

**Mestrado Integrado em Engenharia Informática e Computação**  
**2º Ano**

# **“Planetary Defense”**

**Turma 4 – Grupo 1**

André Cruz – up201503776@fe.up.pt

Edgar Carneiro – up201503784@fe.up.pt

# Índice

<b>1 Instruções de Utilização do programa .....</b>	<b>5</b>
<b>1.1 Menu Inicial .....</b>	<b>5</b>
<b>1.1.1 Botão “Single Player” .....</b>	<b>5</b>
<b>1.1.2 Botão “Multi Player” .....</b>	<b>6</b>
<b>1.1.3 Botão “High Scores” .....</b>	<b>6</b>
<b>1.1.4 Botão “X” .....</b>	<b>6</b>
<b>1.2 Single Player.....</b>	<b>7</b>
<b>1.3 Multi Player.....</b>	<b>9</b>
<b>1.4 End of Game Menu/Animation.....</b>	<b>10</b>
<b>1.5 High Scores.....</b>	<b>11</b>
<b>2 Estado do Projeto .....</b>	<b>12</b>
<b>2.1 Dispositivos Utilizados .....</b>	<b>12</b>
<b>2.2 Timer .....</b>	<b>13</b>
<b>2.3 Rato.....</b>	<b>14</b>
<b>2.4 Teclado.....</b>	<b>15</b>
<b>2.5 Placa Gráfica .....</b>	<b>16</b>
<b>2.6 RTC (<i>Real Time Clock</i>).....</b>	<b>17</b>
<b>2.7 Porta Série.....</b>	<b>18</b>
<b>3 Organização e estrutura do código .....</b>	<b>19</b>
<b>3.1 Bitmap.....</b>	<b>19</b>
<b>3.2 BMPsHolder .....</b>	<b>19</b>
<b>3.3 Communication.....</b>	<b>19</b>
<b>3.4 GVector .....</b>	<b>19</b>
<b>3.5 Highscores.....</b>	<b>20</b>
<b>3.6 i8042 .....</b>	<b>20</b>
<b>3.7 i8254 .....</b>	<b>20</b>
<b>3.8 Input.....</b>	<b>20</b>
<b>3.9 keyboard .....</b>	<b>20</b>
<b>3.10 main .....</b>	<b>20</b>
<b>3.11 Missile .....</b>	<b>21</b>
<b>3.12 mouse .....</b>	<b>21</b>
<b>3.13 planetary .....</b>	<b>21</b>
<b>3.14 rtc_asm .....</b>	<b>21</b>

3.15	RTC .....	21
3.16	Serial .....	22
3.17	timer .....	22
3.18	VBE .....	22
3.19	video_gr .....	22
3.20	<i>Function Call Graph</i> .....	23
4	Detalhes de Implementação .....	25
5	Conclusões .....	26
5.1	Avaliação da Unidade Curricular.....	26
5.2	Apêndice .....	27

## Tabela de Figuras

Figura 1 - menu inicial .....	5
Figura 2 - botão "singleplayer" normal / selecionado.....	5
Figura 3 - botão "multiplayer" normal / selecionado.....	6
Figura 4 - botão "highscores" normal / selecionado .....	6
Figura 5 - botão "X" .....	6
Figura 6 - base destruída .....	7
Figura 7 – decorrer de uma partida no modo singleplayer .....	7
Figura 8 - base deteriorada .....	7
Figura 9 - base inicial .....	7
Figura 10 - excerto da "spritesheet" da explosão .....	8
Figura 11 - "score" .....	8
Figura 12 - modo multiplayer - à espera de outro jogador .....	9
Figura 13 - animação de final de jogo .....	10
Figura 14 - highscores .....	11

