

**Ball vs. Ball!**

MIEIC – LCOM

Desenvolvido por:

João Fidalgo

Pedro Costa

## Índice

[**Introdução** 2](#_Toc502597470)

[**Instruções de Utilização** 3](#_Toc502597471)

[ Menu Principal 3](#_Toc502597472)

[ Estado de Jogo 4](#_Toc502597473)

[ HighScores 6](#_Toc502597474)

[**Estado do Projeto** 7](#_Toc502597475)

[Timer: 7](#_Toc502597476)

[Rato: 7](#_Toc502597477)

[Teclado: 8](#_Toc502597478)

[Placa Gráfica: 8](#_Toc502597479)

[RTC: 8](#_Toc502597480)

[**Estrutura do Código** 9](#_Toc502597481)

[1. Ball 9](#_Toc502597482)

[2. Bitmap 9](#_Toc502597483)

[3. Highscore 9](#_Toc502597484)

[4. i8042 9](#_Toc502597485)

[5. i8254 10](#_Toc502597486)

[6. Field 10](#_Toc502597487)

[7. Keyboard 10](#_Toc502597488)

[8. Lmlib 10](#_Toc502597489)

[9. Mouse 10](#_Toc502597490)

[10. Player 11](#_Toc502597491)

[11. Proj 11](#_Toc502597492)

[12. Rectangle 11](#_Toc502597493)

[13. RTC 11](#_Toc502597494)

[14. Time 11](#_Toc502597495)

[15. Timer 12](#_Toc502597496)

[16. VBE 12](#_Toc502597497)

[17. Vector 12](#_Toc502597498)

[18. Video\_gr 12](#_Toc502597499)

[19. Driver\_Receive 12](#_Toc502597500)

[**Diagrama de chamada de Funções** 13](#_Toc502597501)

[**Detalhes da implementação** 16](#_Toc502597502)

[**Conclusões** 18](#_Toc502597503)

[**Apêndice** 19](#_Toc502597504)

# **Introdução**

Para este projeto, decidimos desenvolver a nossa versão do jogo “*[HaxBall](http://www.haxball.com/)*”.

De uma forma geral, trata-se de um jogo de futebol, entre dois jogadores. Os objetivos do jogo são de marcar golo na baliza no adversário e de defender a própria. O jogo irá acabar quando um dos jogadores marcar 3 golos ou quando o tempo total acabar. O campo é delimitado e nenhum dos jogadores ou bola pode sair do campo. Os jogadores podem interagir com a bola de duas formas, podem “empurrá-la” ou então rematar.

# **Instruções de Utilização**

## Menu Principal

No início do programa, é apresentado ao utilizador o Menu Principal.



Figura 1- Menu Principal

A partir deste menu o utilizador pode aceder a qualquer uma das funcionalidades do jogo. Para fazer tal pode fazer uso do rato e do teclado. Quando o rato se encontra por cima de qualquer uma das opções (“Start”, “Exit” e “HighScores”) aparece um retângulo a delimitar a opção para a destacar. Nas opções de entrada num novo estado, esta cor será azul, nas de saída, vermelha. Para selecionar qualquer uma destas, basta clicar no rato. Em relação ao teclado, o utilizador pode carregar na tecla ESC a qualquer momento o que o fará voltar ao menu anterior (no caso de se encontrar no Menu Principal, causará o término do programa).

## Estado de Jogo

Ao selecionar a opção de “Start”, o estado de Jogo irá começar. A seguinte imagem será apresentada:



Figura 2 - Campo de Jogo

No canto superior direito pode-se ver o contador de tempo, na parte central a pontuação atual do jogo. Cada jogador tem associado a si um conjunto específico de teclas que permitem o movimento e ações. Para o jogador vermelho, lado esquerdo do campo, as teclas de movimento são: W (cima), S (baixo), A (esquerda), D(direita). A tecla de ação, que permite rematar é o Espaço. No caso do jogador castanho, lado direito do campo, as teclas de movimento são as setas do teclado (direção e sentido implícito) e a tecla de ação é o Enter Numérico do Numpad.

Como foi dito anteriormente, na Introdução, o jogo irá acabar quando algum dos jogadores marcar 3 golos ou o tempo acabar.

