Faculdade de Engenharia da Universidade do Porto

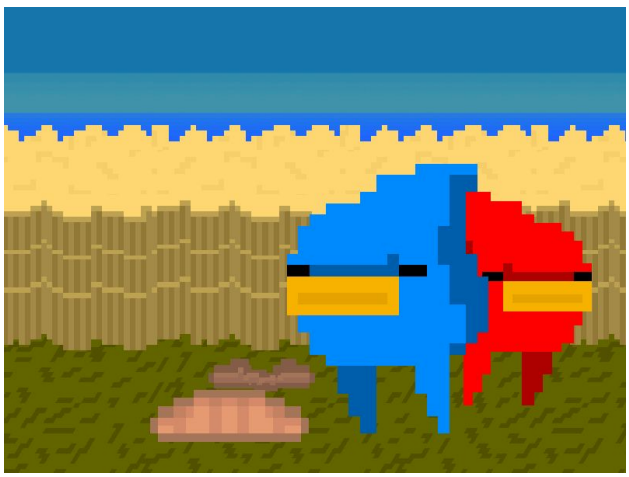
U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Ano Letivo 2015/2016

Unidade Curricular: Laboratório de Computadores

Top Down Shooter



Grupo T6G05:

David Azevedo up201405846@fe.up.pt

João Ferreira up201404332@fe.up.pt

Índice:

1.Instruções

1.1 Menu Principal

Figura 1 - Menu Principal do Top Down Shooter

Este é o menu principal do nosso jogo, mostra as personagens principais do jogo (a azul e vermelho) e ainda um dos inimigos (a minhoca), o menu apresenta as funcionalidades do programa, que consistem no modo *singleplayer* - botão “Play”, no modo *multiplayer* sendo *host* (isto é, ser o recetor de informação e *logic handler*) - botão “Create”, modo *multiplayer* sendo o segundo jogador (conectando-se a um servidor existente criado com o modo anterior numa máquina virtual ligada por porta de série) - botão “Connect” e por último a opção de saída - botão “Exit”.

Figura 2 - Rato sobre um botão

A interface do menu é possível com a utilização do rato, optamos por uma solução simplista e funcional e portanto o rato é representado por um quadrado de cor branca no ecrã como é possível observar na imagem (à esquerda), para indicar que o rato se encontra dentro de um botão clicável este altera de cor, tal pode ser visto na imagem (o rato encontra-se sobre o botão de “Play”). Clicar num dos botões levamos portanto até à opção representada pelo mesmo.

1.2 Modo Survival (*Singleplayer*)

Clicando no botão “Play” o jogo inicia-se o ecrã passar a mostrar o jogo (figura 3), o mapa consiste numa cruz onde o jogador é colocado inicialmente no ponto central e os inimigos aparecem nos extremos do mapa, na imagem é possível notar vários aspetos do jogo, o jogador (vermelho), a mira que representa a posição do rato, os inimigos, é ainda possível observar a destruição de terreno na parte superior esquerda da imagem.

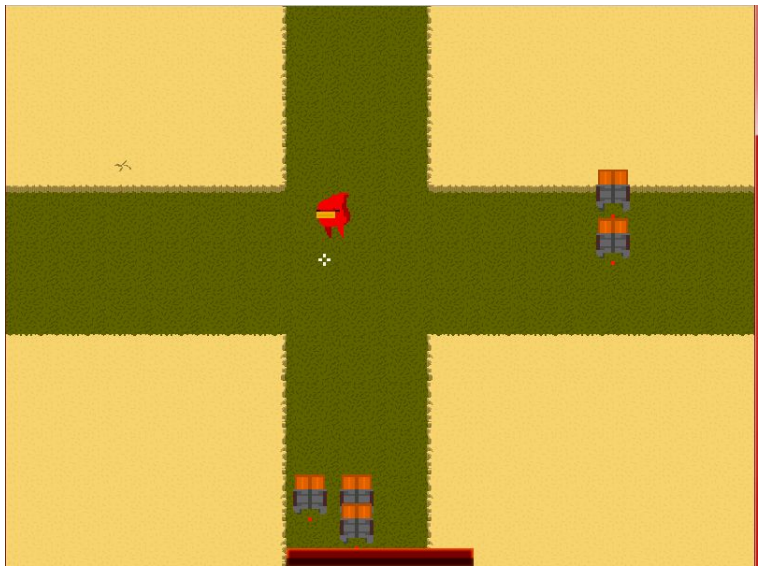


Figura 3 - Jogo na sua fase inicial

O objetivo do jogo é sobreviver o maior tempo possível com inimigos novos constantemente a aparecer assim que um inimigo morre. O jogo termina quando o jogador morre (barra vermelha na parte inferior do ecrã indica a vida do mesmo) ou pressionando a tecla ESC.