# 1 Instruções de Utilização do programa

## 1.1 Menu Inicial

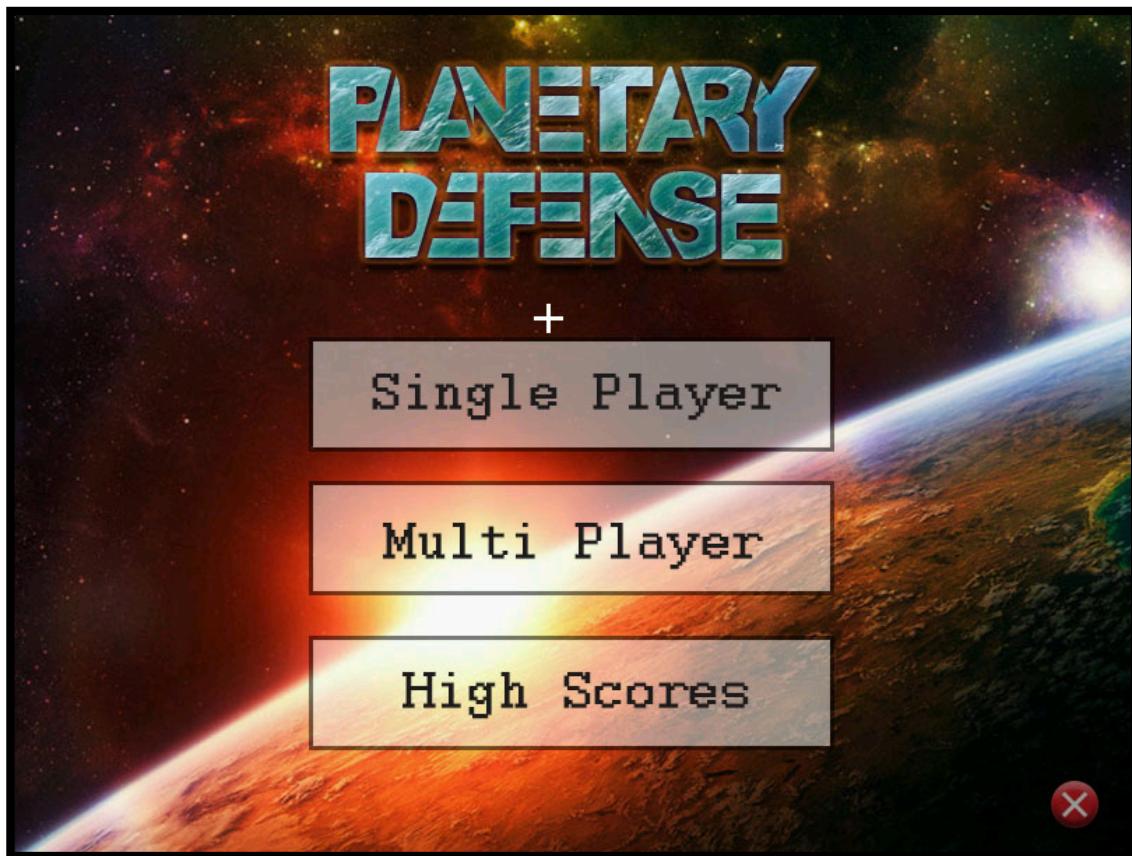


Figura 1 - menu inicial

O menu inicial surge quando se inicia a aplicação, permitindo o uso do rato (através do movimento e dos botões), bem como do teclado (através das teclas cima/baixo, *enter* e *escape*).

### 1.1.1 Botão “Single Player”

Este botão permite iniciar uma nova sessão de jogo no modo *singleplayer*.

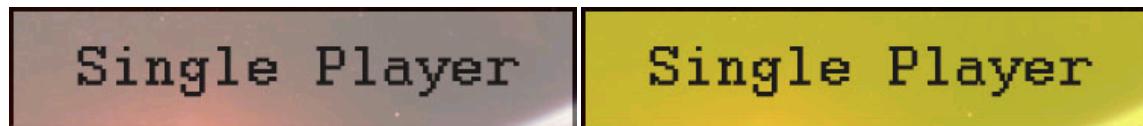


Figura 2 - botão "singleplayer" normal / selecionado

#### 1.1.2 Botão “Multi Player”

Este botão permite iniciar uma nova sessão de jogo no modo *multiplayer*.



Figura 3 - botão "multiplayer" normal / selecionado

#### 1.1.3 Botão “High Scores”

Este botão permite entrar na opção *highscores*, referida em maior detalhe mais à frente.



Figura 4 - botão "highscores" normal / selecionado

#### 1.1.4 Botão “X”

Este botão tem funcionalidade igual à tecla *ESC*, permitindo sair da aplicação.



Figura 5 - botão "X"