A cada golo marcado, a posição dos jogadores e bola é automaticamente renovada para as posições iniciais e o jogo é pausado durante alguns breves momentos, sendo mostrada a seguinte imagem:



Figura 3 - Golo Marcado

Quando o jogo acaba, é apresentada a informação sobre qual dos jogares ganhou e o jogo volta automaticamente para o menu inicial. Também está presente a opção de acabar o jogo antes de ser cumprida qualquer uma das condições anteriores, ao carregar no ESC (neste caso, nenhum dos jogadores é considerado vencedor e não é apresentada a imagem de final de jogo). O cursor do rato é ignorado durante este estado.



Figura 4 - Final de Jogo

## HighScores

Neste menu, o utilizador pode ver as 3 melhores pontuações obtidas em todas as partidas, também como a hora e data dessas mesmas. Por melhor pontuação, entenda-se o menor tempo que qualquer um dos jogadores demorou a marcar os 3 golos. Note-se que os jogos que foram interrompidos pela saída abrupta (tecla ESC) não são considerados para as melhores pontuações.

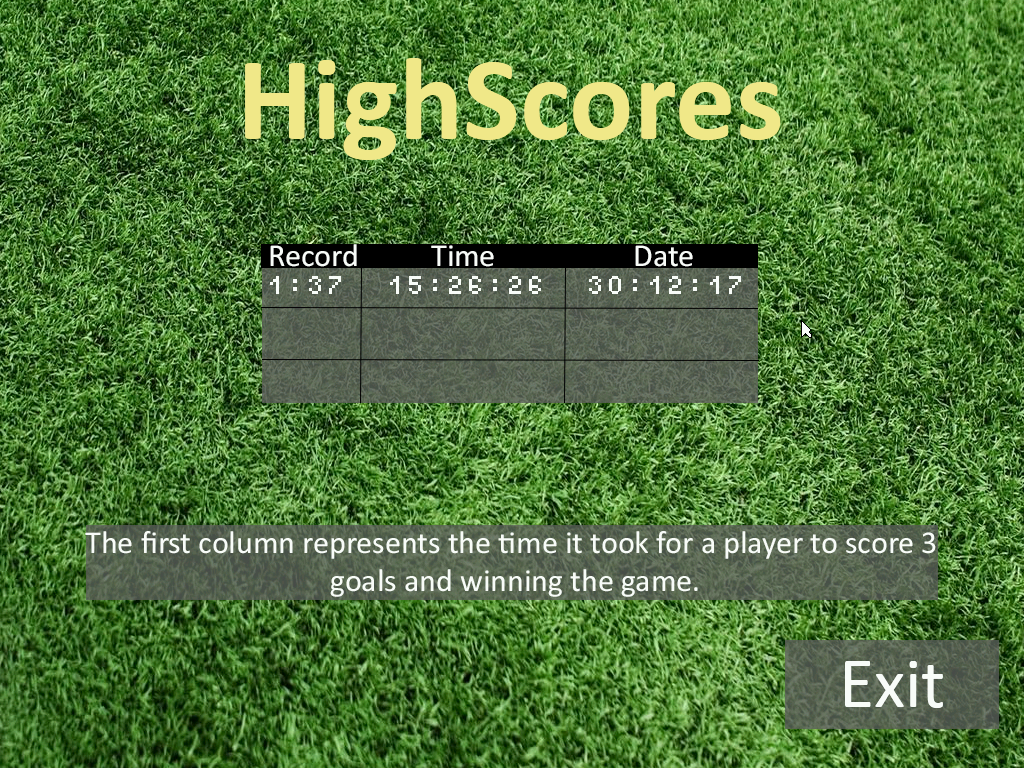


Figura 5 - Tabela de HighScores

# **Estado do Projeto**

Infelizmente, a implementação da porta de série não foi concluída. O elemento encarregue de a desenvolver foi o Pedro Costa, no entanto começou a originar vários problemas e o tempo restante para a entrega final do projeto não era muito, tomando em consideração o tempo já gasto até ao momento com a porta de série.

Tabela 1 – Dispositivos I/O

|  |  |  |
| --- | --- | --- |
| Dispositivo | Utilização | Interrupções |
| Timer | Controlo da frame rate e do contador de tempo de jogo | SIM |
| Rato | Navegação nos menus | SIM |
| Teclado | Controlo dos jogadores e navegação nos menus | SIM |
| Placa Gráfica | Apresentação dos menus e do campo de jogo | NÃO |
| RTC | Informação relativa à hora e data para os highscores | NÃO |

## Timer:

O Timer é usado para controlar a frame rate do jogo através das interrupções. Está implementado no ficheiro timer.c, usado no módulo principal do jogo, proj.c. Também é usado para contabilizar o tempo de jogo, decrementando, a cada segundo um contador. Para o controlo foram criadas duas funções, game\_tick() e seconds\_tick().