Figura 4 - Jogo

O movimento do jogador é controlado pelas teclas “WASD” (W - para cima ; A - para a esquerda ; S - para baixo ; D - para a direita), os disparos (possíveis de observar na figura 4) são realizados com o rato, isto é, ao pressionar o botão do lado esquerdo do rato o programa recolhe a posição do rato assim como a posição central do jogador (as balas são disparadas do centro da imagem do jogador), calcula a direções entre eles, aplica uma normalização de vetor para o tornar unitário e dispara a bala para onde o jogador estava a apontar quando pressionou o botão. Existem dois tipos de inimigos(que navegam pelo mapa com utilização de um algoritmo chamado de A* pathfinding), as minhocas e os robôs, ambos os *mobs* possuem características distintas: os robôs - são mais rápidos e explodem ao serem mortos como é possível observar nas imagens, por isso têm também uma vida inferior, as minhocas - são mais lentas mas o seu *hp* é maior. É importante referir que quando um inimigo morre é aplicado um decal na sua posição onde isto ocorreu mostrando o seu “corpo morto”. (Ver figura 5)

Figura 5 - Uma parte já mais avançada do jogo

Figura 6 - Fim do jogo

Como já foi referido o jogo termina quando o jogador morre quando isso acontece, é utilizado o RTC e determina-se a quantidade de segundos que o jogo durante mostrando a imagem que se vê na figura 6 a indicar que o jogo acabou e o tempo em hh:mm:ss que o jogador consegui sobreviver. Decorrido algum tempo (o tempo que nós consideramos necessário para o utilizador ler e absorver esta informação) o jogo retorna ao menu inicial.

1.3 Modo *Multiplayer*

Figura 7 - Esperando por um servidor

Figura 8 - Esperando por um jogador

Selecionando a opção “Create”, aparece-nos a figura 7, estando o programa a espera que um jogador se conecte ao servidor criado, por outro lado quando é selecionada a opção “Connect” aparece-nos a figura 6 indicando que o programa está à espera de uma mensagem de confirmação do servidor para iniciar o jogo, apos todas as premissas terem sido verificadas o jogo inicia-se (figura 8).
Para que a conexão seja sucedida será necessário inicializar primeiro o servidor e só depois conectar.

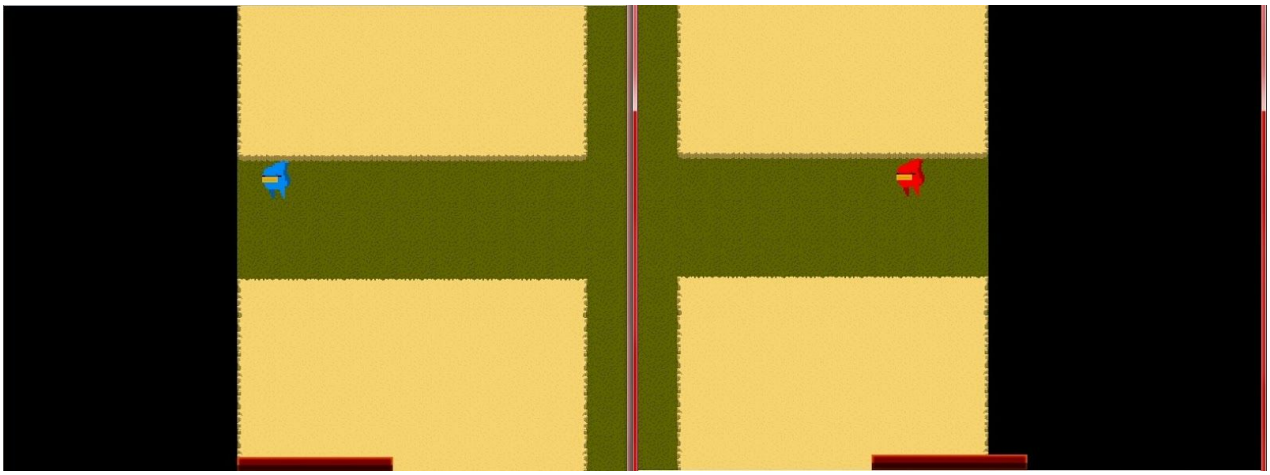


Figura 9 - Inicio do Multiplayer

Por limitações relacionadas com o uart no modo *multiplayer* simplesmente temos dois jogadores que destroem o terreno à sua volta numa especie de “explosão” não tem objetivos o jogo, o objetivo desde modo é simplesmente mostrar o funcionamento da porta de série.