## 1.2 Single Player

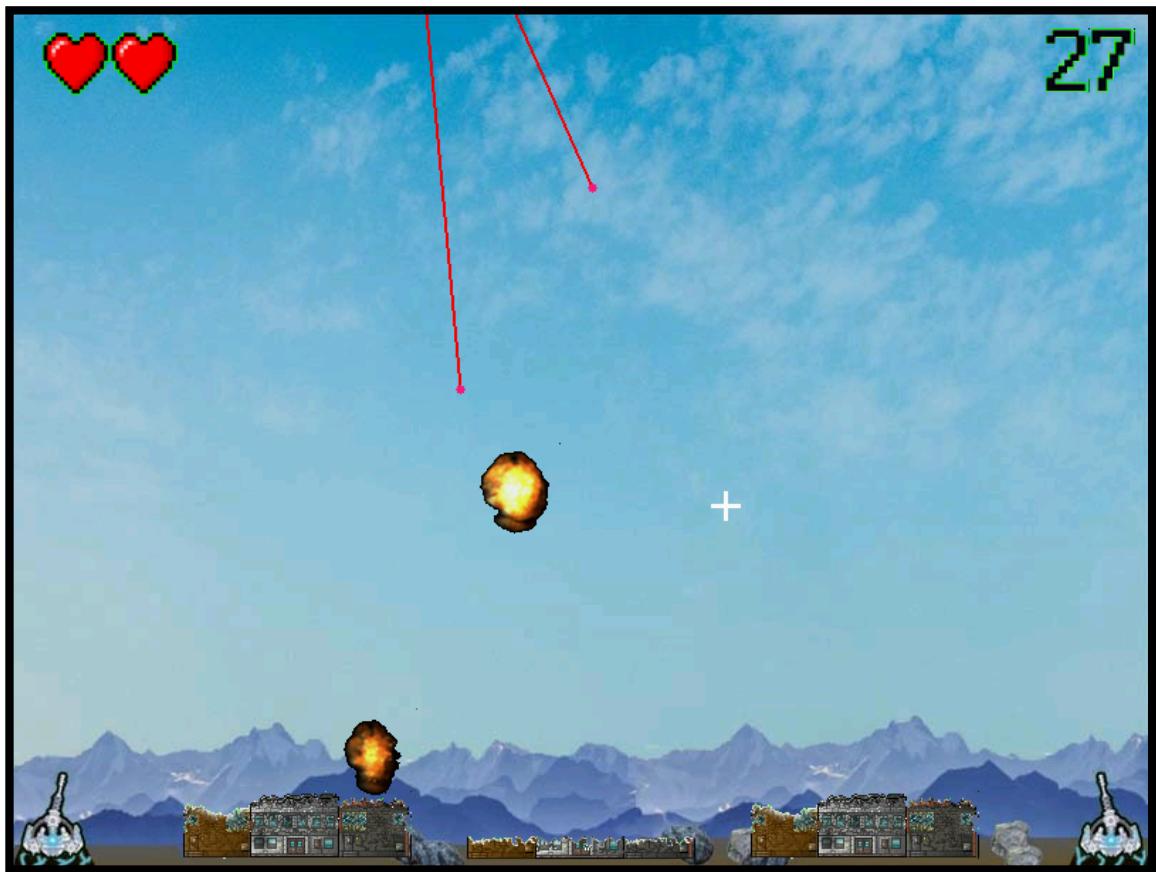


Figura 7 – decorrer de uma partida no modo singleplayer

No modo *singleplayer* o jogador tenta resistir o máximo de tempo possível ao ataque de que é alvo. Para este efeito tem ao seu dispor dois canhões de mísseis, que permitem o lançamento de mísseis através do botão do lado esquerdo (canhão esquerdo) e botão do lado direito (canhão direito), até um máximo de cinco mísseis no ecrã.

Internamente, os mísseis inimigos (de tonalidade vermelha) selecionam uma base para atacar aleatoriamente, de entre as bases do jogador que ainda têm vida (*health points*). Quando uma base é atingida a sua condição é deteriorada até chegar ao último estado de destruição (quando este estado for atingido o jogador perde uma das suas vidas, representadas pelos corações no canto superior esquerdo).

Quando um míssil inimigo colide com o chão, ou com uma das bases (cuja caixa de colisão é aproximada por um retângulo) este origina uma explosão, que é uma animação composta por 16 *sprites* em rápida sequência. No caso de um míssil “amigável”, explode quando atinge a posição onde estava o rato aquando da sua criação.



Figura 6 - base destruída



Figura 8 - base deteriorada



Figura 9 - base inicial

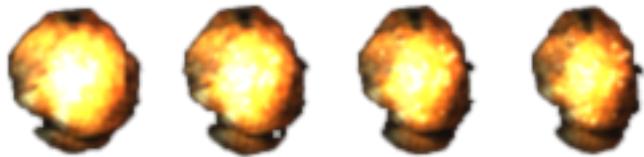


Figura 10 - excerto da "spritesheet" da explosão



Figura 11 - "score"

Aquando da explosão de um míssil, quer seja inimigo ou amigável, é sempre originada uma explosão (cuja área de colisão é aproximada por um círculo), sendo que todos os mísseis que a atravessem originam novas explosões. Este facto acrescenta uma nova dinâmica ao jogo, sendo possível que mísseis (quer os nossos, quer os inimigos) sejam inadvertidamente intercetados por explosões, sendo também possível criar explosões de vários mísseis próximos em cadeia.

Achamos oportuno também referir que a dificuldade do jogo aumenta com o tempo, sendo o intervalo entre ataques inimigos modulado por uma função matemática estritamente decrescente.

Após o jogador perder a totalidade das suas vidas é automaticamente retirado do modo *singleplayer* e passa para a animação de final de jogo, vista em detalhe mais à frente. O jogador pode também regressar ao menu inicial a qualquer momento premindo a tecla *ESC*.

### 1.3 Multi Player



Figura 12 - modo multiplayer - à espera de outro jogador

No modo *multiplayer* os jogadores têm também que resistir ao ataque de que são alvos, no entanto apenas têm que resistir mais tempo que o outro jogador para serem vencedores.