## Rato:

Quer o movimento do rato quer os seus botões são usados no programa. Serve para navegar nos menus, através do pressionar do botão esquerdo. A sua implementação está no ficheiro mouse.c e é usado no ficheiro proj.c (altera o estado do programa).

## Teclado:

É usado quer durante o jogo em si quer na navegação dos menus. Permite o movimento e controlo dos jogadores (cada um com o seu conjunto de teclas específicas) e, na navegação dos menus, para voltar à opção anterior, através da tecla ESC. Está implementado no ficheiro keyboard.c e é usado no ficheiro proj.c e player.c (nas funções de atualização do estado dos jogadores).

## Placa Gráfica:

O modo gráfico usado é o 117h, com a resolução 1024x768 e 64000 cores. É usada durante todo o “run-time” do programa. Está implementada no ficheiro vídeo\_gr.c. É usada a técnica de Double Buffering. São usadas 3 funções VBE, a função para mudar de modo (0x02), a função para obter informação sobre o modo (0x01) e a função para obter informações sobre as capacidades gráficas (0x00). Dadas as formas geométricas dos nossos objetos (esferas) o algoritmo de deteção de colisão desenvolvido depende apenas da posição dos jogadores/bola e do seu tamanha. Este dispositivo é usado praticamente em todo o projeto (mais visível no proj.c/bitmap.c/player.c/ball.c).

## RTC:

É usado para ler a data e o tempo depois do final do jogo, em casa de haver um novo recorde, não sendo então necessárias interrupções. Está implementado no ficheiro rtc.c e é usado pelo ficheiro highscore.c

# **Estrutura do Código**

## Ball

Módulo responsável pela criação, controlo, desenho e eliminação da bola no campo durante o jogo. Atualiza a posição e velocidade da bola a cada frame, delimitando cada um destes atributos aos máximos e mínimos permitidos. Também é feito neste módulo a verificação de golo. Dado que apenas existirá uma bola durante todo o programa, esta encontra-se encapsulada dentro do seu próprio ficheiro. Foi desenvolvido pelo elemento Pedro Costa. Representa 8% do projeto.

## Bitmap

Módulo responsável por ler, desenhar e apagar bitmaps. Estes são usados para todas as imagens de fundo, para o rato e números (contador de tempo/pontuação). O código foi desenvolvido por Henrique Ferrolho e pode ser consultado em: <http://difusal.blogspot.pt/2014/09/minixtutorial-8-loading-bmp-images.html> . Foram realizadas algumas alterações, removendo algumas das funcionalidades do código que não achámos úteis para o nosso projeto (por exemplo, a opção de alinhar os bitmaps). Foi desenvolvida, com base no código existente, uma função para desenhar bitmaps com transparência (drawBitmapShape()), que ignora a cor verde pura rgb(0,255,0). As alterações efetuadas ao código disponibilizado foram desenvolvidas pelo elemento Pedro Costa. Representa 5% do projeto

## Highscore

Módulo que verifica, guarda, desenha e atualiza a informação relativa aos highscores. Faz uso do RTC para associar a cada highscore a data e o dia. Lê e guarda a informação num ficheiro. Cada highscore é representado por uma estrutura, “Top”, e no final de cada jogo, com a função testNewHighscore(), os highscores são atualizados. Este módulo foi desenvolvido pelo elemento João Fidalgo. Representa 5% do projeto.

## i8042

Ficheiro que contém macros e funções úteis relativas à informação proveniente do KBC. Usadas para tornar o código legível. É de notar que este ficheiro já tinha sido criado durante as aulas laboratoriais. Representa 1% do projeto.

## i8254

Seguindo o esquema dos últimos dois ficheiros apresentados, contém macros e funções úteis relativas ao uso e manipulação do Timer, tendo também sido desenvolvido durante as aulas laboratoriais. Representa 1% do projeto.

## Field

Ficheiro que contém apenas macros úteis para o controlo dos objetos em campo, como, por exemplo, os limites do campo. Foi desenvolvido pelo elemento Pedro Costa. Representa 1% do projeto.