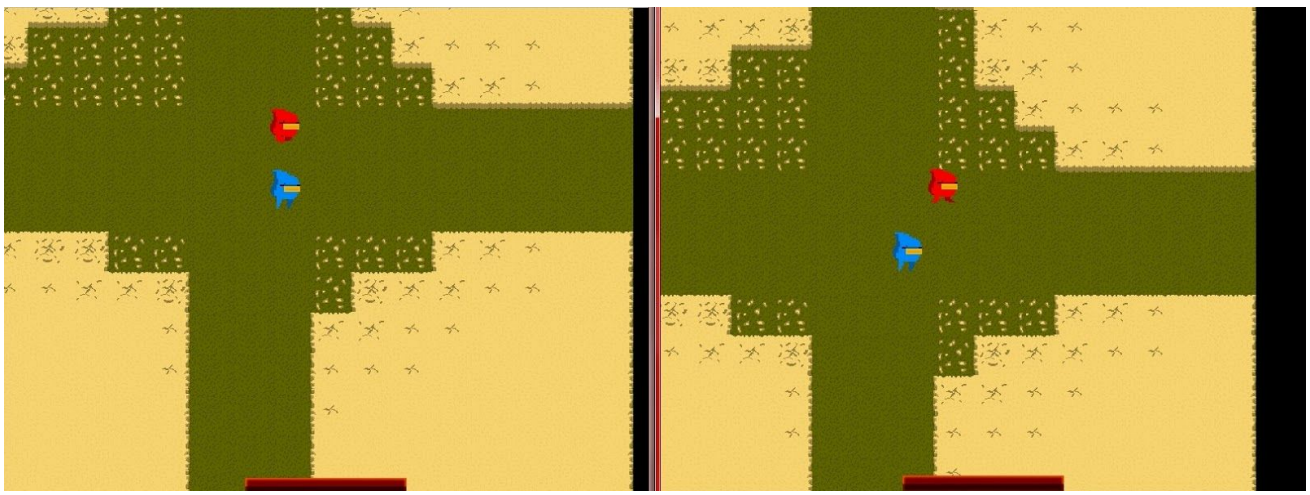


Figura 10 - Jogo multiplayer

2.Estado do Projeto

Em relação a proposta e especificação a versão final do projeto ficou com mais conteúdo e funcionalidades que facilitaram a organização e produtividade do projeto como por exemplo o *Object Manager* que apaga, desenha e atualiza os objetos que lhe são atribuídos(para uma melhor explicação ver o ponto 3.12). Apenas duas funcionalidades não foram adicionadas ao projeto por motivos de tempo e de prioridade sendo elas Filtros de cor no ecrã e *PerlinNoise* para geração de mapas aleatórios baseados em *seeds*.

2.1 Tabela de Dispositivos de Entrada/Saída.

Dispositivo	**Utilidade**	**Modo de Uso**
Teclado	Movimentação das personagens	Interrupções
Rato	Seleção no menu e como mira do jogador	Interrupções
Timer	Controlo da frame-rate	Interrupções
RTC	Tempo sobrevivido	Polling
UART	Multijogador entre dois PC's	Interrupções/Polling
Placa Gráfica	Representação gráfica do jogo	?

2.2 Teclado

O teclado é usado no projeto como forma de o utilizador interagir com o jogo é utilizado como forma de controlo do movimento da personagem. Permite que sejam pressionadas várias teclas em simultâneo sem qualquer interferência.
Código relacionado com o teclado está contido em keyboard.c

2.3 Rato

Possui toda a informação relativa aos botões pressionados (Em simultâneo ou não) e posição global, também é possível obter a posição relativa ao ecrã. O rato é utilizado para navegar o menu e selecionar a opção dentro do mesmo. No jogo o rato é utilizado como mira, e quando é pressionado é disparada uma bala que parte do jogador e vai na direção da posição onde se encontrava o rato no momento do clique.
Código relacionado com o rato está contido em mouse.c

2.4 Timer

O timer tem como objetivo o controlo do *frame rate* do jogo. O código respetivo encontra-se no ficheiro menu.c.

2.5 RTC

No nosso projeto o rtc é utilizado para obter a data atual. No inicio do jogo é chamada essa função e guardado o valor,no fim do jogo repete-se o processo e determina-se a duração do jogo. As funções encontram-se no ficheiro rtc.c e as suas chamadas no programa no ficheiro menu.c (linhas 211,247 e 251). A informação da hora é guardada em segundos totais e posteriormente convertida, no formato hh:mm:ss, numa string.

2.6 UART

Funciona em modo de interrupção e em modo *polled* e utiliza o FIFO, o modo polled é usado na inicialização do UART para limpar o conteúdo presente no FIFO. É possível personalizar a configuração do UART com o uso do metodo UART_SetConfiguration presente em uart.c.
Tem como parâmetros:
Word Length: 8
No. stop bits : 1
Paridade: 0
Bit-rate divisor: 1200

2.7 Placa Gráfica

A placa gráfica é usada no modo de video 0x114 (800 x 600 RGB-565)video mode utiliza double buffering para que a transição entre ecrãs seja o mais fluida possivél. Para mover objetos o ecrã é completamente apagado e desenhado de novo. É possivél carregar bitmaps e fontes.

3. Organização de código

3.1 Bullet

Este é o modulo responsável por tudo relacionado com as balas. À medida que vai atualizando a bala verifica se esta colidiu com um inimigo ou o terreno e aplica o dano no mesmo.
Responsável: David Azevedo
Importância: 2%

3.2 Debug

A função deste módulo foi de auxiliar o desenvolvimento do projeto, não é necessária a inicialização deste módulo pois ele inicializa-se automaticamente quando tem uma mensagem para escrever, foi possível fazer debug de strings inteiros e código em hexadecimal graças a este módulo.
Responsável: João Ferreira
Importância: 1%

3.3 Enemy

Módulo responsável por dar vida aos Inimigos do jogo, ou seja, animar, movimentar, desenhar, sincronizar a posição com o UART(Descartado do projeto final).
Responsável: João Ferreira
Importância: 5%

3.4 EnemyManager

Módulo responsável pelo controlo de grupos de inimigos, permite a atualização e desenho e atualiza a lista de posições de cada inimigo até ao jogador.
Responsável: João Ferreira
Importância: 6%

3.5 Font

Módulo responsável por mostrar texto no ecrã. É utilizado quer no menu quer no fim do jogo para relatar informações e apresentar opções ao utilizador.
Responsável: João Ferreira
Importância: 2,5%