Ao clicar no botão *multiplayer*, o primeiro computador inicia um modo de espera, que só termina quando receber a comunicação correspondente do segundo computador (sinalizando o início da partida). Quando um jogador é finalmente derrotado o seu computador mostra a mensagem “Perdeu...” e envia informação ao outro computador que, correspondentemente, o faz mostrar a mensagem “Ganhou!”.

#### 1.4 End of Game Menu/Animation



Figura 13 - animação de final de jogo

No final de um jogo é verificado se a pontuação do jogador pertence às 5 melhores pontuações registadas e, em caso afirmativo, é disposta a mensagem “HIGHSCORE！”, sendo a sua pontuação exposta no centro do ecrã a piscar. O score é de seguida registado num ficheiro, bem como a data e hora a que foi atingido. Todos os mísseis ainda em jogo explodem e são também criadas explosões em posições aleatórias a cada 0.5 segundos.

A aplicação regressa automaticamente ao menu inicial ao fim de 10 segundos, no entanto o jogador pode premir as teclas *ESC* ou *ENTER* para apressar a animação.

## 1.5 High Scores

Score	Hour	Date
108	14 22	30 12 2016
60	12 41	30 12 2016
57	17 17	30 12 2016
51	11 45	30 12 2016
42	12 19	30 12 2016

Figura 14 - highscores

Acima está representado o menu das pontuações mais elevadas (*highscores*). Este menu permite o uso do rato para regressar ao menu inicial, premindo o botão no canto inferior direito, bem como o uso da tecla *ESC*.

## 2 Estado do Projeto

### 2.1 Dispositivos Utilizados

Dispositivo	Utilização	Interrupção?
Timer	Atualizar o ecrã e regular a <i>frame-rate</i> .	Sim
Rato	Navegação nos menus e interação com a aplicação.	Sim
Teclado	Navegação nos menus e interação com a aplicação.	Sim
Placa Gráfica	Desenho no ecrã dos menus e botões.	Não
RTC	Obtenção da data e hora de jogo, para registo nas melhores pontuações.	Não
Porta Série	Envio de informação acerca do início e fim de um jogo, no modo multiplayer (não acabado)	Sim

## 2.2 Timer

O *timer*, mais precisamente o *timer 0*, é utilizado para atualizar o estado do jogo a uma taxa constante. Para este efeito está configurado para gerar interrupções sessenta vezes por segundo, sendo chamado a função *timer\_handler* a cada interrupção.

## 2.3 Rato

O rato serve de interface de comunicação entre o utilizador e a aplicação, sendo usado em todos os estados da aplicação. Para este efeito permite o clique e seleção de botões nos menus, bem como o lançamento de mísseis no jogo.

Para facilitar a sua utilização, o funcionamento deste periférico (movimento e botões) foi abstraído através da *struct Input\_t*, sendo esta *struct* amplamente usada na atualização do estado do jogo (no *timer\_handler* e nas funções chamadas por este).

## 2.4 Teclado

O uso deste periférico restringe-se às teclas *ESC*, *ENTER*, *CIMA* e *BAIXO*, sendo, portanto, compatível com *make/break codes* de 1 ou 2 bytes. No entanto, o uso de mais teclas seria simples, sendo apenas necessário acrescentar o seu *make/break code* à *enum keycode\_t*, estando o seu funcionamento abstruído pela *struct Input\_t* e os seus métodos.

## 2.5 Placa Gráfica

A placa gráfica é utilizada no modo RGB 5:6:5 com resolução 800 pixéis de largura por 600 pixéis de altura (modo de código 0x114), sendo usada para expor imagens no ecrã (imagens essas criadas ou editadas por nós). No entanto, o código necessário para desenhar, carregar e apagar *Bitmaps* é da autoria de Henrique Ferrolho, estando disponível no seu [blog](#). É de salientar que alteramos o algoritmo de desenhar *Bitmaps* para refletir as nossas necessidades no projeto, nomeadamente reservamos uma cor para simular um canal de transparência (verde – `rgb 0:255:0`), sendo, por este motivo, esta a cor de fundo da maioria das nossas imagens.

Fazemos também uso da técnica *double buffering* de modo a fornecer uma experiência visual mais agradável ao utilizador. Deste modo evitamos que o utilizador assista ao “piscar” de elementos ou artefactos visuais (*tearing*).

A deteção de colisões é feita com recurso a aproximações das imagens por figuras geométricas mais simples, nomeadamente por retângulos (bases), círculos (explosões) e pontos (mísseis). Com isto em mente o grupo desenvolveu algoritmos que verificam colisões entre as necessárias formas geométricas.

Implementamos também a animação de explosões, através de 16 *sprites* em rápida sucessão. Bem como um algoritmo para desenhar números no ecrã a partir de uma determinada fonte (fornecida como um array de 10 bitmaps – 1 para cada algarismo).