## Keyboard

Módulo responsável por lidar com o input do teclado. Módulo essencial no projeto, dado que todo o controlo do jogador por parte dos utilizadores é feito através do teclado. Este lê os scancodes gerados pelo KBC e atualiza uma estrutura criada para representar uma tecla. De acordo com esta informação, os jogadores serão então atualizados. Este módulo foi desenvolvido em conjunto pelos dois elementos, sendo bastante código das aulas laboratoriais reutilizado. Representa 11% do projeto.

## Lmlib

Módulo fornecido pelo professor, não tendo sido efetuada nenhuma alteração por nós.

## Mouse

Módulo responsável por lidar com o input do rato. Trata de ler os bytes gerados pelo KBC e criar os packets pertencentes ao rato. Atualiza a estrutura usada para representar o rato com a informação contida no rato e também o desenha no ecrã, através do módulo do Bitmap. O rato é essencial para a navegação nos menus. Foi implementado pelo elemento Pedro Costa. Representa 10% do projeto.

## Player

Módulo responsável pela criação, manipulação, desenho e eliminação dos jogadores em campo. Neste módulo foram usadas duas estruturas. Uma representa os jogadores em si (“Player\_t”) a outra representa o movimento gerado pelo utilizado referente a cada um do jogador (“MOVEMENT\_EVENT”). A atualização do jogador é feita de acordo com esta última. Também é tratado neste ficheiro a colisão entre jogadores (dealWithColision() e fixPlayerOverlaping()). Este módulo foi desenvolvido pelo elemento Pedro Costa. Representa 13% do projeto.

## Proj

Módulo principal no programa. Inicializa todos os recursos usados pelo jogo. Subscreve e trata das interrupções geradas pelos dispositivos, inicializando também o modo gráfico. Contém o ciclo principal do jogo onde relaciona todos os módulos presentes. Contém em si também a implementação da máquina de estados. Este módulo foi desenvolvido pelo elemento Pedro Costa. Representa 15% do projeto.

## Rectangle

Módulo usado para a criação, manipulação, desenho e eliminação de retângulos nos menus. Serve de abstração para criar o código mais legível e para ajudar na navegação dos menus. Este módulo foi desenvolvido pelo elemento Pedro Costa. Representa 3% do projeto.

## RTC

Módulo usado para aceder à informação do RTC. Usado no final de cada jogo para obter a hora e data na qual o highscore foi obtido. Este módulo foi desenvolvido pelo elemento João Fidalgo. Representa 3% do projeto.

## Time

Módulo usado para o controlo do tempo de jogo. Sempre que um novo jogo começa, inicializa e desenha um contador que representa o tempo restante do jogo. Relaciona-se bastante com o módulo do Timer. Este módulo foi desenvolvido pelo elemento João Fidalgo. Representa 4% do projeto.

## Timer

Módulo usado para a manipulação do timer. Serve para controlar as frames do projeto e para controlar o contador de tempo descrito anteriormente. Foi implementado pelo elemento João Fidalgo, recorrendo a código previamente desenvolvido nas aulas laboratoriais. Representa 5% do projeto.

## VBE

Módulo importado na íntegra do código das aulas Laboratoriais. Testa se o modo escolhido no projeto é suportado pela placa gráfica do computador. Representa 2% do projeto.

## Vector

Módulo criado com o objetivo de criar uma certa camada/abstração para resolução da colisão dos jogadores/bola, tornando a “física” do jogo mais fácil. Módulo desenvolvido pelo elemento Pedro Costa. Representa 3% do projeto.

## Video\_gr

Módulo responsável por inicializar o programa no modo gráfico pretendido, os Buffers usados para o desenho e outras informações úteis para o desenho dos objetos no ecrã. Grande parte do código foi importado das aulas laboratoriais e o restante desenvolvido em conjunto. Representa 10% do projeto.

## Driver\_Receive

Módulo onde a função driver\_receive() se encontra documentada, para gerar informação completa com a ferramenta Doxygen.