3.6 Keyboard

Módulo responsável pela interpretação do *Input* do Teclado, com este módulo é possível saber se múltiplas teclas foram pressionadas ao mesmo tempo de um método simples e comodo.
Responsável: David Azevedo
Importância: 6%

3.7 List

Perante a necessidade de uma estrutura dados foi criado este modulo, que implementa uma lista permitindo guarda vários elementos do mesmo tipo numa ordem sequencial para facilitar o seu acesso.
Responsável: João Ferreira
Importância: 2%

3.8 Map

Módulo responsável pelo desenho, texturização, atualização, destruição, calculo de trajetória, sincronização e geração do mapa do jogo.
Responsável: João Ferreira
Importância: 10%

3.9 Menu

Modulo responsável pelo jogo. Este modulo é por outras palavras, o *loop* do jogo, isto é, o jogo enquanto se desenrola encontra-se sempre neste modulo, ele tratar a informação e chama as funções necessárias (menu,jogo,*multiplayer*) ao jogo. Contem um *function pointer* que aponta para a função do estado atual do jogo.
Responsável: David Azevedo
Importância: 8%

3.10 Mouse

Módulo responsável pela interpretação do *input* do rato, é permitido saber a posição global e local(relação a janela) do rato e também se múltiplos botões do rato estão a ser pressionados.
Responsável: David Azevedo
Importância: 6%

3.11 Network Manager

Módulo responsável pela recepção, interpretação e envio de mensagens, também é responsável pera verificação da conectividade de um utilizador ao nosso computador.
Responsável: João Ferreira
Importância: 5%

3.12 Object Manager

Módulo responsável pelo desenho, atualização e eliminação de objetos do jogo de forma automática
Responsável: João Ferreira
Importância: 10%

3.13 Particle

Modulo responsável pelas partículas de explosão relacionadas com o robôs. Este modulo tem uma funcionalidade muito parecida ao modulo *Bullet*. Trata da animação da mesma.
Responsável: João Ferreira
Importância: 2%

3.14 Player

O modulo *Player* assim como o nome indica trata da informação relativa ao jogador. Criação e destruição , *update* e animação, sincronizar a posição com o UART. Se a variável *hp* do jogador for igual a zero, o jogo acaba.
Responsável: David Azevedo
Importância: 5%

3.15 Rectangle

Este modulo é utiliza primariamente para a verificação de colisões e também utilizado para o dimensionamento de *sub-sprites*.
Responsável: João Ferreira
Importância: 5%

3.16 RTC

Utilizado para acessar à hora do pc. De forma a determinar tempo decorrido.
Responsável: David Azevedo
Importância: 2%

3.17 Sprite

Módulo responsável pelo desenho e armazenamento de bmp's. Também é possível o desenho de *subsprites* para que animações sejam possíveis.
Responsável: João Ferreira
Importância: 4,5%

3.18 TextureManager

Módulo responsável por armazenar todas as texturas do jogo para que não haja necessidade de duplicar a mesma textura em memória.
Responsável: João Ferreira
Importância: 3%

3.19 Timer

Modulo responsável essencialmente pelo controlo do *frame rate* do jogo.
Responsável: David Azevedo
Importância: 2%

3.20 UART

Módulo responsável pela comunicação com a porta UART, ou seja, responsável por recepção e envio de mensagens, esvaziamento do FIFO e por guardar a informação a enviar e recebida em listas.
Responsável: João Ferreira
Importância: 3%