## 2.6 RTC (*Real Time Clock*)

O RTC (relógio em tempo real) é utilizado para registar a data e hora a que as melhores pontuações foram atingidas (pontuações essas que ficam armazenadas num ficheiro de texto). Para este efeito não necessitamos de interrupções, sendo apenas útil a leitura da data e hora atual. Para facilitar este processo criamos também a *struct Date\_t*, que encapsula os dados de um ponto temporal (hora, minutos, dia, mês e ano).

## 2.7 Porta Série

Planeamos em usar este periférico para a comunicação entre computadores no modo *multiplayer* (interrupções), no entanto este modo de jogo ficou apenas parcialmente implementado, não tendo sido possível uma implementação completa a tempo da entrega

### 3 Organização e estrutura do código

#### 3.1 Bitmap

Contém as funções necessárias para carregar, desenhar e apagar imagens no ecrã, em ficheiros do tipo Bitmap (rgb 5:6:5). Código da autoria de Henrique Ferrolho e disponível no seu [blog](#), alterado de modo a permitir imagens com transparência (para este efeito foi reservada a cor verde, de código rgb 0:255:0).

As imagens usadas no nosso projeto são todas editadas ou criadas por nós, usando os programas de edição de imagens “Photoshop” e “GIMP”, sendo o crédito pela maioria das imagens devido ao elemento Edgar Carneiro.

As alterações a este código foram desenvolvidas unicamente pelo elemento Edgar Carneiro. Peso: 3%.

#### 3.2 BMPsHolder

Organiza o uso de *Bitmaps* (*bmps*) ao longo do projeto. Foi criada a *struct* *BMPsHolder\_t* para armazenar todos os *Bitmaps* usados no projeto, sendo a maior parte de uso comum a vários estados da aplicação e, portanto, a várias funções.

Foi também criada uma função para carregar uma sequência de *Bitmaps* de *path* comum, para facilitar o carregamento das imagens correspondentes a animações, e fontes de números. Por exemplo, as imagens da explosão animada diferem apenas nos últimos dois caracteres (.../res/Explosion/00.bmp, ..01.bmp, ...02.bmp, etc.).

Esta *struct* implementa comportamento semelhante a uma classe *singleton*. Para este efeito tem o seu construtor definido apenas no ficheiro .c e precedido pela keyword “static”, de modo a limitar o número de instâncias a uma. Para aceder ao único objeto da *struct* foi definido um método que, caso esse objeto esteja instanciado, devolve-o, caso não esteja instanciado cria-o e devolve este novo.

Este módulo foi desenvolvido unicamente pelo elemento André Cruz. Peso: 5%.

#### 3.3 Communication

Facilita a comunicação entre computadores através da porta série, sendo, portanto, usado no modo de jogo *multiplayer*. Foi definida a *enum* *serial\_state\_t* para facilitar a implementação de uma máquina de estados, usada no referido modo multijogador.

Este módulo foi desenvolvido unicamente pelo elemento André Cruz. Peso: 1%.

#### 3.4 GVector

Encapsula a estrutura de dados vetor, permitindo que o programador se abstraia da alocação dinâmica de memória essencial à programação em C. Foi desenvolvido para responder à necessidade que sentimos em usar *vetores* de C++ (por estarmos mais habituados a esta linguagem). Todo o módulo foi feito de modo a permitir programação *Object Oriented* em C (tal

como explicado nas aulas teóricas). A *struct GVector* aqui definida encapsula, portanto, um vetor genérico (correspondente a *templates* de C++), tendo métodos privados para a realocação de memória caso a diferença entre a capacidade (memória alocada) e o tamanho (*size* – número de elementos) seja diminuta ou em excesso.

Este módulo foi desenvolvido unicamente pelo elemento André Cruz. Peso: 12%.

### 3.5 Highscores

Permite a obtenção de informação sobre as cinco melhores pontuações, bem como a data a que foram atingidas. Essas pontuações são guardadas num ficheiro de texto, cujo *input/output* de informação está também implementado neste módulo. Para facilitar este processo criou-se a *struct Score\_t*, que mantém toda a informação associada a uma pontuação.

Este módulo foi desenvolvido unicamente pelo elemento Edgar Carneiro. Peso: 6%.

### 3.6 i8042

Contém *macros* essenciais para o funcionamento do teclado e rato.

Código desenvolvido nas aulas práticas do laboratório 3.

### 3.7 i8254

Contém *macros* essenciais para o funcionamento do *timer*.

Código desenvolvido nas aulas práticas do laboratório 2.

### 3.8 Input

Contém funções que mantêm o estado dos periféricos de *input* do utilizador (rato e teclado), e abstraem o seu funcionamento. Engloba também a *struct Input\_t* e a *enum keycode\_t*.

Código desenvolvido igualmente por ambos os elementos do grupo. Peso: 6%.

### 3.9 keyboard

Contém o conjunto de funções essenciais para o funcionamento do teclado.

Código desenvolvido aquando do laboratório 3.

### 3.10 main

Inicializa a aplicação e trata todas as interrupções recebidas (através do *loop driver\_receive* único). Essencialmente evoca o *handler* apropriado para cada interrupção.