# **Diagrama de chamada de Funções**

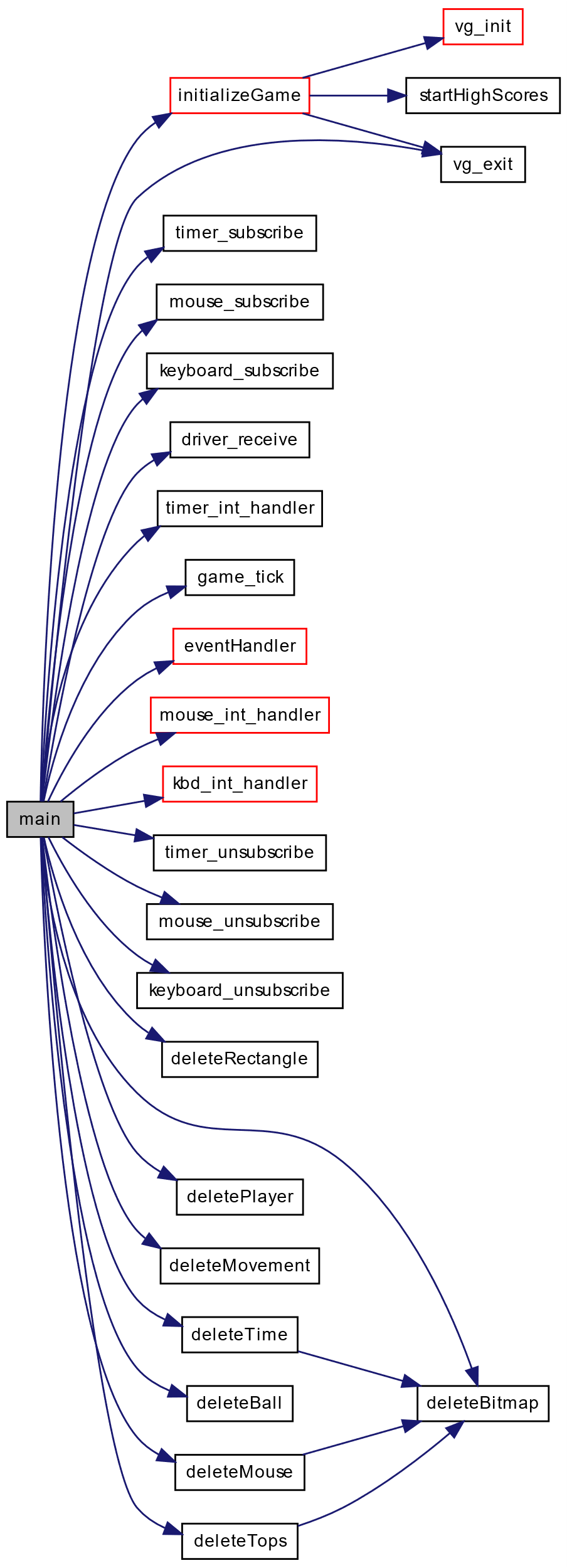
 Dado que o nosso gráfico principal era demasiado grande e pouco legível, dado o número de funções presentes, decidimos dividi-lo em 3 parte principais.