3.21 Vector

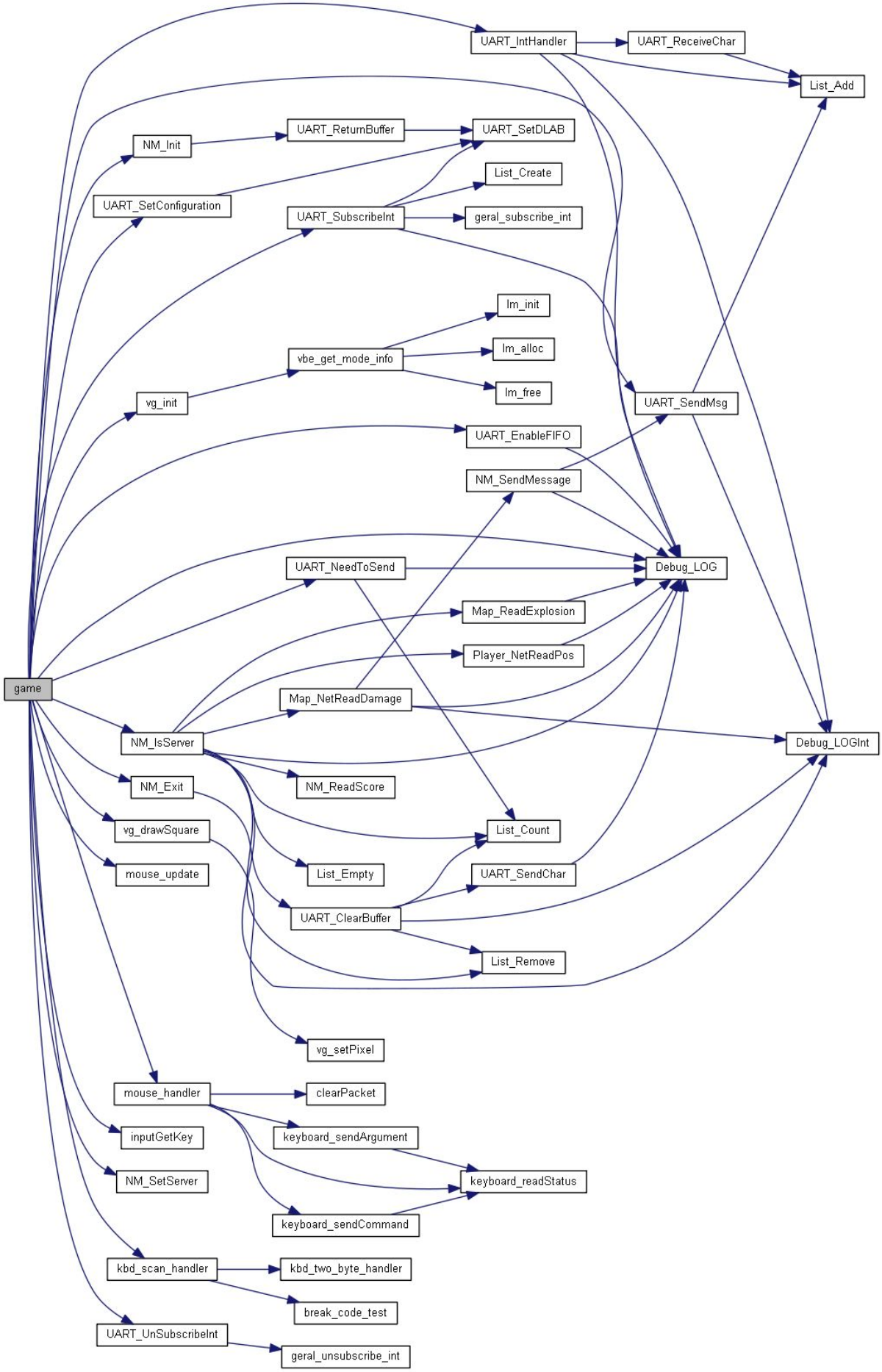
Módulo responsável pela realização de operações relacionadas com vetores bidimensionais.
Responsável: João Ferreira
Importância: 3%

