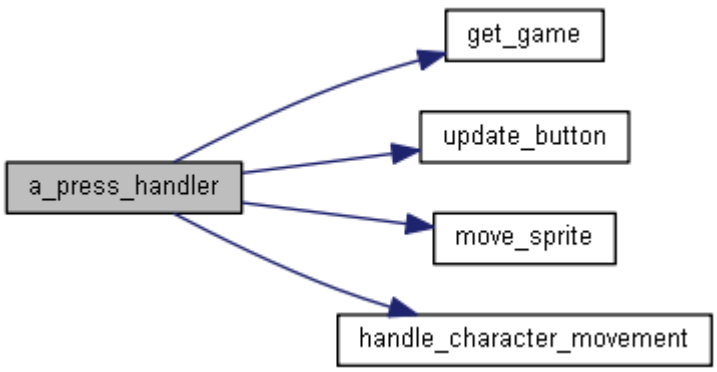
O módulo character tem funções que permitem criar objetos para representar a personagem jogável e o inimigo, e manipulá-los. A classe Character consiste em um objeto do tipo Sprite (imagem + posição), um valor que representa a vida restante da personagem, e um valor que representa o turno atual da personagem no combate (usado para determinar ataques inimigos). Peso relativo de 4%.

Cursor:

O módulo cursor tem funções que permitem criar o objeto do cursor, utilizado para navegar no menu inicial, e manipulá-lo. A classe Cursor consiste simplesmente de um objeto do tipo Sprite, e um valor a indicar se o botão esquerdo está premido ou não. Peso relativo de 3%.

Dispatcher:

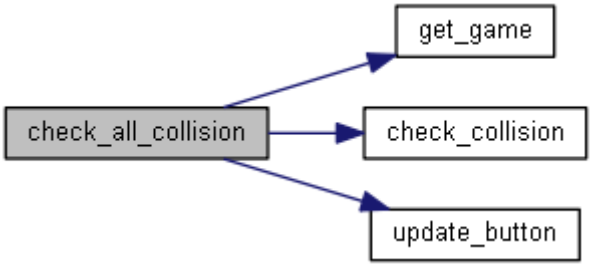
O módulo dispatcher tem funções que permitem criar a state machine do jogo, sendo o motor por trás do jogo, que deteta os eventos chamados através dos interrupt handlers, e chama as apropriadas funções baseado no evento recebido, e no estado em que o jogo atualmente se encontra. Tem-se aqui o call graph de um event handler, que faz o que se deve esperar no jogo, após a deteção da tecla A (controlo para andar para a esquerda), o jogo vai atualizar o movimento da personagem (se estiver no estado apropriado), ou vai atualizar os botões do menu de combate (novamente, se estiver no estado apropriado). Peso relativo de 15%.



Fantale:

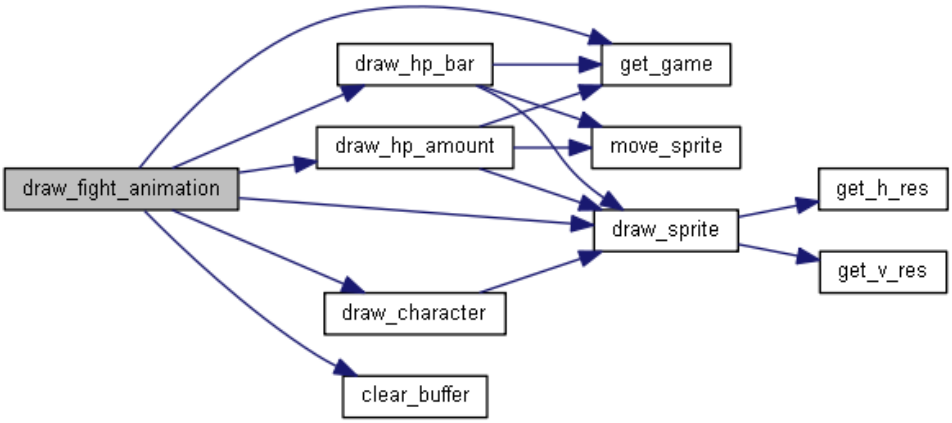
O módulo fantale é o módulo principal do programa. É responsável pela criação da classe FanTale, a classe principal do jogo que contém cada um dos objetos, de diversas classes diferentes, necessários ao seu funcionamento. Também contém todas as funções que são chamadas pela state machine, que tratam da 'jogabilidade' do jogo em si. É neste módulo também que se encontra a verificação de interrupts em loop, na função *fantale_loop.*

Exemplo de call graph de uma função chamada pela state machine:

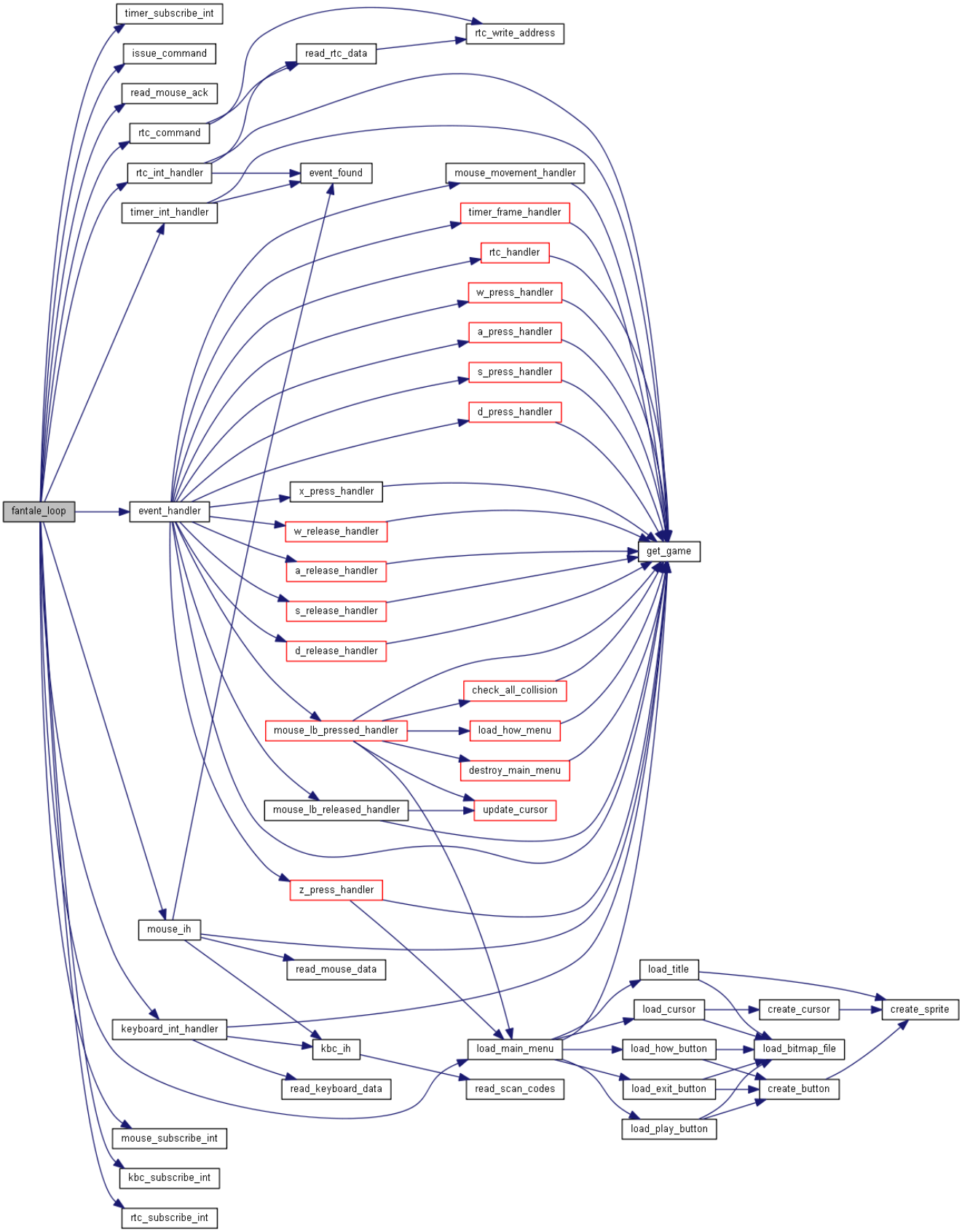


Esta função compara o objeto do cursor com todos os objetos de botões no menu inicial/menu de instruções, para saber se deve mudar a imagem ou mudar de menu dependendo de onde se encontra o cursor.

Call graph de função de desenho do ataque do jogador:



Call graph da função onde se encontra o *driver_recieve* (*fantale_loop*):



Como o módulo fantale é o principal, e tem imensas linhas de código, tem um peso relativo de 50%.

Proj:

O módulo proj contém o ponto de começo do programa (função *main* com *lcf_start).* Chama as 3 funções do módulo fantale para a execução do jogo: a criação da classe FanTale, o interrupt loop, e quando o jogo é terminado, e a destruição da classe FanTale. Este módulo tem também uma funcionalidade importante – guarda a classe FanTale criada como variável global, e tem uma função que a retorna, usada em basicamente quase todas as funções do programa que necessitam da classe em questão. Peso relativo de 1%.

Macros:

O objetivo deste módulo era guardar macros que fossem úteis para diversos módulos do programa, para evitar problemas com macros definidos mais que uma vez nas dependências, mas só acabou por ter um, para facilitar a manipulação de bits/bytes. Visto que o módulo é extremamente básico (só tem uma macro definida), não contei este módulo para o peso relativo.