Código desenvolvido por ambos os elementos do grupo. Peso: 1%.

### 3.11 Missile

Contém todas as funções que permitem a criação, atualização e destruição de mísseis e explosões. Para este efeito criaram-se as structs `Missile_t` e `Explosion_t`, acessíveis apenas pelos métodos disponíveis, de modo a garantir a estabilidade do projeto.

Este módulo foi desenvolvido maioritariamente pelo elemento André Cruz (80%). Peso: 15%.

### 3.12 mouse

Contém o conjunto de funções essenciais para o funcionamento do rato. Código desenvolvido aquando do laboratório 4.

### 3.13 planetary

Módulo de mais alto-nível do projeto, contém toda a lógica de jogo e, para isso, usa a maior parte dos restantes módulos.

Criaram-se as *structs* `Game_t` e `Menu_t` para manter o estado do jogo e da interação com o menu, respetivamente. Estas *structs* foram implementadas com *singleton-like behaviour* em mente, sendo acedidas apenas através das funções `game_instance` e `menu_instance`. Criou-se também a *enum* `game_state_t` que representa o estado atual da aplicação, tendo um valor para cada diferente estado do projeto, para a implementação de uma máquina de estados no `timer_handler`.

Este módulo foi desenvolvido maioritariamente pelo elemento André Cruz (70%). Peso: 20%.

### 3.14 rtc\_asm

Contém algumas funções essenciais para a operação do *RTC* escritas em *Assembly*, que são depois evocadas no módulo *RTC*. Nomeadamente incorpora funções para a conversão de números em *BCD* (*binary coded decimal*) para binário, e para a leitura de registo do periférico em causa.

Este módulo foi desenvolvido unicamente pelo elemento André Cruz. Peso: 5%.

### 3.15 RTC

Contém as funções que permitem a operação e configuração do *RTC*, e faz a ligação entre o correspondente módulo de assembly.

Este módulo foi desenvolvido unicamente pelo elemento Edgar Carneiro. Peso: 10%.

### [3.16 Serial](#)

Contém as funções necessárias à operação, configuração e tratamento de interrupções da porta série, para que seja possível a receção e envio de dados através deste periférico.

Este módulo foi desenvolvido unicamente pelo elemento Edgar Carneiro. Peso: 12%.

### [3.17 timer](#)

Contém o conjunto de funções essenciais para o uso e configuração do *timer*.

Código desenvolvido nas aulas práticas do laboratório 2.

### [3.18 VBE](#)

Contém as funções que permitem a obtenção de informações sobre o modo de vídeo utilizado, bem como sobre a placa gráfica.

Código desenvolvido aquando do laboratório 5.

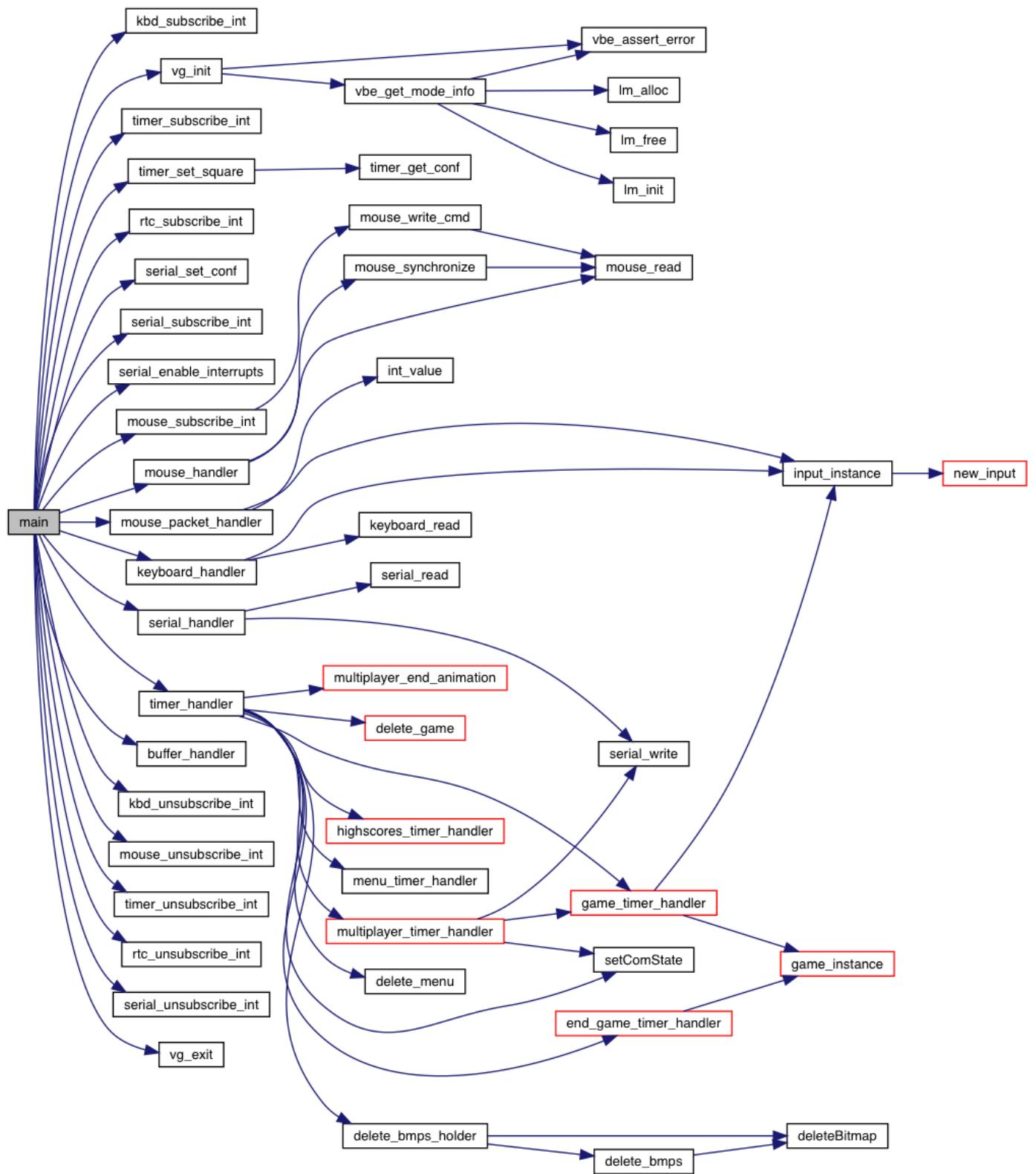
### [3.19 video\\_gr](#)