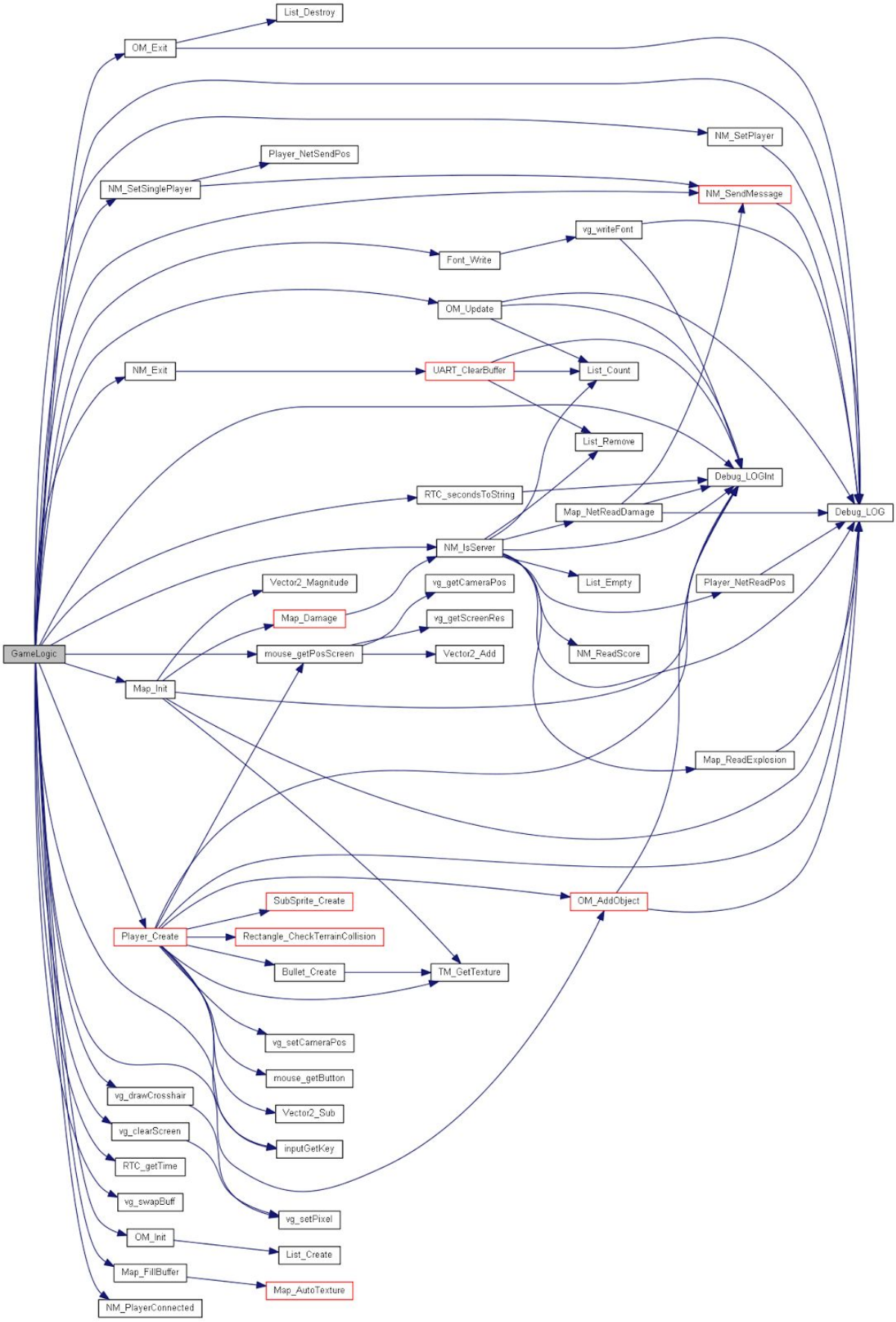
3.22 Video_gr

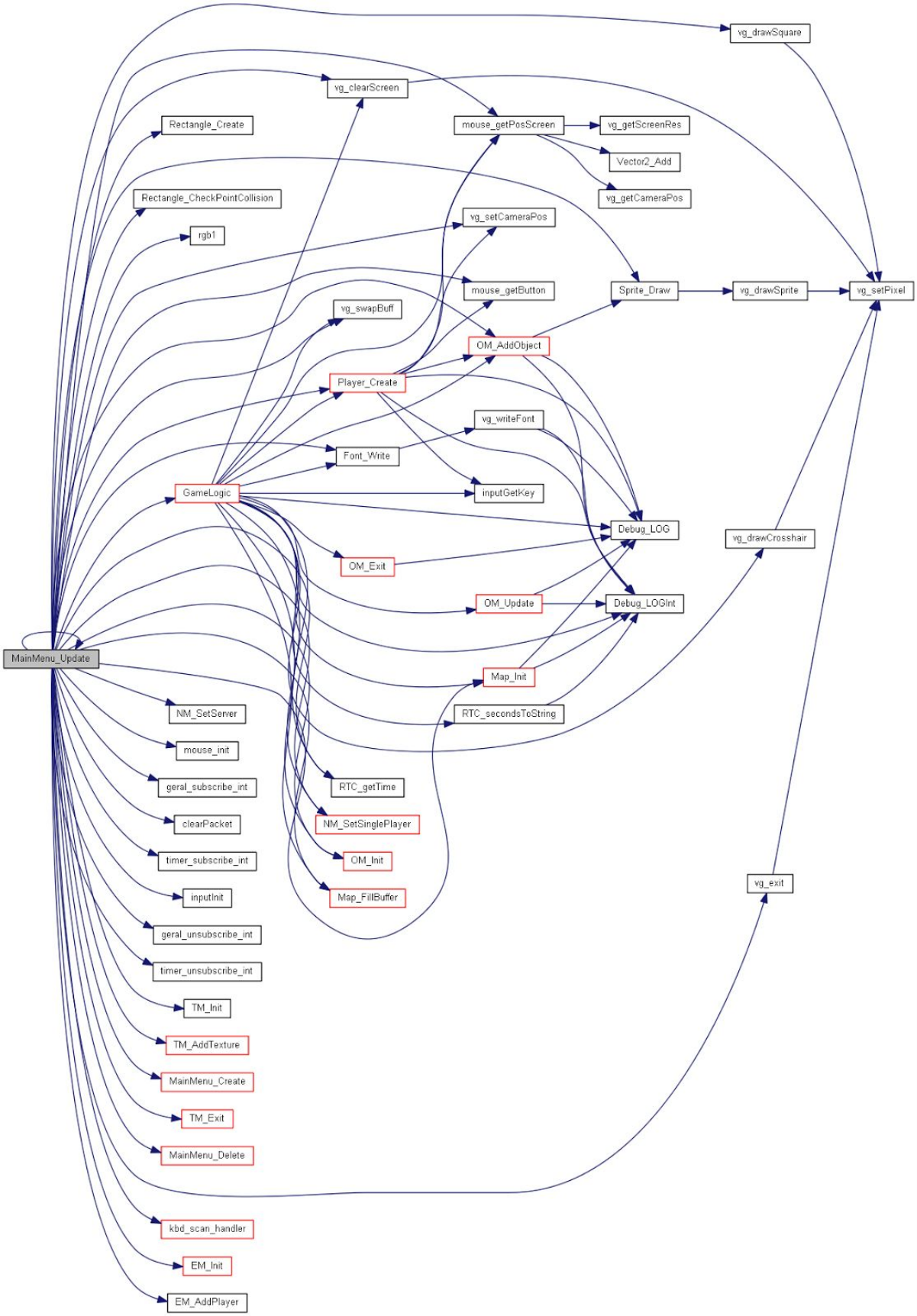
Módulo base responsável pelo desenho de *bitmaps*, fontes. retângulos, quadrados, pontos, crosshair, *double buffering*, mapa, *sprites*, *subsprites*, inicialização em modo de vídeo e obtenção e definição da posição da câmera.
Responsável: David Azevedo
Importância: 7%

3.23 Gráfico de chamadas de funções

A razão pela qual são demonstradas 3 *function call graphs* é explicada pelo facto de termos utilizado um *function pointer* para chamar a função que se está a desenrolar no momento seja ela, menu/jogo/*multiplayer*, segundo o que verificamos que o *doxygen* não realiza o *call graph* corretamente devido a este pormenor, pelo que optamos por mostrar os 3 gráficos sendo que os últimos dois são chamados pela função ***game*** do primeiro gráfico.


