FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Ano letivo 2016/2017

LCOM – Laboratório de Computadores

Trabalho realizado pelo grupo T6-G25:

Bernardo Manuel Costa Barbosa – up201503477@fe.up.pt

Duarte Nuno André Carvalho – up201503661@fe.up.pt

Índice

1 Breve descrição do Penaltix

O Penaltix é um jogo de grandes penalidades, baseado nas situações de desempate por penaltis da realidade, e permite o utilizador jogar contra o computador ou contra outro utilizador.

Só são necessários um rato e um teclado. Ambos servem para navegar pelo menu, no entanto, para defender usam-se as *ARROW_KEYS* (se o utilizador quiser defender para o canto inferior direito, apenas tem de clicar *DOWN_KEY* e clicar *RIGHT_KEY*), para rematar o utilizador só tem de clicar no RIGHT_BUTTON do rato e arrastar o rato na direção que quer (se o utilizador quiser rematar para o canto inferior direito, arrasta, mais ou menos a olho, para o canto inferior direito do trackpad ou do tapete do rato).

Para ganhar, o utilizador tem de marcar mais golos que o adversário nas primeiras cinco grandes penalidades. Em caso de empate, o primeiro interveniente a marcar penalti e a defender o seguinte ganha (morte súbita). Tudo isto ocorre tal e qual a realidade.

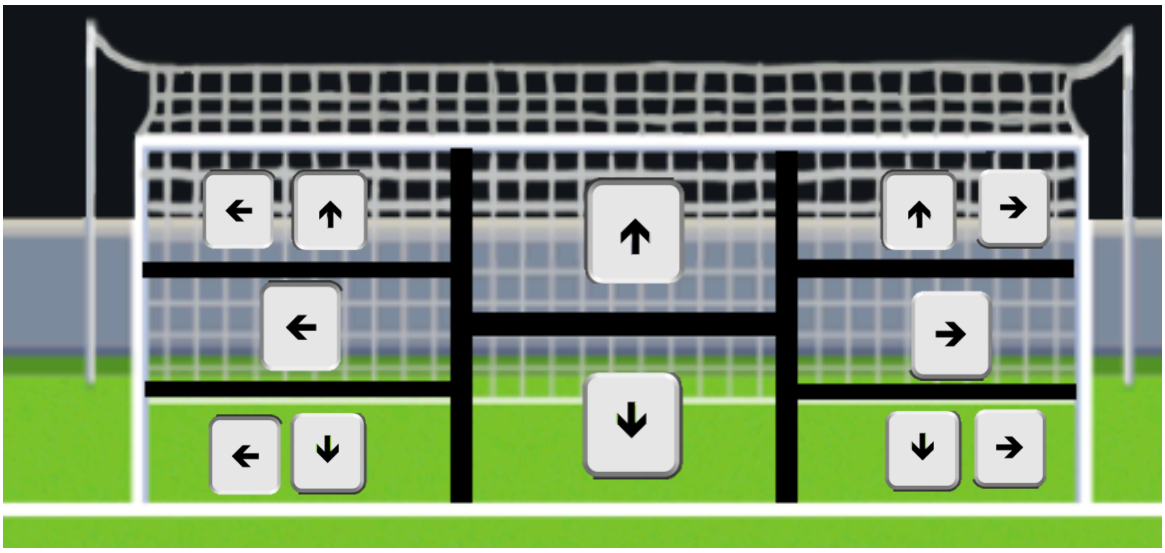


Figura 1 – save_area_about.bmp

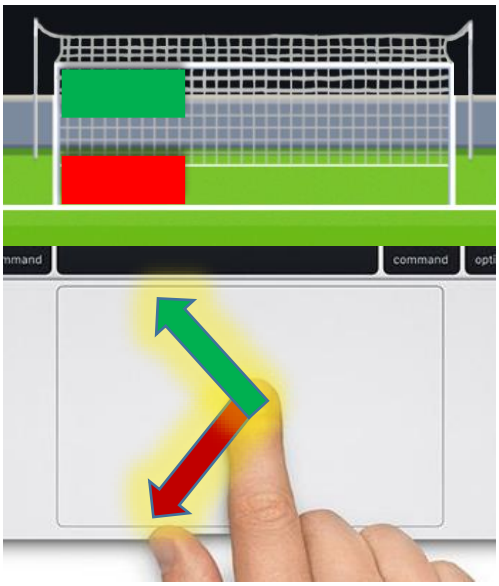


Figura 2 – shoot_about.bmp

2 Organização do Penaltix

2.1 Menu

Quando iniciamos o programa, é apresentado ao utilizador um dos seguintes menus (Figura 3 – Figura 4 – Figura 5).

Figura 3 - day_final.bmp

Figura 4 – sunset_final.bmp

Figura 5 - night_final.bmp

Estes, são apresentados conforme a hora do dia, ou seja, se quando o utilizador inicializa o jogo as horas se encontram entre as 8 da manhã e as 4 da tarde, será apresentado o menu com o céu azul claro (Figura 3), caso o utilizador inicialize o jogo entre as 5 da tarde e as 8 da noite, será apresentado o céu com o pôr-do-sol (Figura 4), nas restantes horas é apresentado o menu num cenário noturno (Figura 5). As horas são obtidas através do RTC.

Qualquer um destes menus iniciais apresentam 3 opções. Uma "*Play*", outra "*About*" e, finalmente, a "*Exit*". Destacamos que, com qualquer tipo de pré-seleção, a opção fica destacada, como podemos observar na Figura 6. Destacamos também que a navegação ou pré-seleção pode ser feita ou com o rato ou com a tecla TAB. No primeiro caso, o utilizador é redirecionado para um submenu, onde lhe será dada a oportunidade de escolher o modo de jogo que pretende. No segundo caso, são apresentados ao mesmo os créditos do jogo. No terceiro caso, o jogo fecha e volta ao ambiente do Minix.

Figura 6 - exit.bmp

2.2 About

Quando o utilizador seleciona a opção "*ABOUT*", é lhe apresentado um submenu (Figura 7) que contém quatro opções: "*SHOOT*", "*SAVE*", "*CREDITS*" e "*BACK*". Tal como no menu principal, o utilizador pode navegar pelas várias opções usando o rato ou a tecla *TAB*, no entanto, só precisa de passar com o rato por cima das opções para aparecerem as imagens correspondestes às várias opções, à exceção de "*BACK*" que é preciso clicar.

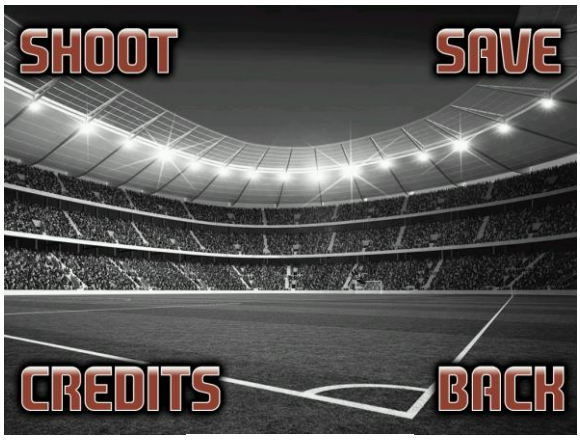


Figura 7 - sub_About

As primeiras duas opções são as instruções do jogo. Se o utilizador pré-selecionar a opção "*SHOOT*", é mostrado como rematar, como podemos verificar na Figura 8, se o utilizador pré-selecionar a opção "*SAVE*", o jogo mostra como defender, tal como vemos na Figura 9.