Contém o conjunto de funções que permitem a operação e configuração da placa gráfica, bem como o desenho de formas geométricas básicas no ecrã. Incorpora também funções para desenhar números no ecrã, a partir de um array de *bmps* fornecido, permitindo o uso de diferentes estilos e tamanhos de algarismos.

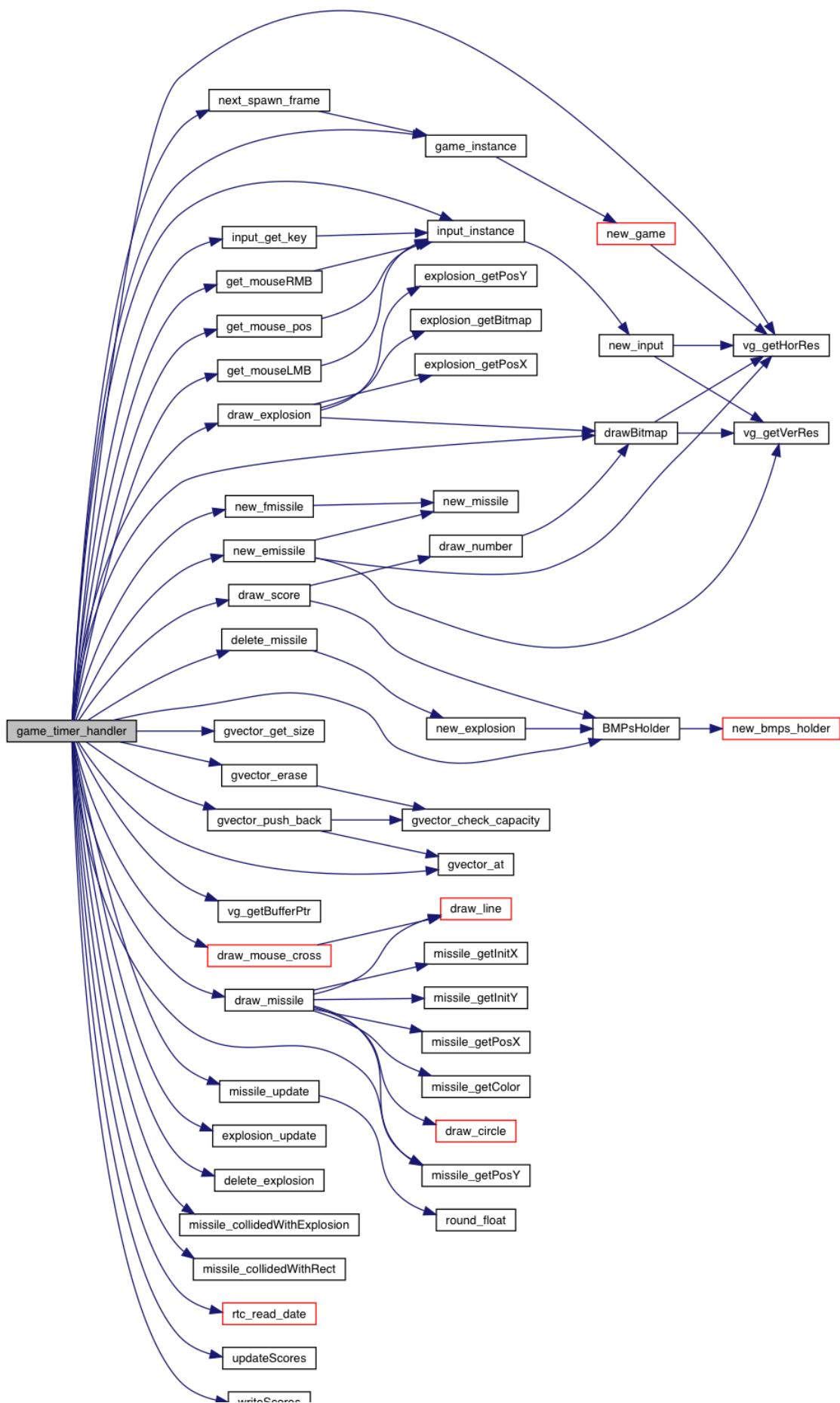
Este módulo foi desenvolvido igualmente pelos dois elementos do grupo, André Cruz e Edgar Carneiro (50%/50%). Peso: 4%.