UART_IntHandler
UART_ReceiveChar
List_Add
NM_Init
UART_ReturnBuffer
UART_SetDLAB
List_Create
UART_SetConfiguration
UART_SubscribeInt
geral_subscribe_int
lm_init
vbe_get_mode_info
lm_alloc
lm_free
vg_init
UART_SendMsg
UART_EnableFIFO
NM_SendMessage
UART_NeedToSend
Debug_LOG
Map_ReadExplosion
Player_NetReadPos
game
Map_NetReadDamage
NM_IsServer
Debug_LOGInt
NM_Exit
NM_ReadScore
vg_drawSquare
List_Count
mouse_update
List_Empty
UART_SendChar
UART_ClearBuffer
List_Remove
vg_setPixel
mouse_handler
clearPacket
inputGetKey
keyboard_sendArgument
NM_SetServer
keyboard_readStatus
keyboard_sendCommand
kbd_scan_handler
kbd_two_byte_handler
break_code_test
UART_UnSubscribeInt
geral_unsubscribe_int


List_Destroy
OM_Exit
NM_SetPlayer
Player_NetSendPos
NM_SetSinglePlayer
NM_SendMessage
vg_writeFont
Font_Write
OM_Update
NM_Exit
UART_ClearBuffer
List_Count
List_Remove
Debug_LOGInt
RTC_secondsToString
Map_NetReadDamage
Debug_LOG
NM_IsServer
Vector2_Magnitude
vg_getCameraPos
List_Empty
Player_NetReadPos
Map_Damage
vg_getScreenRes
GameLogic
mouse_getPosScreen
Vector2_Add
NM_ReadScore
Map_Init
Map_ReadExplosion
OM_AddObject
SubSprite_Create
Player_Create
Rectangle_CheckTerrainCollision
Bullet_Create
TM_GetTexture
vg_setCameraPos
mouse_getButton
vg_drawCrosshair
Vector2_Sub
vg_clearScreen
inputGetKey
RTC_getTime
vg_swapBuff
vg_setPixel
OM_Init
List_Create
Map_FillBuffer
Map_AutoTexture
NM_PlayerConnected


vg_drawSquare
vg_clearScreen
Rectangle_Create
mouse_getPosScreen
vg_getScreenRes
Vector2_Add
vg_getCameraPos
Rectangle_CheckPointCollision
vg_setCameraPos
rgb1
mouse_getButton
Sprite_Draw
vg_drawSprite
vg_setPixel
vg_swapBuff
OM_AddObject
Player_Create
vg_writeFont
Font_Write
GameLogic
inputGetKey
Debug_LOG
vg_drawCrosshair
OM_Exit
OM_Update
Debug_LOGInt
MainMenu_Update
Map_Init
NM_SetServer
RTC_secondsToString
mouse_init
geral_subscribe_int
RTC_getTime
clearPacket
NM_SetSinglePlayer
timer_subscribe_int
OM_Init
vg_exit
inputInit
Map_FillBuffer
geral_unsubscribe_int
timer_unsubscribe_int
TM_Init
TM_AddTexture
MainMenu_Create
TM_Exit
MainMenu_Delete
kbd_scan_handler
EM_Init
EM_AddPlayer

4. Detalhes de Implementação

Durante a primeira fase de desenvolvimento foi optado por se desenvolver primeiro o *timer*, o teclado, o rato e a placa de vídeo.

No Menu, optamos por utilizar uma solução baseada em apontadores para funções, algo que foi abordado de forma leve nas aulas e que tivemos a curiosidade de aprender mais, pelo que decidimos utilizar esses conhecimentos na realização do projeto. Outro motivo é também o facto de após nos ter sido mostrado uma implementação usando máquinas de estado pelo Henrique Ferrolho na aula por ele dada, termos decidido fazer de modo diferente uma vez que evitamos o risco de sermos acusados de plágio.

Para o teclado surgiu um problema, se fosse preciso pressionar teclas simultaneamente havia erros de *input* e o resultado não era o desejado, por isso, foi criado um *array* do tipo *char,* que contém todos os estados das teclas, quando uma tecla for pressionada (makecode) a posição no *array* destinada a essa tecla passará a 1(ativa) e caso em que *keyboard handler* receba um (*breakcode*) essa posição passará a 0(inativa) assim uns simples métodos get e set foram o suficiente para obter qualquer informação necessária, relativa ás teclas pressionadas. Estes métodos estão contidos no ficheiro keyboard.c.

Para o rato foi criada uma estrutura de dados capaz de conter informação relativa a posição e estado dos botões pressionados do rato, esta aplicação foi simples e intuitiva de ser feita e está representada no ficheiro mouse.c.