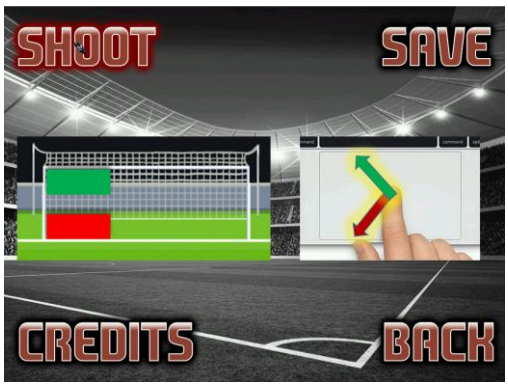


Figura 8 - Shoot Instructions

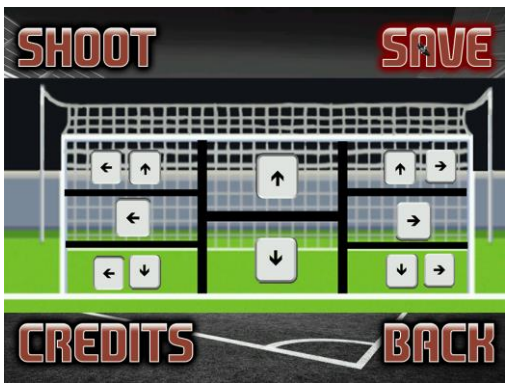


Figura 9 - Save Instructions

A opção "*CREDITS*" mostra os autores do *Penaltix*, como podemos observar na Figura 10. E a opção "*BACK*" volta ao Menu Inicial.

Figura 10 - CREDITS

2.3 PLAY

Quando, no menu inicial, escolhemos a opção "*PLAY*", é dado a escolher ao jogador se quer jogar contra o computador ou se quer jogar contra outra pessoa. Estas opções são dadas a escolher no menu que vemos na Figura 11. Realçamos que neste menu a navegação também pode ser feita com o rato ou com a tecla *TAB*. Tal como nos outros menus, se o jogador clicar em "*BACK*", voltará ao menu principal.

Figura 11 - Submenu de PLAY

2.3.1 Modo Multiplayer

No caso de o jogador selecionar a opção "*MULTI*", será apresentado uma imagem (Figura 12 e Figura 13), em ambas as máquinas, a dizer "*WAITING*" enquanto que o jogo tenta detetar a outra máquina, através da porta de série (UART).

Figura 12 - Máquina 1 à espera da 2

Figura 13 - Máquina 2 à espera da 1

Quando o segundo jogador é detetado, aparece no ecrã um aviso que indica que está tudo pronto e indica ao utilizador se é o “Player 1” ou se é o “Player 2” como também a sua cor. Tudo isto é apresentado de forma simples e intuitiva como podemos observar nas Figuras 14 e 15!

Figura 14 - Aviso para o Player 1

Figura 15 - Aviso para o Player 2

No caso de alguma das máquinas se disconectar, aparecerá um aviso ao jogador que ainda se encontra no jogo (Figura 16 e 17) a informá-lo que a conexão foi perdida, e depois voltará, automaticamente, para o menu inicial.

Figura 16 - Player 1 desconecta-se

Figura 17 - Player 2 desconecta-se

2.3.2 Single Vs Multi

Os modos "*Singleplayer*" e "*Multiplayer*" apresentam algumas diferenças que devem ser evidenciadas:

1. Começando logo pelo "*Scoreboard*", que, no primeiro modo, apresenta o nome do utilizador como "*YOU*" do lado esquerdo e a máquina como "*COM*" do lado direito, como se pode observar na Figura 18; no segundo modo, o "Scoreboard" apresenta a máquina "*host*" do lado esquerdo como "Player 1" e o "Player 2" do lado direito (Figura 19).

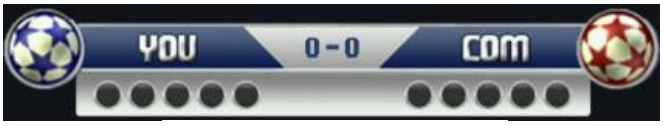


Figura 18 - Scoreboard Singleplayer

Figura 19 - Scoreboard Multiplayer

2. No que toca à maneira como o jogo decorre, pode afirmar-se que o jogo, no segundo modo de jogo, os jogadores têm mais controlo da velocidade do jogo, porque só ocorre ação quando um dos intervenientes remata, querendo isto dizer que os jogadores, no modo "Multiplayer", podem ter mais tempo para se preparar, enquanto que, no modo "Singleplayer", isso não é possível, porque passado um segundo de ser a vez do "COM" chutar, este o faz, quer o jogador esteja preparado para defender ou não.
3. Por último, quando o jogo acaba e é anunciado o vencedor da partida, são apresentadas mensagens diferentes. Apresentam-se a baixo os vários casos possíveis:
    a. No primeiro modo:

Figura 20 - COM ganha

Figura 21 - YOU ganha

    b. No segundo modo:

Figura 22 - Player 1 ganha

Figura 23 - Player 2 ganha

2.4 Estado de jogo

Na altura de bater grandes penalidades, a mecânica e os gráficos de jogo são exatamente iguais, exceto, claro, nas situações descritas em cima.

A máquina fica à espera do jogador que chuta. A partir do momento em que o jogador chuta, o guarda-redes já não consegue decidir o lado e fica no meio. Portanto, o jogador que está a controlar o guarda-redes tem de decidir primeiro que o jogador que chuta. No caso de se estar a jogar contra o computador, o jogador tem 5 segundos para decidir para que lado quer atirar o guarda-redes. No caso de estarem dois jogadores físicos a jogar, apela-se ao bom senso de ambos os jogadores.

2.4.1 Mecânica de jogo

2.4.1.1. Defender

Como é dito no primeiro capítulo do relatório, o jogador tem de usar as teclas para defender. Isto é feito através de constantes *interrupts* ao controlador do *Keyboard.* Depois de decidido uma zona da baliza, já não é possível alterá-la. É de denotar que, caso seja um utilizador a defender, é apresentado um retângulo amarelo para mostrar ao utilizador que área é que está a escolher, como se pode ver na Figura 24. Já no caso de ser o *COM* a defender, quando o utilizador está em Singleplayer, para simular o comportamento de um jogador real, é gerado, aleatoriamente, um número de 0 a 8 e esses números correspondem a zonas de baliza, tal como as *ARROW_KEYS* quando um jogador está a defender.

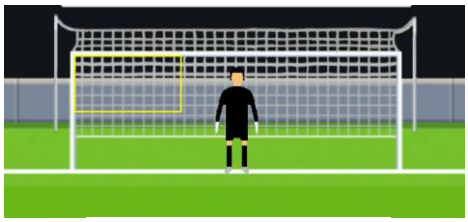


Figura 24 - Retângulo de apoio

2.4.1.2. Rematar

Para rematar, o utilizador usa o rato. A inclinação do remate (*slope*) é determinada através da posição inicial e final do rato. A posição inicial é determinada apenas no momento em que o utilizador pressiona o *Right Button* do rato. Optamos por este método para distinguir entre movimentos involuntários do jogador e o movimento do remate pretendido. Quanto à altura, se o movimento for feito para cima a bola vai para a metade superior da baliza, caso contrário, a bola vai para a metade inferior da baliza. A velocidade do movimento vai alterar o valor de deslocamento da bola em ambas dimensões (X e Y). Tudo isto é determinado e calculado através de constantes *interrupts* ao controlador do *Mouse*.

Para simular o remate do COM, é gerada, aleatoriamente, uma posição que para além de englobar a baliza, engloba também uma área à volta, podendo o COM, por isso, falhar, aumentando assim o realismo do jogo.

2.4.1.3 Deteção de golo

Neste caso optamos por detetar se a bola se sobrepõe a qualquer cor do guarda-redes, prevenindo casos em que a bola bate nas pernas do guarda-redes e seja considerado golo, simplificando, se a bola tocar no guarda-redes não é golo.

2.4.1.4 Resultado e Cálculo de Vitória

A cada remate, o *Scoreboard* é atualizado:

1. Atualiza-se o número de golos;
2. Preenchemos um dos círculos não preenchidos com cor verde, caso seja golo, vermelho em caso contrário;
3. Verificamos se algum dos utilizadores alcançou vitória;

Para a impressão e atualização de resultados, recorremos a um *number set* (Figura 25) em que a partir do valor de golos é imprimido uma certa área desse *number set*.