### 3.20 Function Call Graph

Segue-se o *function call graph* para a função *main*.



Achamos útil também incluir o *function call graph* para a função *game\_timer\_handler*, que é responsável pela sessão de jogo.



## 4 Detalhes de Implementação

O desenvolvimento da aplicação teve sempre em mente uma estruturação *event driven*, conseguida pelo uso de uma máquina de estados no *timer\_handler*, bem como o tratamento de interrupções em si. A máquina de estados implementada tem como objetivo facilitar a programação e compreensão das mudanças entre os diferentes estados do projeto, e a sua implementação é auxiliada pela *enum game\_state\_t*. Em adição, todo o funcionamento do programa se baseia no tratamento de interrupções (eventos gerados pelos periféricos) no ciclo *driver\_receive* único, delegando adequadamente as tarefas a realizar como resposta a determinada interrupção. Todas estas características, para além de tornarem o código mais legível, contribuem para a uma fácil escalabilidade das funcionalidades implementadas.

Mantivemos também uma consistente modularidade ao longo do desenvolvimento, com o objetivo de ter uma base *low level* estável (comunicação com os periféricos), que nos permite uma maior abstração na implementação do jogo em si. Para este efeito procuramos também seguir uma orientação a objetos onde possível, apesar das limitações da linguagem neste campo (inexistência de classes propriamente ditas). Com isto em mente as classes *Missile*, *Explosion* e *GVector* (usadas extensamente no projeto) seguem escrupulosamente as indicações fornecidas nas aulas, nomeadamente, para impedir que fossem manuseadas indevidamente, encontram-se definidas no ficheiro “.c”, estando declaradas no ficheiro “.h” como tipos incompletos. Assim sendo, estas *structs* são apenas acessíveis pelos seus métodos. Devo também realçar que o *GVector* permite, como o nome indica, o armazenamento de instâncias de qualquer *struct*, sendo apenas necessário fornecer o tamanho em bytes de cada objeto.

Em relação ao desenvolvimento do excerto de *assembly*, sentimos alguma dificuldade na obtenção de informação sobre pormenores das instruções na sintaxe *AT&T*, nomeadamente na interação entre código de diferentes linguagens (sendo a informação relativa à sintaxe *Intel* muito mais abundante).

## 5 Conclusões

### 5.1 Avaliação da Unidade Curricular

Em retrospectiva, e tendo sido este o nosso primeiro contacto com uma UC laboratorial, podemos afirmar que a carga horária exigida ultrapassa largamente qualquer outra a que tenhamos assistido. No entanto, a exigência porque é pautada torna-a desafiante mas recompensadora.

Ainda assim, sugerimos o relaxamento das datas de entrega dos laboratórios para a quarta, quinta, sexta e sábado seguinte à aula laboratorial, respetivamente para as turmas de segunda, terça, quarta e quinta, ficando as turmas de sexta também sujeitas à entrega no sábado seguinte. Achamos esta solução justa, pois as turmas que devem entregar o código durante a semana têm dois dias para o fazer, e as turmas de sexta têm parte do fim de semana. Consideramos esta alteração benéfica, pois o *stress* resultante de a nossa avaliação se cingir aos resultados conseguidos até ao final da aula é um obstáculo à aprendizagem nessa mesma aula.

Sugerimos também que as aulas teóricas visem alguns exemplos práticos de código, de modo a facilitar a passagem entre a teoria e o que teremos eventualmente que desenvolver.

## 5.2 Apêndice

Para facilitar a instalação dos recursos necessários ao funcionamento do programa, implementamos alguns *shell scripts* simples. Assim, as instruções de instalação são:

- Entrar no diretório do projeto *proj*;
- Correr, como *superuser*, o script *install.sh*;
- Correr, como *superuser*, o script *compile.sh*;
- Correr, como *superuser*, o script *run.sh*.