



# MARIO-MAN

*POKEBALL EDITION*

Alberto José Ribeiro da Cunha(up201906325)  
Martim Raúl da Rocha Henriques(up202004421)  
Rúben Costa Viana(up202005108)  
Tiago Filipe Magalhães Barbosa(up202004926)

Licenciatura em Engenharia Informática e Computação  
Laboratório de Computadores  
Turma 7 - Grupo 1

10 de junho de 2022

# Índice

<b>User instructions .....</b>	<b>3</b>
Ecrã Principal .....	3
Ecrã de jogo .....	5
Ecrã de fim de jogo.....	8
<b>Project Status .....</b>	<b>10</b>
Timer .....	11
Teclado .....	11
Rato .....	12
Video Card .....	13
Real Time Clock (RTC) .....	13
<b>Code organization/structure .....</b>	<b>13</b>
Timer.c .....	13
Keyboard.c .....	14
Mouse.c.....	14
Videocard.c.....	14
Rtc.c .....	14
Sprites.c.....	15
Utils.c .....	15
Assets .....	15
Proj.c .....	15
Call Graph.....	16
<b>Implementation Details .....</b>	<b>17</b>
RTC.....	17
Colisões .....	17
Lançamento de pokebolas .....	17
<b>Conclusão .....</b>	<b>18</b>

# User instructions

## Ecrã Principal

Iniciando o projeto com o comando lcom\_run proj é nos levado para o menu inicial onde encontramos duas opções : "PLAY" e "EXIT". O utilizador pode com recurso ao rato mover o cursor e selecionar uma dessas opções.



- **PLAY** - Ao selecionar esta opção o utilizador é levado para o ecrã de jogo e pode começar a jogar.



- **EXIT** - Ao selecionar esta opção o utilizador pode sair do jogo e terminar o programa.



Ao invés de clicar no botão "EXIT" também é possível pressionar a tecla ESC para terminar o programa.

## Ecrã de jogo

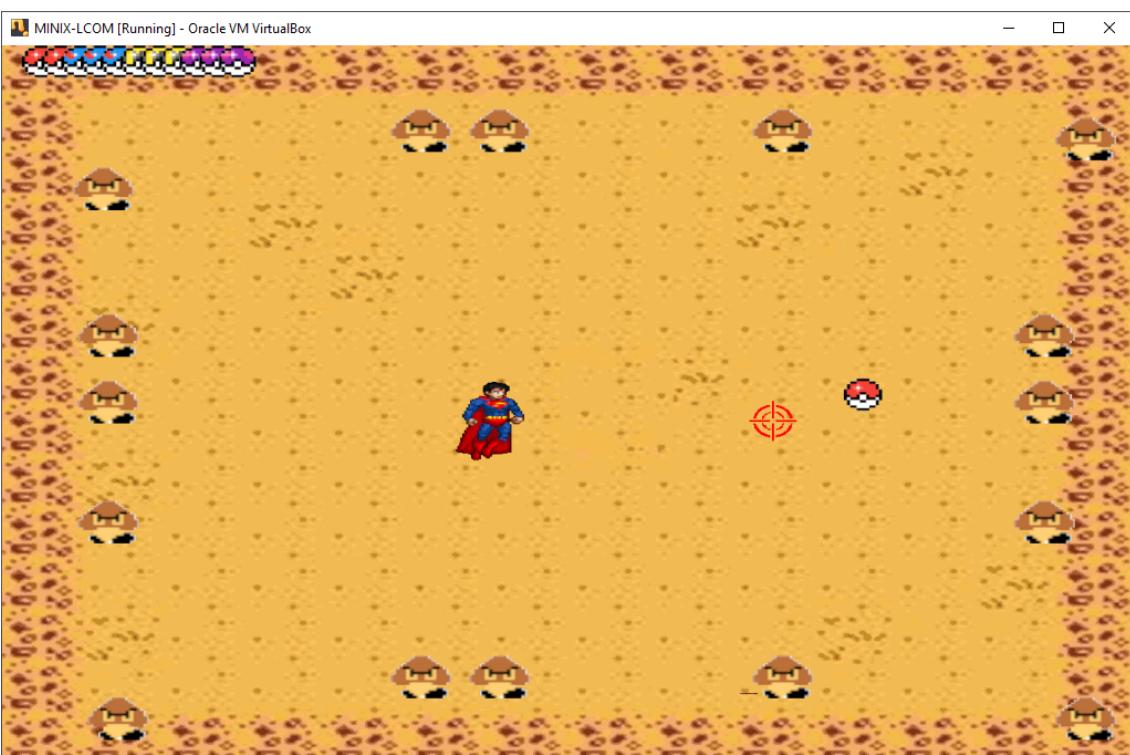
Após selecionada a opção de "PLAY" o jogo inicia com o jogador no centro do ecrã e com os adversários ("goombas") a nasceram em vários locais .