Figura 6 - Ciclo Principal do Jogo

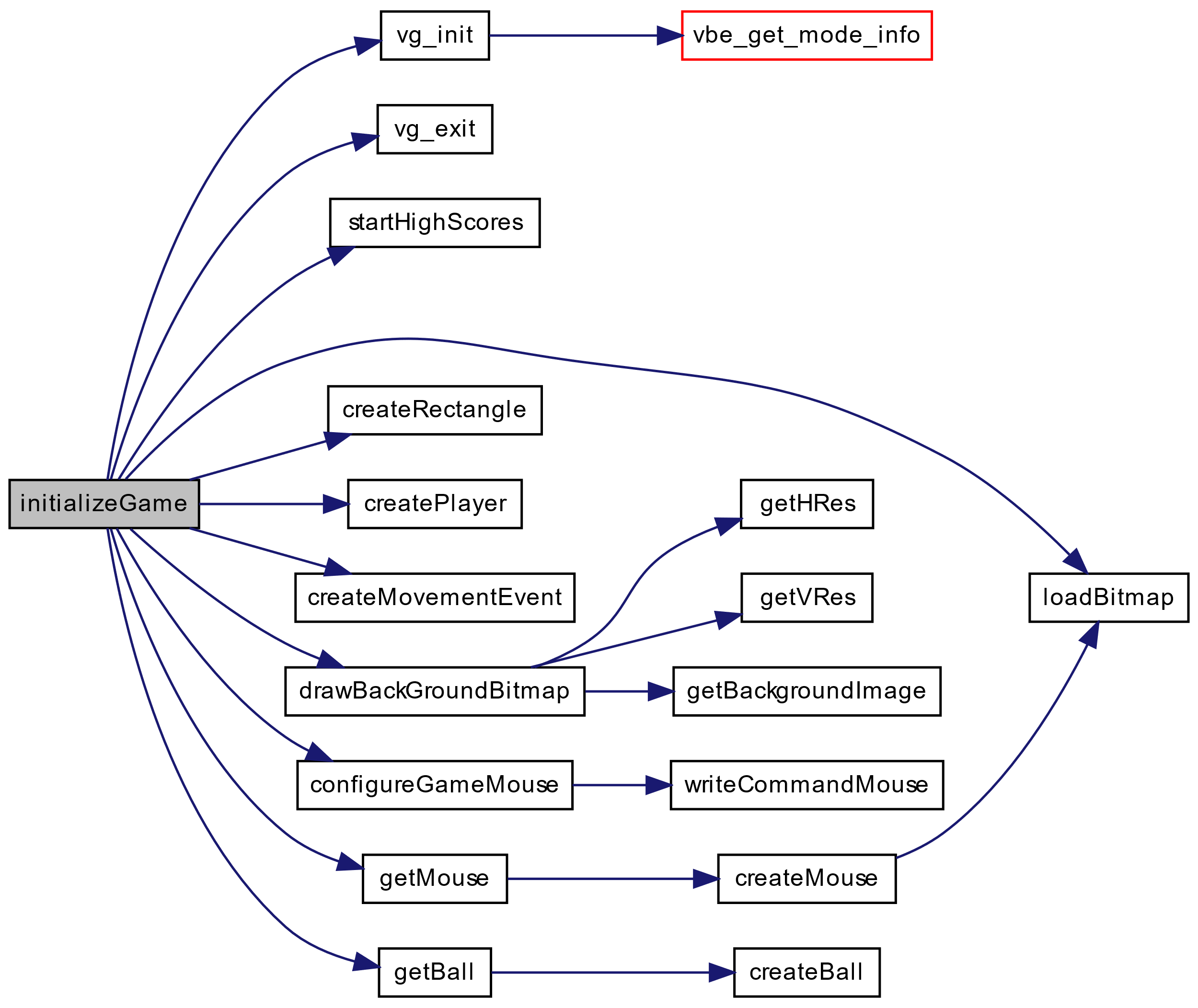


Figura 7 - Função de Inicialização de Recursos

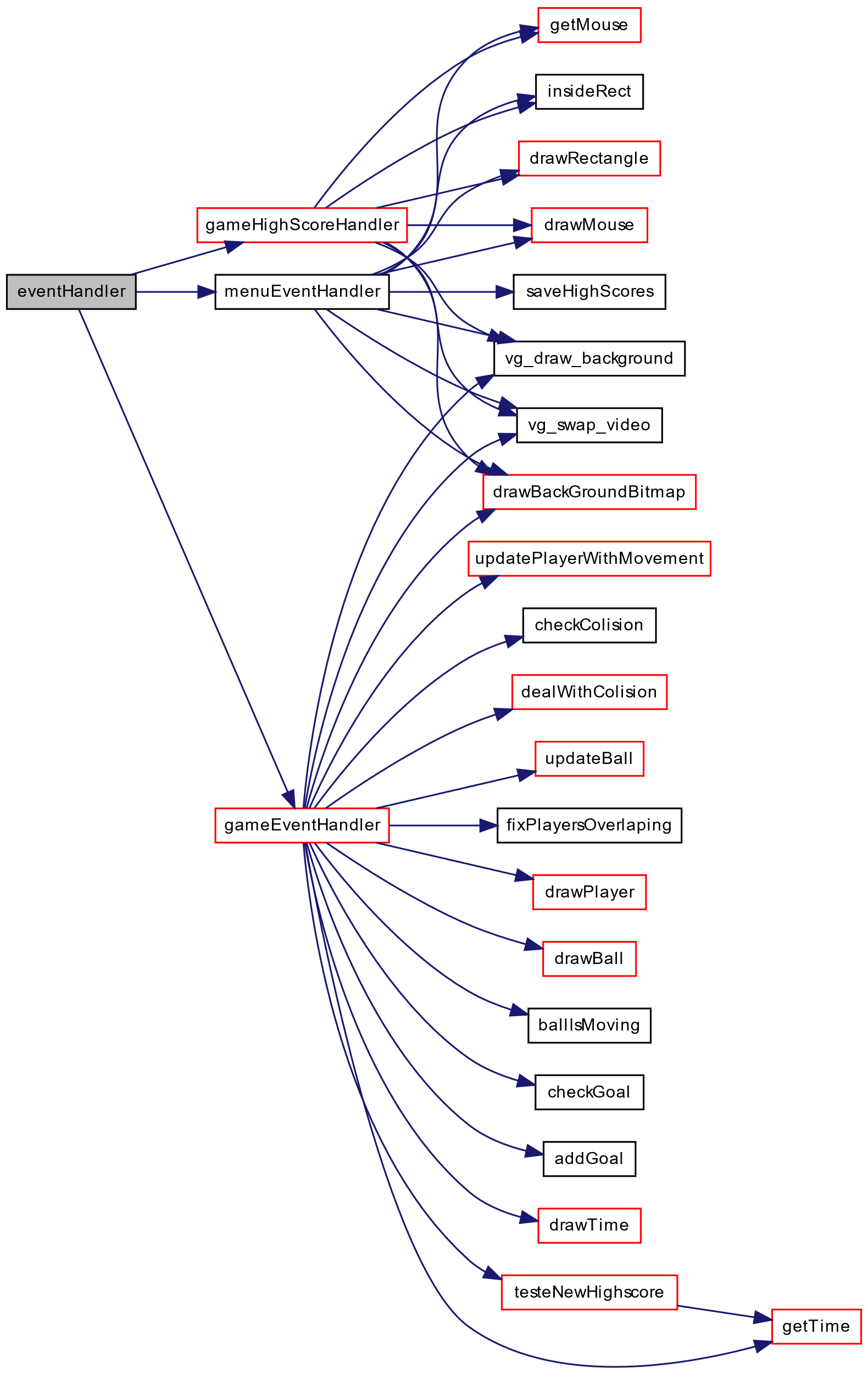


Figura 8 - Event Handler Geral

# **Detalhes da implementação**

Em todos os módulos relativos aos dispositivos IO foi reaproveitado código dos labs anteriores, tornando a criação de uma boa base uma tarefa mais fácil, permitindo construir o nosso projeto em cima de tal. A maioria das técnicas usadas no projeto foram discutidas nas aulas teóricas o que facilitou o desenvolvimento.

A física/mecânica do nosso jogo revelou-se ser um dos problemas centrais. Dada as simples formas geométricas dos nossos objetos, todos círculos, pensámos de início que a resolução das colisões seria algo relativamente fácil. Tal não se verificou. A mera deteção da colisão foi algo trivial (é apenas necessário tomar em consideração as posições dos objetos e o seu raio), no entanto o que fazer depois revelou-se mais complicado. Por vezes, a velocidade obtida após a colisão não era suficiente e os objetos ficavam sobrepostos, outra vezes o movimento não parecia natural. Para além das colisões entres os objetos, também tivemos de tomar em consideração que estas podiam acontecer ao mesmo tempo que um deles colidia com a periferia do campo. No entanto, a nosso ver, todo isto foi resolvido, apresentando um movimento suave e esperado de todos os objetos.