Figura 25 - number.bmp

Já o preenchimento dos círculos é feito através de posições já conhecidas (são sempre as mesmas). Se já não houver espaços restantes para preencher, o programa limpa todos espaços e volta a preenche-los de início. Isto só ocorre em caso de empate ao fim das primeiras dez grandes penalidades, o que faz com que o jogo entre no estado de morte súbita.

3 Periféricos

Apresentamos na Tabela 1 todos os periféricos utilizados e se usamos interrupções ou não.

Periféricos	*Interrupts*
Timer	SIM
Keyboard	SIM
Mouse	SIM
Placa Gráfica	NÃO
RTC	NÃO
Serial Port (UART)	SIM

Tabela 1 – Periféricos utilizados

3.1 Timer

O timer, para o nosso projeto, é dos periféricos mais importantes, pois é este que permite controlar o *frame rate* do jogo, a velocidade da bola, o *timing* das animações, todos os movimentos dos jogadores. Realçamos que apenas foi usado o Timer 0.

3.2 Teclado

O keyboard é utilizado como alternativa de navegação pelos vários menus do jogo e, também, para selecionar para que zona da baliza é que o jogador que está a defender pretende atirar o guarda-redes.

3.3 Rato

O rato tem como principal função o remate, mas também serve para navegar pelos menus, com um cursor, em que definimos uma posição inicial para o rato, no início do programa, e conforme os movimentos do rato, movemos o cursor.

3.4 Placa Gráfica

Usamos a placa gráfica para desenhar imagens (bitmaps), no modo 0x117 (1024x768), 16 bits por pixel) com 2^{16} cores.

Para o desenho do rato, do jogador e da bola usamos double buffering e para o guarda-redes é usado triple buffering.

Também usamos a placa gráfica para detetar colisões, ou seja, se alguma cor da bola se sobrepuser à do guarda-redes, existe colisão.

3.5 RTC

Usamos o RTC para saber as horas em que o utilizador abre o jogo e conseguir apresentar o fundo adequado à hora do dia em que o utilizador se encontra.

3.6 Serial Port (UART)

Aproveitamos a porta de série para possibilitar o jogo em modo *multiplayer*. Conforme os *interrupts* do rato e do teclado (remate e defesa) são enviados carateres universais e, de seguida, as respetivas coordenadas e área.

No outro lado, a cada interrupção da *serial port*, é processada a informação contida caso esteja disponível.

Para a sincronização das duas máquinas cada uma envia um carater dependente do jogador (1 ou 2) e quando ambos recebem mensagem do outro passam para o mesmo estado de iniciação ao jogo.

Para detetar se um jogador abandona o jogo antes de ser encontrado um vencedor caso um deles pressione ESC é enviado o carater corresponde ao *breakcode* da mesma.

4 Módulos implementados

4.1 Main.c

Neste módulo, são subscritos, o rato, o teclado e o timer para além de ser inicializado o modo de vídeo. No fim do jogo são interrompidas as subscrições e é fechado o modo de vídeo.

Este módulo foi realizado pelo Duarte Nuno André Lima de Carvalho, com um peso relativo de 3%.

4.2 Menu.c

Neste módulo, é apresentado o menu inicial com diferentes opções (“PLAY”, “ABOUT”, “EXIT”), sendo que a segunda opção abre um submenu (“SINGLE”, “MULTI” e “BACK”).

É criada uma estrutura *MenuInfo* com duas máquinas de estados, a *state_option* (contém os butões) *e a state_menu* (distingue se o utilizador se encontra no menu inicial ou no submenu), uma estrutura *Mouse mse (rato), dois inteiros selected e above (botões selecionados), uma estrutura Bitmap wallpapper (contêm o fundo), um vetor de estrutura Bitmap buttons (imagens dos botões quando pré-seleccionados) e, por fim, um vetor do tipo short background, que contem o buffer a ser impresso na VRAM quando solicitado.*

Quando é selecionado a opcão EXIT, é libertada toda a memória alocada à estrutura MenuInfo.

Este módulo foi realizado pelo Duarte Nuno André Lima de Carvalho, com um peso relativo de 10%.

4.3 About.c

Este módulo é responsável por toda a ação quando o utilizador entra na opção “ABOUT”. Apresenta as várias opções (“SHOOT”, “SAVE”, “CREDITS” e “BACK”), sendo que a última opção volta para o menu inicial.

É alocada uma estrutura A*bout,* que contém quatro estruturas, *wallpapper*(contém o *bitmap* de fundo), *buttons* (tem as imagens dos butões)*, info* (contém a informação que é impressa conforme a opção) e *mse* (estrutura *mouse*), um vetor do tipo *short background*, que contem o *buffer* a ser impresso na *VRAM* quando solicitado e três variáveis *int*, a *selected* (determina a opção selecionada), a *above* (determina se o rato está por cima de algum dos botões) e a *option* (determina se estamos dentro do menu about ou não).

Este módulo foi realizado pelo Duarte Nuno André Lima de Carvalho, com um peso relativo de 5%.

4.4 Game.c

Este módulo é responsável por toda a implementação do modo de jogo *Singleplayer* (Vs.COM), quer na parte lógica, quer na parte gráfica.

São criadas duas máquinas de estados: s*tate_game* e *state_mode*, sendo que a primeira máquina informa o programa sobre que animações apresentar e a segunda informa apenas que o jogo está numa fase normal ou numa fase morte súbita.

É também criada uma estrutura, *Game*, que inicializa as duas máquina anteriormente referidas e aloca cinco *Bitmaps*, *wallpapper* (contém o *bitmap* do fundo de jogo)*, numbers (*contém o *bitmap* para imprimir os números no marcador)*, score* (contém o *bitmap* do *scoreboard*)*, gameover* (contém os *bitmaps* correspondente à vitória do COM ou do jogador) e *referee* (contém o *bitmap* para desenhar o árbito). São também definidos arrays do tipo *short*, *slope*(inclinação ou ângulo de remate) *,background, background_stop, background_run, background_reaction* (os últimos quatro elementos referidos são *buffers* para o fundo, para quando o jogador está a parado, para quando o jogador vai rematar e para quando o jogador vai festejar) *e background_shoot* (contém os buffers relativos às três posições do guarda-redes). São também alocadas duas estruturas do tipo *player* (*user* e *com*) que contém todas as informações relativas aos movimentos dos jogadores. E, por último, uma estrutura do tipo *GoalKeeper* que contém as informações do guarda-redes.

Este módulo foi realizado por ambos os membros do grupo e tem um peso relativo de 15%.

4.5 Game_multi.c

Este módulo, tal como o módulo anteriormente descrito, é responsável por toda a implementação do modo de jogo *Multiplayer* (Vs.PLAYER2), quer na parte lógica, quer na parte gráfica.

São criadas três máquinas de estados: *state_game_multi, state_shoot_multi*, *state_mode_multi*. A primeira controla que jogador é que chuta, se a máquina está à espera do remate, se o jogador um ou o jogador dois está a correr, se as duas máquinas estão conectadas ou se está à espera de conexão. A segunda apenas indica ao jogo se se está numa fase de espera de conexão ou se já se está a jogar. A última indica se o jogo está numa fase normal ou numa fase de morte súbita.

É criada apenas uma estrutura, *Game_Multi*, que inicializa as três máquinas de estados, aloca seis estruturas *Bitmap* (*wall_wait*, *wallpapper*, *numbers*, *score*, *gameover* e *referee*), que contêm os bitmaps para, enquanto uma máquina está à espera de ser detectada ou de detetar alguma, apresentar ao utilizador, o fundo do jogo, os números para imprimir na *scoreboard*, a própria *scoreboard*, e os textos de vitória e desconexões dos jogadores, respectivamente. São também definidos arrays do tipo *short*, *slope*(inclinação ou ângulo de remate) *,background, background_stop, background_run, background_reaction* (os últimos quatro elementos referidos são *buffers* para o fundo, para quando o jogador está a parado, para quando o jogador vai rematar e para quando o jogador vai festejar) *e background_shoot* (contém os buffers relativos às três posições do guarda-redes). São ainda criadas duas estruturas do tipo *Player* (*p1* e *p2*), que contêm todas as informações relativas aos

movimentos dos jogadores, e uma estrutura do tipo *GoalKeeper*, que contém as informações relativas ao guarda-redes. E, ainda, uma variável *key*, do tipo *unsigned long*, onde é guardada a informação lida na porta de série.