O objetivo do jogo é capturar o máximo número de goombas com pokebolas antes do alarme de 1 minuto ou antes que o jogador seja apanhado por um goomba. O jogador para se movimentar deverá utilizar as teclas AWSD:

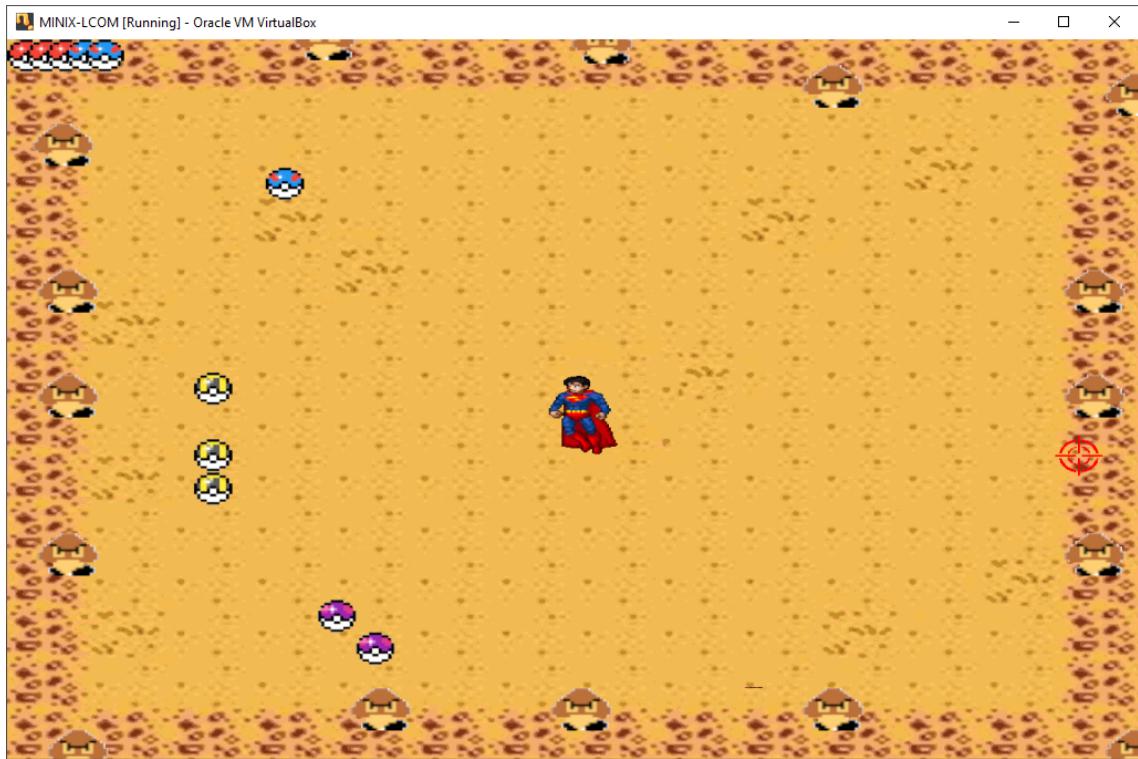
- A - mover-se para a esquerda
- W - mover-se para cima
- S - mover-se para baixo
- D - mover-se para a direita

Se estiverem pressionadas duas teclas ao mesmo tempo o jogador irá se mover nas duas direções , por exemplo , se tiver pressionado o A e o W o jogador irá se mover na diagonal para cima e para a esquerda. O jogador pode utilizar o rato para mover a mira e utilizar o botão principal do rato (mouse1) para atirar as pokebolas na direção da mira.





Cada pokebola é capaz de capturar um goomba e é possível disparar várias pokebolas segurando o botão principal do rato e movendo-o em várias direções.



Os goombas conseguem farejar o jogador e estão sempre a ir atrás deste a partir do momento que nascem. Quando um goomba é capturado o jogador ganha 1 ponto e o goomba renasce.

## Ecrã de fim de jogo

Após passar 1 minuto desde o início de jogo, quando um goomba consegue apanhar o jogador ou quando o jogador pressiona a tecla ESC durante o jogo, este termina e é apresentado um ecrã de "GAME OVER" apresentando os pontos que o jogador fez e duas opções : "PLAY AGAIN" e "EXIT"



1. **PLAY AGAIN** - Permite ao jogador começar um novo jogo.



2. EXIT - Permite ao jogador regressar ao menu.



O jogador ao invés de clicar no botão "EXIT" também pode pressionar a tecla ESC para regressar ao menu.