Embora a linguagem C não seja a mais apropriada para o assunto, usou-se bastante uma abordagem orientada a objetos, como se pode ver pela criação de estruturas relativas a todos os aspetos importante do projeto (jogadores, bola, contador, etc…). Isto permitiu-nos a criação de camadas para a atualização de todos estes, tornando o módulo principal, proj.c, muito mais legível.

No nosso módulo principal, o nosso código é maioritariamente “event driven”. Após a receção e tratamento de dados por qualquer um dos dispositivos, é chamado um Handler Geral com o tipo de evento gerado (entenda-se por isto o teclado, rato ou timer). Nesse Handler Geral, dependendo do estado atual do jogo (guardado numa variável estática existente apenas neste), são chamados handlers mais específicos que, dependendo do evento gerado, chamam as funções respetivas e atualizam o estado do jogo. Isto permitiu-nos facilmente ignorar alguns dispositivos em certos estados do jogo (por exemplo, o rato no estado de jogo). A fácil estrutura e transparência do Handler torna o processo de adicionar um outro estado uma tarefa simples.

Como foi dito na secção da estrutura do código, para a atualização e desenho dos jogadores foram usadas duas estruturas. Dado que os dois utilizadores podem premir em várias teclas várias vezes no tempo entre duas frames consecutivas decidimos guardar esta informação sobre quais as teclas premidas numa estrutura. Assim, quando uma nova frame surgir, o jogador só será atualizado aí com base nessa mesma estrutura. Torna-se então impossível ocorrerem saltos súbitos no ecrã e o movimento tornou-se bastante suave e responsivo.