Este módulo foi realizado pelo Bernardo Manuel Costa Barbosa, com um peso relativo de 15%.

4.6 Ball.c

Este módulo é responsável pelos movimentos da bola, assim como o seu desenho e possibilitar o trabalho da estrutura.

É criada uma estrutura do tipo *Ball*, que contém quatro variáveis do tipo *float* (x, y, *xspeed* e *yspeed*). As primeiras duas são o local exato da bola (coordenadas) e a outras duas são o que vai ser incrementado à variável x e à variável y a cada *frame*. São também criadas seis variáveis *int:* atual, *ticks*, *interrupts*, *xf*, *yf* e *collision*. As variáveis *xf* e *yf* são as coordenadas finais da bola. A variável *collision* indica se a bola colidiu com alguma coisa ou não e a variável atual indica o tamanho da bola (aspeto relacionado com questões de perspetiva). É ainda criado um duplo *array* do tipo *Bitmap*, *bmp*, que contém várias imagens da bola para dar a noção que esta a rodar e a afastar-se.

Este módulo foi realizado pelo Bernardo Manuel Costa Barbosa, com um peso relativo de 5%.

4.7 goalkeeper.c

Este módulo trata das frames do guarda-redes (que imagens mostrar), deteção das áreas escolhidas pelos utilizadores ou pelo *COM*.

Este módulo tem apenas uma estrutura (*GoalKeeper*) que contém quatro variáveis *int* (x, y, *key_n* e *save_area*). As primeiras duas variáveis é a posição do guarda-redes, já o *key_n* é o número de teclas premidas e a variável *save_area* é o, que depois de analisar as teclas premidas, vai informar a máquina do lado para que o redes se vai atirar. O vetor *keys* guarda as teclas premidas pelo utilizador. Depois também são criados dois duplos *arrays* do tipo *Bitmap* (*area* e *save*), em que o primeiro array são os vários retângulos de ajuda ao defensor e o segundo contém todos os *Bitmap* dos movimentos do guarda-redes.

Este módulo foi realizado pelo Bernardo Manuel Costa Barbosa, com um peso relativo de 5%.

4.8 player.c

Este módulo é responsável pelo movimento, pelos *frames* a apresentar dos jogadores a chutar a bola e a celebrar o golo.

Neste módulo está presente a estrutura *Player, constituída por* três *Bitmaps, stop, shoot e miss,* que contêm, respetivamente, os *buffers* para a impressão do jogador parado, a chutar e a sua expressão ao falhar. Depois temos dois *arrays* de *Bitmaps*, *goal* e *run*, que contêm os buffers para a celebração de golo e o movimento de corrida para a bola do jogador. Depois são criadas cinco variáveis *int* (*x, y, goals, attempts, atual)*, sendo que *x* e *y* é a posição do jogador,

goals e attempts são os golos marcados e o número de penaltis batidos, respetivamente. Depois temos a alocação de duas estruturas uma *GoalKeeper* e outra *Ball*.

Este módulo foi realizado pelo Bernardo Manuel Costa Barbosa, com um peso relativo de 5%.

4.9 rtc.c

Este ficheiro contém todas as funções relativas à interação com este periférico (RTC).

Contém apenas uma pequena *struct* chamada *hour_min* e é a esta estrutura que vamos buscar a hora de execução do programa para mudar os fundos.

Este módulo foi realizado pelo Duarte Nuno André Lima de Carvalho, com um peso relativo de 2%.

4.10 timer.c

Este módulo contém todas as funções necessárias para manipularmos o *timer0* de maneira a podermos integrá-lo nosso trabalho.

Para este módulo não foram usados quaisquer tipos de estruturas.

Este módulo foi realizado por ambos os membros do grupo e tem um peso relativo de 3%.

4.11 keyboard.c

Neste módulo estão contidas todas as funções para a deteção de *inputs* provenientes de *interrupts* do teclado, de maneira a ser-nos útil integrá-lo no nosso trabalho.

Para este módulo também não foram usados quaisquer tipos de estrutura.

Este módulo foi realizado por ambos os membros do grupo e tem um peso relativo de 3%.

4.12 Mouse.c

Neste módulo estão presentes todas as funções para a leitura dos movimentos do rato, de maneira a ser possível fazer parte do nosso projecto.

Neste módulo existe uma estrutura *Mouse*, que contém sete variáveis do tipo *unsigned int*: *X, Y, LB, MB, RB, XOV, YOV*. As primeiras duas variáveis detetam deslocamentos quer no eixo dos XX, quer no eixo dos YY, respectivamente, a variável LB deteta se o *Left Button* está premido, a RB deteta se o *Right Button* está premido, a MB deteta se o *Middle Button* está premido e, finalmente, a *XOV* e a *YOV* detetam se ocorreu *overflow* em qualquer uma das dimensões.

Este módulo foi realizado por ambos os membros do grupo e tem um peso relativo de 5%.

4.13 Bitmap.c

Neste módulo estão presentes todas as funções para a manipulação de gráficos e desenho de bitmaps, permitindo a interface gráfica e as animações do jogo.

Neste módulo está presente a estrutura Bitmap, com todo o crédito pertencente ao Henrique Ferrolho.

Este módulo foi realizado pelo Bernardo Manuel Costa Barbosa baseando-se no trabalho de Henrique Ferrolho, com um peso relativo de 5%.

4.14 serial.c

Neste módulo estão presentes todas as funções necessárias para a comunicação entre duas máquinas virtuais e ser possível o modo de *Multiplayer* .

Neste módulo não foram usadas estruturas.

Este módulo foi realizado pelo Bernardo Manuel Costa Barbosa, com um peso relativo de 5%.

4.15 vbe.c

Neste módulo encontram-se as funções que permitem guardar informação sobre o modo de vídeo em que o jogo está a correr e mais coisas relacionadas com a placa gráfica.