Como a parte visual do jogo ainda estava muito limitada com a ajuda da Wikipédia para perceber o formato de um *bitmap*, foi criado o módulo bmp capaz de ler o conteúdo de um ficheiro no formato .bmp e armazená-lo num novo módulo *Sprite*. Este módulo *Sprite* já possui a informação relativa do *bitmap* numa forma simples e ordenada para que a imagem seja desenhada de cima para baixo e numa posição relativa do ecrã.

Após a criação dos *Sprites* foram criadas classes relativas ao jogo, como não podia faltar foram implementados os objetos *Player*, *Enemy*, *Bullet* e Particula que possuem de alguma forma uma identificação para a sua textura e métodos de atualização(update), desenho(draw) e de eliminação(delete) para que a logica do jogo siga um padrão. Estes métodos vão ser importantes num passo mais a frente.

É também importante que referir que os objetos, como por exemplo, os inimigos e também o jogador utilizam máquinas de estados de forma a realizar a correta animação aplicando a textura adequada e no caso dos inimigos se está é do tipo A ou do tipo B.

No caso das balas, foi preciso rever conteúdos programáticos de álgebra linear uma vez que foi criada uma função de normalização de um vetor de forma as balas terem a direção correta quando o jogador a dispara.

Para que a jogabilidade seja mais interessante foi adicionado o módulo map que é capaz de de desenhar um mapa destrutível, texturar de acordo com o estado do mapa, ou seja, analisa todos os pontos e verifica a vizinhança á sua volta e finalmente textura a partir de uma imagem, que possui todas as possibilidades de junção entre os pontos, a sub imagem respetiva, também é capaz de devolver a trajetória mais curta entre dois pontos, para essa possibilidade foi implementado o algoritmo A* pathfinding,(pessoalmente, não me lembro da fonte onde aprendi o algoritmo já que se passaram uns 2/3 anos, João Ferreira). O módulo map possui métodos também possui objetos relativos á sua atualização(update) e ao seu desenho(draw).

Com bastantes objetos num cenário que se destroem frequentemente(Particle e Bullet) e para uma fácil leitura do código foi criado um *objetmanager* que possui uma lista com todos os objetos que estão a ser desenhados e a camada(ordem) onde serão desenhados, para isso foram implementadas maquinas de estado no momento em que se adiciona um objeto á lista do *objectmanager* para saber se o objeto irá ser adicionado como UI(user interface), primeiro plano, segundo plano e terceiro plano, para saber o objeto que foi adicionado também é necessário guardar informação relativa ao tipo de objeto, por isso o que é adicionado á lista é uma estrutura de dados com um apontador genérico (void*) o objeto destinado e o tipo do objeto. O objectmanager possui um método de update que permite a atualização e desenho de todos os objetos contidos na lista, para isso mais uma vez foi implementada uma maquina de estados para determinar o tipo de objeto e para realizar as respectivas operações de acordo a todos os diferentes objetos. Também possui um método de saída(exit) para eliminar todos os objetos da lista e limpar toda a memória ocupada por todos os objetos e pelo próprio *objectmanager*.

Foi implementada o RTC de forma um pouco leve devido a constrangimentos relacionados com prioridades de outros módulos e também de tempo, pelo que não utilizamos interrupções do RTC mas a matéria foi de facto revista e estudada pelos elementos do grupo, só que no fim acabou por sofrer uma implementação um pouco mais minimalista no projeto.

Após um singleplayer estável foi implementado o UART em modo interrupção e com a utilização de FIFOs para a transmissão mais eficiente de dados, também foi implementada uma lista com informação relativa aos dados de entrada e outra para armazenar os dados de saída, para o tratamento dos dados de entrada foi criado o networkmanager que usa uma máquina de estado e converte os objetos de uma lista numa string e que trata essa informação conforme for preciso, assim foi possível trocar informação relativa á posição do jogador, explosões entre outras. Vale a pena informar que para transformar o movimento do jogador mais fluido foi usado um método de lerp entre o jogador e a nova posição recebida, esta execução está descrita em player.c, método Player_SyncPos linha 96.

Com um problema de ter que carregar múltiplas texturas para as balas ou partículas foi criado um texture manager que cria um array de sprites, para organização do projeto foi criado um enumerador chamado que contem a posição de cada Sprite. Também foi criado um método de saída para libertar memória.

5. Conclusão

5.1 Avaliação do curso

- O curso requerer um esforço acima daquilo que vale em termos de ECTS, achamos que a cadeira apesar de essencial deveria valer mais ECTS, pelo menos comparativamente a outras cadeiras.
- Falta de informação relativa a far pointers e falta de documentos relativos ao UART, apesar de possuir links, estes links estavam inativos.
- A ideia de ter um monitor, ainda estudante, nas aulas laboratoriais foi uma ajuda enorme uma vez que o professor não consegue fisicamente atender a todas as duvidas dos alunos, ainda que a duração das aulas seja de 3 horas.
- A data limite de entrega para os laboratórios deveria, na nossa opinião, ser revista uma vez que é injusto para os alunos que têm as aulas no inicio da semana.

6. Instruções de instalação

Para instalar o jogo basta mover o ficheiro game que se encontra na pasta conf para a pasta etc/system.conf.d. (nota: para tal são necessárias permissões root). Após isto, para correr o jogo simplesmente fazer make na consola (dentro da pasta code) e executar o comando “service run `pwd`/game”.