Foi adicionado a todos objetos uma certa “força de atrito” que torna toda a “flow” do jogo muito mais natural. No entanto, na implementação do atrito, surgiu-nos um problema cujo motivo ainda nos é desconhecido. A força de atrito trata-se de uma certa constante entre 0 e 1 que é multiplicada à velocidade e aceleração dos jogadores, diminuindo-as. No entanto, quando a velocidade era negativa, o atrito, por alguma razão, era muito menos eficaz, por vezes só parando os jogadores quando estes colidissem com os limites do campo; quando a velocidade era positiva, funcionava como pretendido. Para “resolver” este problema, em vez de ser aplicado o atrito à velocidade em si, foi aplicado ao absoluto da velocidade e depois convertido para positivo ou negativo, dependendo dos casos.

Em relação ao desenho das imagens principais de fundo, foi usado o código desenvolvido pelo Henrique Ferrolho. Para evitar chamar a função desse mesmo código de desenhar tais imagens a cada frame, que tem de percorrer a imagem do Bitmap “ao contrário” e fazer várias verificações, decidimos usar uma variável temporária que guarda, em cada um dos estados do jogo essa imagem. Assim é só necessário chamar uma vez a função que desenha a imagem, no início de cada alteração de estado; ao ser chamada, em vez de desenhar diretamente para o buffer, desenha para uma variável (mais se assemelha a um load, mas essa terminologia já era usada para o load literal de uma imagem bitmap raw). Para depois desenhar no buffer, será apenas necessário efetuar um memcopy, função que já se encontra otimizada. Isto pode ser verificado nas funções drawBackgroundBitmap(), ficheiro bitmap.c, e vg\_draw\_background(), ficheiro vídeo\_gr.c.

# **Conclusões**

A nosso ver, esta foi uma das cadeiras mais interessantes até ao momento no curso. É da maior importância para o curso, permitindo aos alunos criar algo interessante e do qual gostam através de conhecimentos adquiridos durante o primeiro ano do curso. Alertou-nos para a importância da documentação do código e do trabalho de pesquisa, mais evidente na realização do lab 5. Aumentou em bastante o nosso conhecimento em programação em C, em implementação de máquinas de estado, aplicação de OOP em C, fornecendo-nos ferramentas para criar algo do qual realmente gostamos.

No entanto, é também uma cadeira muito desafiante desde a primeira aula laboratorial. Sentimos que o grande trabalho colocado e requerido todas as semanas para as aulas laboratoriais não é de todo refletido pelos créditos que a cadeira vale. Para realizar corretamente e perceber bem os Labs é necessário muito trabalho prévio.

Na nossa opinião, deveriam existir aulas suplementares sobre a linguagem C. Até ao momento, todo o que aprendemos foi C++ e assembly, no entanto todo o projeto é realizado em C. Certos aspetos como a manipulação de ficheiros, de strings (bastante importante para o desenvolvimento da serial port) e a ausência da tão usada STL, tornou o projeto ainda mais difícil, requerendo ainda mais pesquisa para além da sobrecarga presente pelos labs.

# **Apêndice**

No nosso Projeto são usados bastantes ficheiros (imagens e texto) que não se encontram no mesmo diretório do código. Infelizmente, não conseguimos desenvolver nenhuma função 100% funcional para obter o caminho absoluto desses mesmos. Assim, para tornar o processo de mudar o caminho menos penoso, criámos duas MACROS no ficheiro field.h, IMAGES\_PATH e FILE\_PATH, linhas 18 e 19 respetivamente, que ditam o caminho a ser usado em todos os restantes ficheiros. Por predefinição, estão definidas como “/home/lcom/proj/…”. Se o checkout for efetuado para outra localização diferente, terá de se atualizar o caminho antes da execução do código.

O ficheiro presente na pasta conf/ também tem de ser copiado para /etc/system.conf.d/. Estando dentro do diretório do projeto, proj/, isto pode ser realizado com o comando “cp conf/ballvsball /etc/system.conf.d”. Depois de compilado, para correr o programa, “service run `pwd`/ballvsball”.

No computador onde o Doxygen foi usado para gerar a documentação, não foi possível usar o formato .png ou .jpeg para as imagens, obtendo uma mensagem de erro. Decidimos, então, selecionar o formato .svg. As variáveis Depth e Max Nodes do ficheiro doxygen também foram alteradas, portanto toda a informação criada pelo doxygen foi colocada no modulo “Documentos” do repositório.