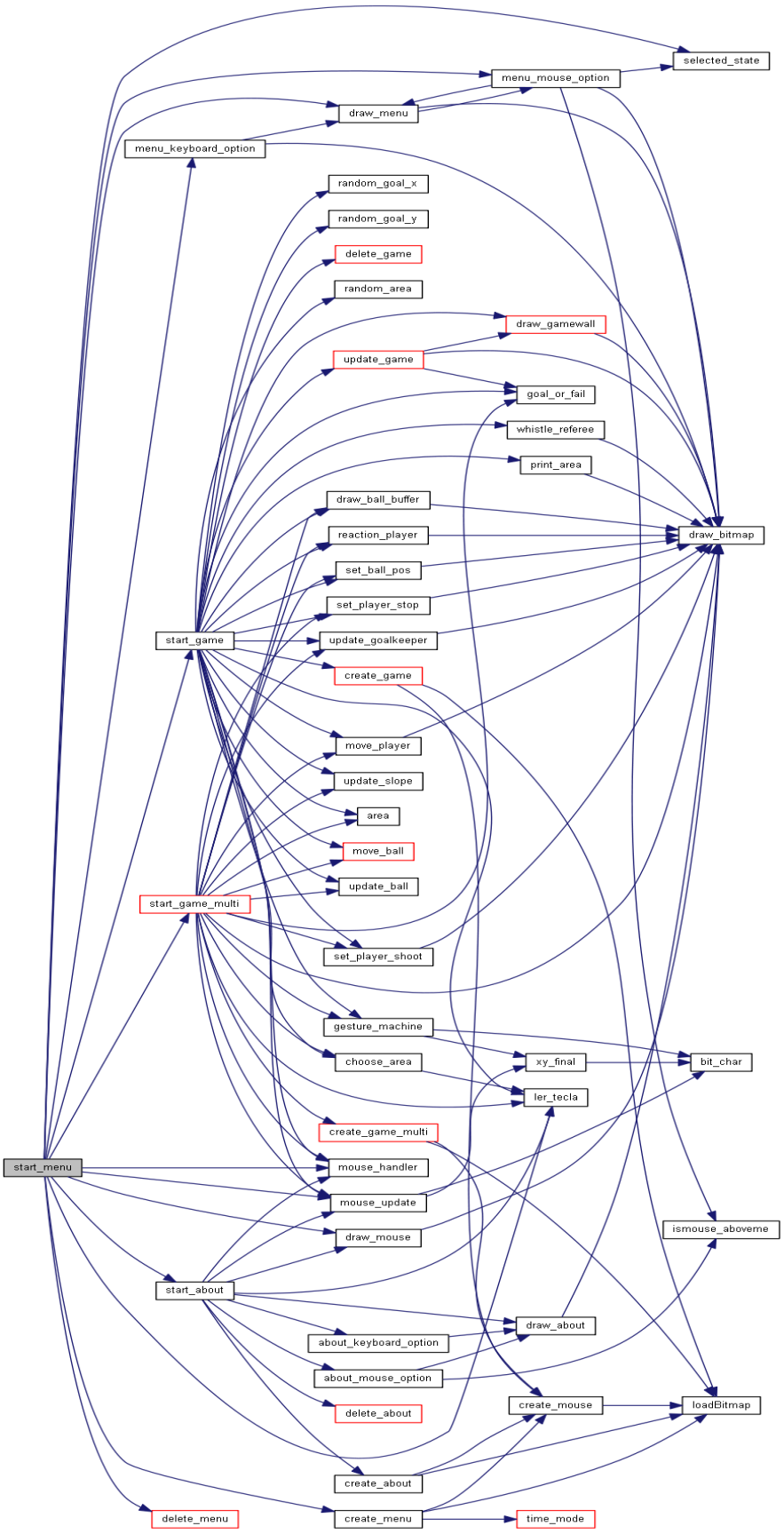
Este módulo foi realizado por ambos os membros do grupo e tem um peso relativo de 2%.

4.16 video_gr.c

Este módulo contém funções relacionadas com o carregamento de frames e gráficos do jogo e que permitem inicializar ou findar o modo de vídeo.

Este módulo foi realizado por ambos os membros do grupo e tem um peso relativo de 7%, denotando-se que o Bernardo Manuel Costa Barbosa fez as funções draw.

5 Gráficos de Chamadas de Funções

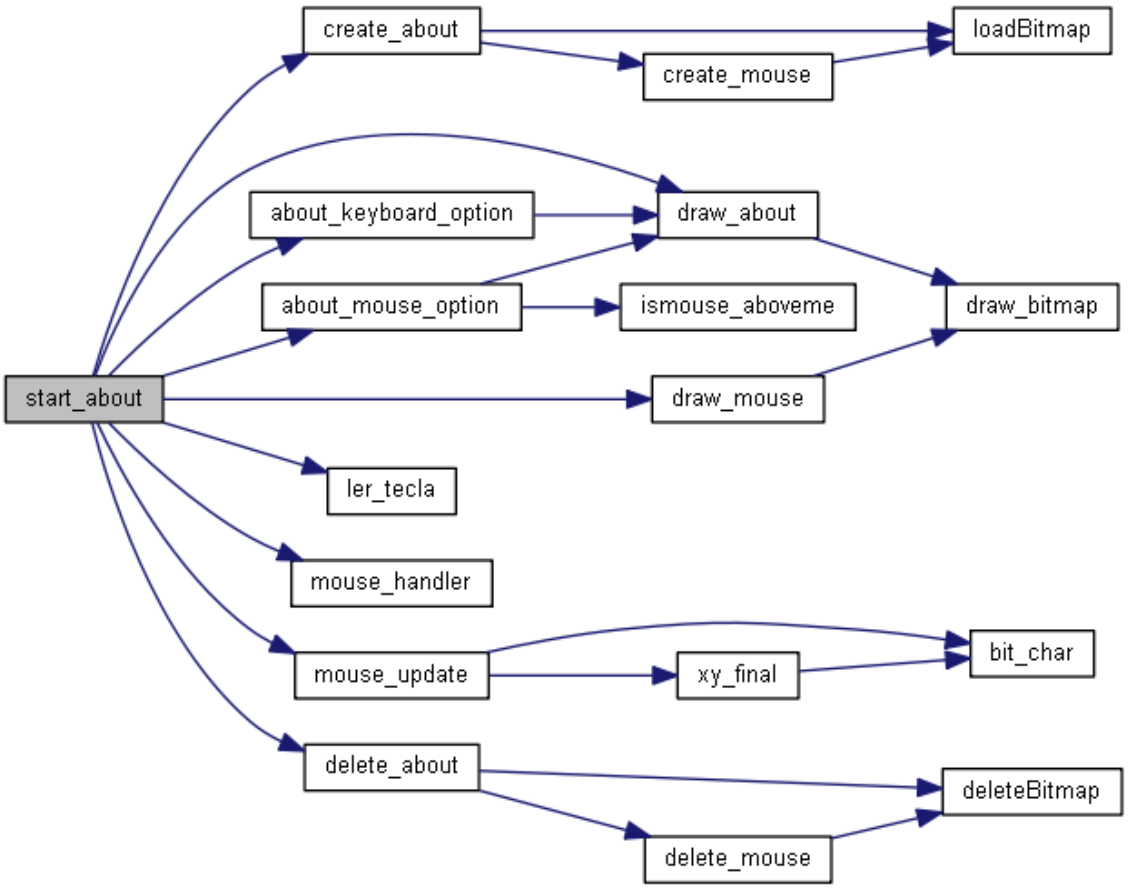


A função *start_menu()* é responsável pelo Menu Inicial do jogo, onde o jogador tem várias opções, desde aceder a breves instruções, aos créditos e aos diferentes modos de jogo que tem ao seu dispor.

Apoiada pela estrutura *MenuInfo*, começa por imprimir no buffer *background* a imagem de fundo, conforme a hora atual, e entra num ciclo onde receberá *interrupts* do rato, teclado e timer.

À medida que o utilizador interage com os vários periféricos, vai alterando os estados das duas máquinas de estados que o Menu contem, sendo no fim de cada ciclo verificado o estado e feitas ações (ou não) conforme o estado em que se encontra.

No fim, como já foi referido, é libertada toda a memória alocada à mesma estrutura.