Peso relativo de todos os módulos:

Timer – 2%

Keyboard – 4%

Mouse – 4%

Video card – 4%

RTC – 4%

Bitmap – 5%

Button – 4% **TOTAL:** 100%

Character – 4%

Cursor – 3%

Dispatcher – 15%

Fantale – 50%

Proj – 1%

Macros – 0%

Detalhes de Implementação

O código do projeto (e também dos laboratórios), foi feito com atenção as várias recomendações dadas durante as aulas, notavelmente o design por camadas, tornando o código muito mais legível/fácil de entender, e principalmente, facilitando imenso a adição de novas funcionalidades ao programa, se já começamos com uma camada por baixo ao criar a nova funcionalidade.

A máquina de estados criada no laboratório 4, do rato, também ajudou a fazer algo semelhante neste projeto, sendo o motor por trás de todos as funções do jogo. A máquina de estados implementada permite que o programa seja orientado a eventos. Cada evento detetado após uma interrupção de um periférico gera uma entrada num array, este array que, após cada ciclo do *driver_recieve* loop, é processado, e cada evento encontrado gera uma chamada ao apropriado event handler. Cada event handler, por sua vez, chama funções diferentes dependendo do estado em que o jogo se encontra. Por exemplo, clicar na tecla A no turno do inimigo (estado atual), mexe a personagem para a esquerda em combate, mas clicar na tecla A no turno do jogador (estado atual), irá para a opção à esquerda no menu de combate.

O código é também um pouco orientado a objetos, tendo diversos módulos para cada classe necessária ao funcionamento do programa. No entanto, não foi feita a encapsulação dos conteúdos da classe, visto que é possível aceder diretamente a qualquer elemento de qualquer classe em qualquer parte do programa, que contenha o módulo dessa respetiva classe. O jogo em si é operado como uma classe que contém objetos de todas as outras classes também, e é nesta classe do jogo (classe FanTale) que são feitas quase todas as operações a objetos no jogo.

Era para ser utilizado algum código assembly, mas devido à falta de tempo, este objetivo não foi cumprido. Também, não acho que era necessário, visto que o jogo corre bem e não precisa muito de mais eficiência.

A frame rate do jogo é regulada pelo timer. Como o timer tem uma frequência base de 60 Hz (e como 60 FPS tende a ser o standard aceitado em jogos), o programa simplesmente desenha tudo no buffer secundário, e depois copia do buffer secundário para o ecrã, a cada interrupt.

A implementação do RTC foi bastante simples, visto que o programa necessita de poucas funcionalidades do mesmo. Apesar de não ter sido feito um lab no RTC, foi simples de entender e desenvolver.

A deteção de colisões provou não ser muito difícil também, apesar de não ter sido mencionado muito nas aulas. Foram usados dois tipos de deteção de colisão no programa. O primeiro é usado para detetar se o cursor se encontra em sobreposição com algum botão (fácil de executar, verificar se o x/y do cursor encontram-se dentro do x/y iniciais e x/y finais do botão em questão). O segundo é usado para detetar se a personagem, durante o turno do inimigo, está em sobreposição com um ataque cinzento (aviso de ataque iminente, em que a personagem não deve perder vida), ou outro ataque, geralmente vermelho (personagem deve perder vida).

Realmente, podia-se ter usado o primeiro método, mas teria que ser especializado para cada ataque diferente. Como o fundo do combate onde a personagem se mexe é preto (tudo a 0), pode-se utilizar uma forma mais geral. Se a personagem é desenhada em cima de algo que já existe (ataque vermelho), há uma colisão e a personagem perde vida. Existe também um check para ver se a imagem por baixo da personagem é da cor específica cinzenta. Neste caso, há colisão, mas a personagem não perde vida.

Conclusões

A cadeira de LCOM foi a primeira cadeira que tivemos completamente baseada em trabalhos. De certa forma, acho que a estrutura da cadeira assim é interessante, pois obriga os alunos a trabalhar mais, e a saber realmente a matéria. Também gostei do facto que os pequenos trabalhos que andamos a fazer durante o semestre, combinam todos num grande projeto no final.

No entanto, como é de esperar de uma cadeira que é feita desta maneira, acho que LCOM é em geral, muito stressante. Não tenho necessariamente ideias para melhorar este aspeto, mas quando uma cadeira exige um trabalho a ser entregue a cada 2 semanas, baseado num sistema operativo não muito usado (difícil de esclarecer dúvidas por vezes, apesar dos professores serem bastante responsivos no fórum), mais um projeto em que é necessário entregar bastantes coisas, são muitos objetivos por cumprir, que pode tornar-se chato rápido.

De resto, acho que aprendi bastante com esta cadeira, não só programação baseada em hardware, mas também o uso de plataformas para projetos em grupo, e também o processo de programar em grupo (apesar de não ter utilizado muito este último, devido ao meu parceiro ter desistido logo no início).