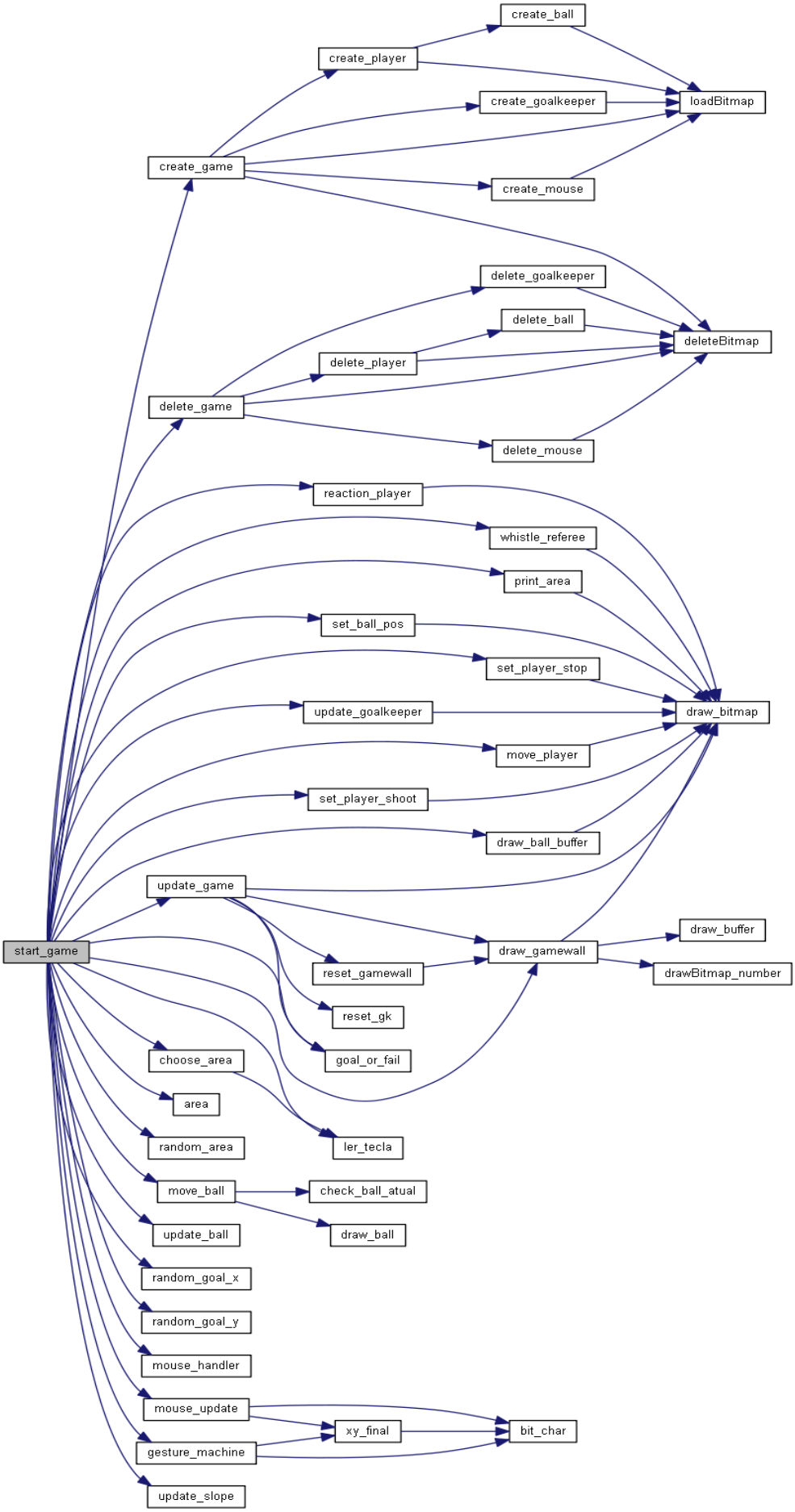


A função *start_about()* possui a responsabilidade de dar a entender ao utilizador como se usa os vários periféricos no decorrer do jogo e também os criadores deste projeto.

Tendo como base a estrutura *About,* imprime novamente no buffer *background* o fundo, e volta a entrar num ciclo que processa os interrupts vindos do rato e do teclado.

Conforme as ações do utilizador, são utilizadas funções que atualizam e analisam todas as variáveis da estrutura.

Por fim, novamente, é libertada toda a memória que esta alocou no início desta função.

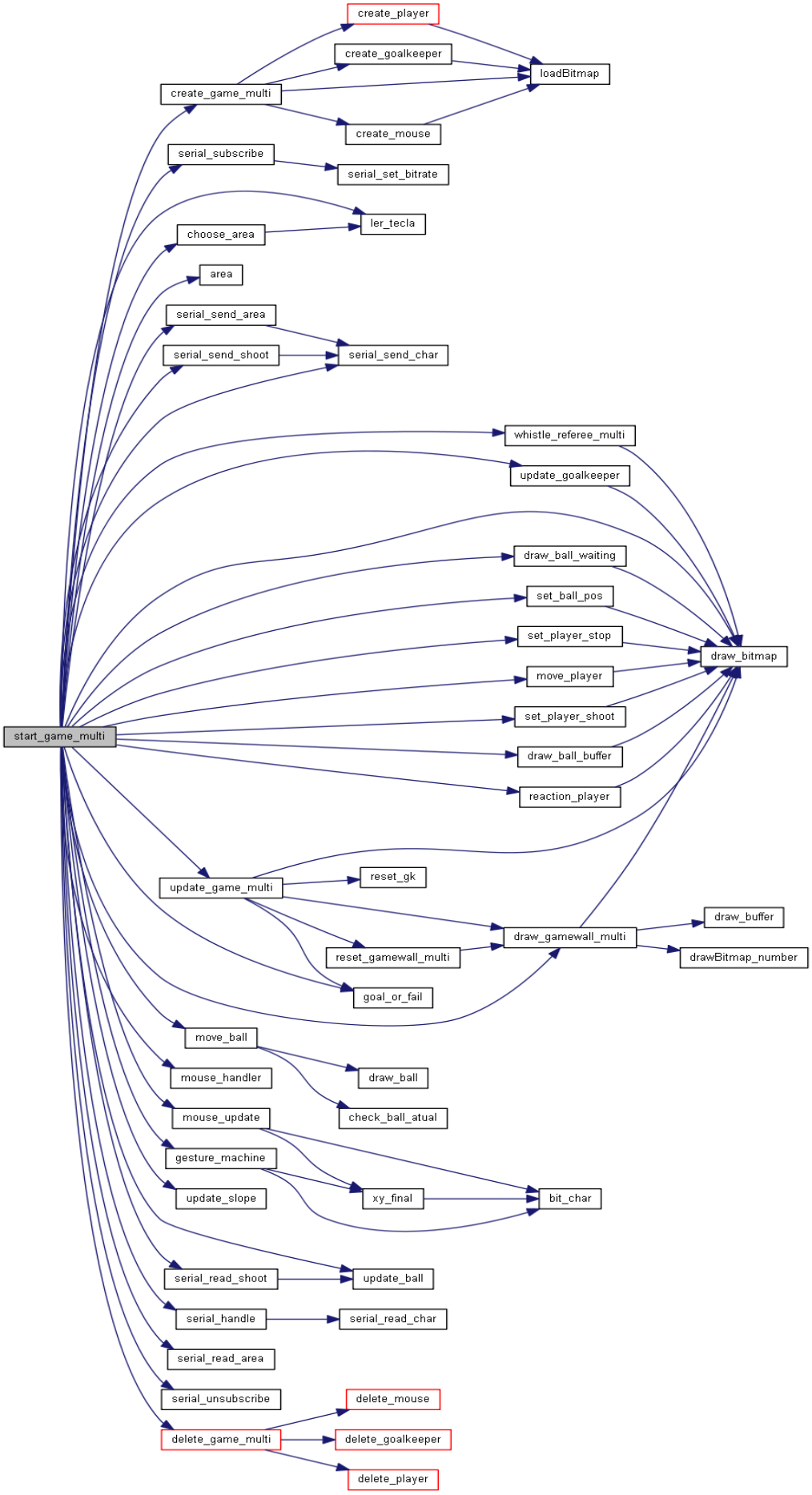

create_ball
create_player
loadBitmap
create_goalkeeper
create_game
create_mouse
delete_goalkeeper
delete_ball
deleteBitmap
delete_player
delete_game
delete_mouse
reaction_player
whistle_referee
print_area
set_ball_pos
set_player_stop
draw_bitmap
update_goalkeeper
move_player
set_player_shoot
draw_ball_buffer
update_game
draw_buffer
draw_gamewall
drawBitmap_number
start_game
reset_gamewall
reset_gk
goal_or_fail
choose_area
area
ler_tecla
random_area
move_ball
check_ball_atual
update_ball
draw_ball
random_goal_x
random_goal_y
mouse_handler
mouse_update
xy_final
bit_char
gesture_machine
update_slope

A função *start_game()* é usada para gerar a estrutura Game, que por sua vez é responsável por toda a implementação do jogo, desde imagens até ao marcador.

Suportada pela mesma, começa por imprimir o fundo de jogo já com o *scoreboard* e entra num ciclo onde irá receber e analisar os interrupts dos vários dispositivos subscritos, conforme o estado do jogo em que se encontra.

Mais uma vez, à medida que o utilizador interage com o programa, a função gere a máquina de estados da estrutura conforme a lógica de jogo que é analisada na função *update_game()*.

Para terminar, mais uma vez, é libertada toda a memória que a estrutura *Game* alocou.


create_player
create_goalkeeper
loadBitmap
create_game_multi
create_mouse
serial_subscribe
serial_set_bitrate
ler_tecla
choose_area
area
serial_send_area
serial_send_shoot
serial_send_char
whistle_referee_multi
update_goalkeeper
draw_ball_waiting
set_ball_pos
set_player_stop
draw_bitmap
move_player
set_player_shoot
start_game_multi
draw_ball_buffer
reaction_player
update_game_multi
reset_gk
draw_buffer
draw_gamewall_multi
drawBitmap_number
reset_gamewall_multi
goal_or_fail
move_ball
draw_ball
mouse_handler
check_ball_atual
mouse_update
gesture_machine
update_slope
xy_final
bit_char
serial_read_shoot
update_ball
serial_handle
serial_read_char
serial_read_area
serial_unsubscribe
delete_mouse
delete_game_multi
delete_goalkeeper
delete_player

A função *start_game_multi()* é idêntica à *start_game()*, usada para gerar a estrutura Game_Multi, que por sua vez é responsável por toda a implementação do jogo em modo multiplayer, desde imagens até ao marcador.

Diferenciada pela introdução da porta de série, começa por imprimir o fundo de jogo já com o *scoreboard* e entra num ciclo onde irá receber e analisar os interrupts dos vários dispositivos subscritos, conforme o estado do jogo em que se encontra. Nota para a adição de estados, nomeadamente em relação ao estado da conexão com o outro jogador, de modo a ser possível efetuar uma sincronização entre ambas as máquinas de estados.

Desta vez, à medida que os utilizadores interagem com o programa, a função gere a máquina de estados da estrutura conforme a informação recebida pela UART, sendo toda a lógica de jogo analisada na função *update_game_multi()*.

Para terminar, como sempre, é libertada toda a memória que a estrutura Game_Multi alocou.

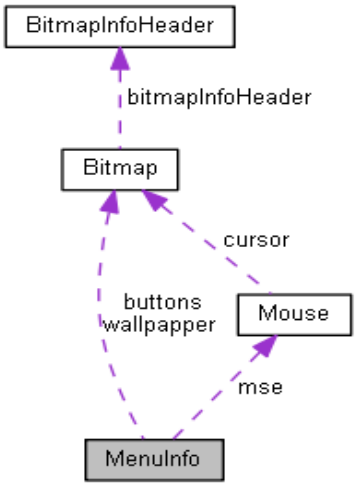


Figura 26 - MenuInfo Struct

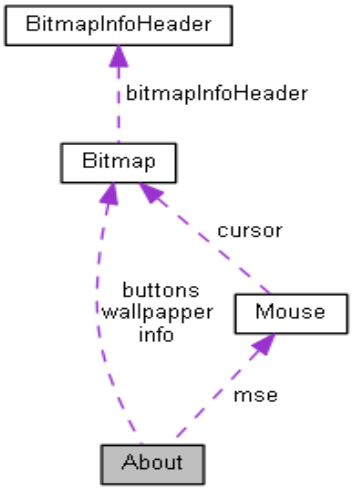


Figura 27 - About Struct

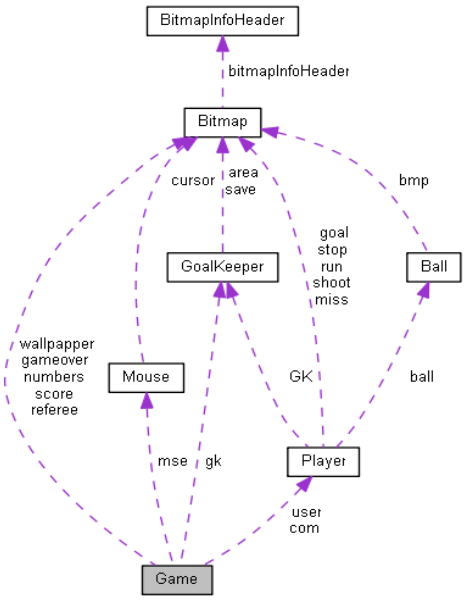


Figura 28 - Game Struct

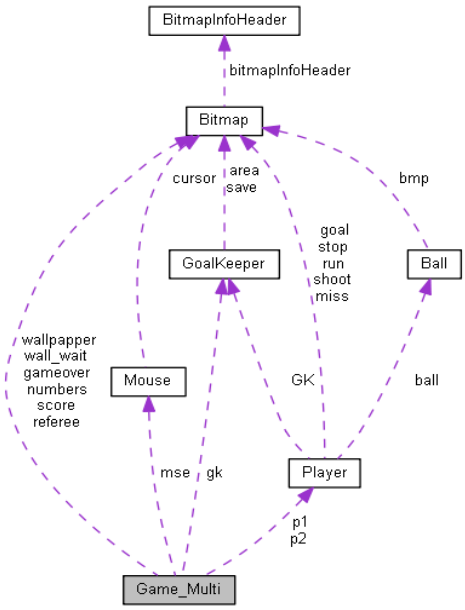


Figura 29 - Game_Multi Struct

6 Detalhes de Implementação

6.1 Timer

É utilizado apenas para controlo de frame rate das várias animações, controlo de tempo entre transição de estados e pouco mais.

Quanto a sua implementação, baseia-se na criação de uma variável "time" inicializada a 0, que é incrementada a cada interrupção do Timer, e reinicializada na transição entre estados.

Para facilitar o seu uso, foi criada uma macro SEC(x), em que x é tempo em segundos desejado, que devolve em inteiro o número de interrupções durante o espaço de tempo pretendido.

Quanto ao controlo de frame rate, é usado o resto da divisão do "time" pelo frame rate selecionado.

6.2 Teclado

Para o teclado, é apenas usada uma função *ler_tecla()* que tem como objetivo ler tanto os *makecodes* como os *breakcodes* das teclas pressionadas pelo utilizador, sendo depois processadas pelas respetivas funções.

Para simplificar o código e o seu uso, foram criadas macros relativamente às teclas *ARROW*, *ESC*, *TAB* e *ENTER*.

6.3 Rato

Quanto ao rato, foi implementada a estrutura *Mouse*, responsável por armazenar toda a informação descodificada dos *packets*, bem como a imagem do cursor respetiva.

Nota ainda para a função *gesture_machine*, responsável por obter o deslocamento do rato enquanto o *Right Button* está premido, de forma a obter a direção do remate.

6.4 RTC

Relativamente ao RTC, foram apenas implementadas as funções necessárias para obter as horas e minutos a que o utilizador inicia o jogo.

Para isso, obtemos em modo 0-24h o registo 4, e é chamada a função *rtc_get_hour()* que devolve em inteiro o número das horas.

6.5 Placa Gráfica

Para a placa gráfica, foram desenvolvidas várias funções de modo a facilitar tanto a escrita de imagens em buffers como a sua impressão na VRAM.

Antes de falar sobre isso, queríamos referir e agradecer ao Henrique Ferrolho pela partilha da biblioteca Bitmap, pois durante todo o projeto revelou ser fundamental para o concretizar da visão que ambos tivemos no início deste percurso.

Destacamos as funções *draw_bitmap()* e *draw_ball()*, responsáveis por toda a ligação entre Bitmap's, buffers e VRAM, onde a variável *background* corresponde ao buffer em uso.

Na função *draw_bitmap(),* é possível:

- Desenhar um Bitmap diretamente na VRAM, e depois copiar para **background* o que se encontra na VRAM;

- Copiar o **background* para um buffer auxiliar, desenhar o Bitmap no auxiliar e imprimi-lo na VRAM. (Jogador a correr / Movimento da Bola / Rato / Área de Defesa);
- Desenhar o Bitmap em *background* e imprimi-lo na VRAM;
- Desenhar o Bitmap em *background* sem que se imprima na VRAM (p.e. Buffers dos movimentos do Guarda-Redes).

Quanto à *draw_ball()*, é possível desenhar a bola e saber se esta colidiu com o guarda-redes ou não.

Como já referimos várias vezes ao longo do relatório, foram criados "buffers" em cada estrutura, de modo a poder usar o double buffering e a criação de cenários durante o jogo. Exemplo disso é o remate, em que o guarda-redes dá a ilusão de se mover, quando na verdade o jogo está apenas a mudar de índice no vetor de buffers *background_shoot*, os quais já foram trabalhados momentos antes do remate, sendo apenas necessário desenhar a bola por cima, em movimento.

6.6 Porta de Série – UART

Quanto à porta de série, talvez o maior desafio de todo este projeto, foram implementadas funções simples e diretas devido ao pouco tempo que tivemos para estudar este periférico.

Foi estabelecido uma taxa de bits standard de 1200 *bauds* e um *delay* de 10 ms, pois durante os nossos testes obtivemos bons resultados e ambos concordamos em seguir com estes valores.

Com base nos PDF's fornecidos e na página do Lab 7 de 2013, foi criada uma biblioteca com macros relativas aos vários registos do UART, de modo a facilitar a sua implementação.

Quanto à sua configuração, configuramos o Line Control Register com 8 bits, sem paridade e apenas com um bit de stop e ativamos apenas o Received Data Interrupt e o Receiver Line Status Interrupt, pois achamos desnecessário para a nossa implementação ativar o Transmitter Empty Interrupt.

Relativamente ao seu uso, apenas é usada durante o jogo em modo multiplayer, onde começa por estabelecer uma sincronização entre as duas máquinas, enviando um "key_match" que é influenciado pelo número do jogador (1 ou 2).

Durante o jogo em si, foram implementadas funções para o envio e receção de informação (coordenadas do remate e área de defesa), de forma a que sejam chamadas (as de envio) quando haja uma interrupção do rato ou do teclado e quando haja uma interrupção da porta de série (as de receção) , conforme o estado do jogo.

Nota para a variável *key* integrada na estrutura *Game_Multi*, que guarda a informação recebida caso esteja disponível para leitura.

Por fim, é de referir que há carateres que não são recebidos a tempo, causando apenas uma diferença na posição final do remate, problema este que raramente acontece e apenas ocorreu quando testado com o computador do Bernardo, que já tem alguma idade.

Pensamos que o objetivo foi alcançado apesar de que podíamos ter explorado um bocado mais a verificação da informação recebida (Erros de Paridade, Overrun e Framing).

7 Avaliação

Relativamente à nossa avaliação e à avaliação da unidade curricular, ambos temos pontos positivos e negativos a destacar.

Começando por nós, tiramos bastante partido do facto de a UC se basear em programação em C pois ambos concluímos as cadeiras de programação no curso de Engenharia Eletrotécnica na FEUP, daí estarmos completamente à vontade com qualquer conceito.

Por outro lado, sentimo-nos pouco à vontade com Assembly, daí não termos implementado qualquer tipo de função neste projeto, pois apenas tivemos poucas oportunidades de interagir com Assembly nestes dois anos no ensino superior.

Quanto ao projeto, refletimos e concordamos que a auto-avaliação se distribui da seguinte forma :

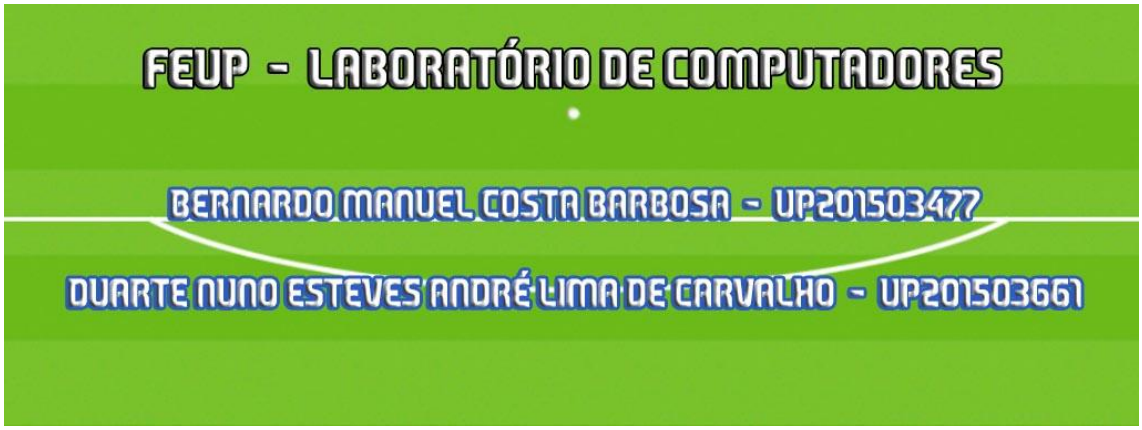
- 65 % - Bernardo Manuel Costa Barbosa
- 35 % - Duarte Nuno André Lima de Carvalho

Falando da UC, com o intuito de melhorar a qualidade da aprendizagem, temos alguns pontos a referir:

- Interação prática nas aulas teóricas;
- Aulas laboratoriais sobre RTC e UART;
- Maior atenção aos alunos das aulas, nomeadamente de segunda-feira, que não estão em pé de igualdade relativamente ao tempo que tem para estudar e preparar a aula prática, pois esta só era lançada às sextas da semana anterior;
- PPT das aulas laboratoriais mais informativos.
- Vale poucos créditos ECTS para o tempo que se dedica à UC.

Sem dúvida que esta foi a unidade curricular que mais nos cativou durante este semestre, estar em contacto com periféricos que usamos todos os dias vezes sem conta e poder perceber a “magia” por detrás da nossa ferramenta de trabalho é fantático, e acreditamos ser uma disciplina fundamental no curso em que estamos.

Foi um projeto interessante, pensamos que a ideia inicial foi conseguida apesar de todo o trabalho que exigiu, e o facto de termos conseguido a porta de série, elevou o nível do nosso jogo ao ponto de poder ser jogado entre dois computadores e se aproximar assim de uma aplicação do mundo real.



8 Anexos

8.1 Instalação do Penaltix

Para a instalação, esforçamo-nos para facilitar a mesma ao utilizador, de modo que criamos dois scripts que são responsáveis por copiar o *penaltix* para o sistema, descarregar as imagens e a biblioteca *liblm.a* e coloca-las no respetivo diretório, e por fim, compilar.

Para tal, basta seguir os seguintes passos:

```
$ cd proj/
$ chmod 777 conf.sh
$ su
# ./conf.sh
# ./install.sh
# cd src
# ./run.sh ( player 1)
# ./run_p2.sh ( player 2)
```

Nota para o script da execução do projeto, por norma, é executado o *run.sh*, dado que foi criado um *run_p2.sh* para possibilitar o modo multiplayer, visto que não conseguimos arranjar outra forma de com o mesmo código, o jogo entender que são 2 jogadores distintos e que, ao contrário de grande maioria dos jogos, as ações que praticam são contrárias (neste caso, quando um remata, o outro defende).

8.2 Bibliografia

Bitmap - http://difusal.blogspot.pt/2014/09/minixtutorial-8-loading-bmp-images.html

Imagens - https://images.google.com/?gws_rd=ssl

Adobe Photoshop CS6 - https://www.adobe.com/pt/products/photoshop.html

GIMP - https://www.gimp.org/

World Cup Penalty - http://www.gamesxl.com/sport/world-cup-penalty

CM 2010 GOLOS DE PENALTI - http://www.brincar.pt/cm-2010-golos-de-penalti.htm

8.3 Agradecimentos

- Professor Pedro Souto;
- Professor José Pinto;
- Henrique Ferrolho.

9 Índice de figuras