## Project Status

Dispositivo	Para quê	Int.
Timer	Controlar a frame rate	Sim
Teclado	Mover o jogador Terminar jogo mais cedo Termina programa no menu Regressar ao menu no fim de jogo	Sim
Rato	Mover o cursor do menu Selecionar opções no menu Mover a mira do jogo Lançar pokebolas no jogo	Sim
Video Card	Iniciar o modo de vídeo do programa Desenhar sprites no ecrã	N/A
Real Time Clock(RTC)	Obter a data atual de início de jogo Programar um alarme para 1 minuto após o início do jogo	Sim

Para além destes dispositivos, tínhamos como ideia inicial implementar a porta de série que permitia ligar duas máquinas virtuais possibilitando assim que um dos jogadores controlasse o super homem enquanto outro controlava os goombas que o tentam apanhar. No entanto, devido à falta de tempo não foi possível implementar esta funcionalidade.

## Timer

Foram utilizados os interrupts do timer para atualizar o ecrã de uma forma constante, desenhando todo o estado do jogo a cada interrupt. Assim permitia-nos mover objetos que não se moviam através de rato ou teclado e apenas se moviam constantemente com uma velocidade definida pela frequência destas interrupções. Também a cada interrupção do timer , como existiam objetos que se moviam, verificamos sempre se existiam colisões e atualizávamos o estado de cada objeto e o estado do jogo de acordo com isso.

Assim a cada interrupção do timer é chamada a função UpdateScreen() que utiliza um case para verificar o estado do jogo : "PLAY", "MENU", "GAME\_OVER" , etc.. e desenhava todos os sprites necessários recorrendo à função draw\_sprite\_proj() e realizava posteriormente a cópia desses sprites do buffer para a memória de vídeo usando a função double\_buffer(). Quando o programa está no jogo em si, as interrupções do timer movem todos os objetos necessários utilizando as funções movePlayer(), moveBullets() e moveGoombas() e verifica as colisões entre estes usando as funções checkPlayerColisions() e checkGoombaColisions(). É utilizada a função time\_int\_handler() para lidar com as interrupções aumentando um contador, no entanto, neste momento esse contador não é utilizado pois é utilizado o Real Time Clock (RTC).

## Teclado

As interrupções do teclado são usadas para verificar se alguma tecla foi pressionada e alterar a posição do jogador ou alterar o estado do programa (tecla ESC).

Assim a cada interrupção utilizamos a função kbc\_ih() para , no caso de existir uma tecla pressionada, obter o scancode da mesma. Depois com recurso à função UpdateStateKbd() alteramos o estado do programa no caso da tecla ESC ou alteramos a direção do jogador no caso das teclas A,W,S ou D tendo em consideração os make e breakcodes para saber se a tecla foi pressionada ou largada. Todos estes updates são realizados dependendo do estado do programa sendo usada uma implementação de uma máquina de estados.

## Rato

As interrupções do rato são usadas para construir os packets de um movimento do rato e quando este packet estiver completo atualizar o estado do cursor ou mira dependendo se estamos no menu ou no jogo e verificar se o botão principal do rato foi pressionado para selecionar uma opção no menu ou para atirar pokebolas.

É utilizada a cada interrupção a função kbc\_ih() que permite ler do rato os bytes necessários para a construção de um packet de movimento do rato e a função UpdateStateMouse() que verifica se o packet está construído e utiliza a função organize\_packets() para colocar a informação do packet numa struct que facilite a utilização destes dados . Depois de ter o packet totalmente construído e a informação de movimento ou de botões pressionados, a função UpdateStateMouse(), dependendo do estado do programa(máquina de estados) irá atualizar a posição do cursor no menu ou da mira no jogo e irá , se o botão principal tiver sido pressionado, mudar o estado do programa se uma opção tiver sido selecionada ou lançar uma pokebola na direção da mira durante o jogo. Esta função, quando selecionada a opção de play no menu , utiliza a função InitializeGame() para "spawnar" os goombas, as pokebolas e o jogador nas posições corretas e para ler a data atual e programar um alarme para 1 minuto depois acabar o jogo.

## Video Card

Foi utilizado modo 0x14C com resolução de 1152x864 com "Direct color model" e com 32 bits por pixel((8):8:8:8). Para iniciar o modo de vídeo é usado a função `vg_init()` e para voltar ao modo de texto é usado a função `vg_exit()`. Foi utilizado um método de "double buffering" onde para desenhar todos os sprites do estado atual do programa, estes são primeiros desenhados num buffer utilizando a função `draw_sprite_proj()` que utiliza a função `vg_draw_pixel()` para colocar cada pixel do sprite no buffer. Depois usa-se a função `double_buffer()` que copia o que estiver no buffer para a memória de vídeo(VRAM) evitando assim "tearing".

## Real Time Clock (RTC)

O real time clock é utilizado tanto para obter a data atual como para programar um alarme para 1 minuto após o início do jogo.

A cada interrupção do RTC é chamada a função `rtc_ih()` que lê o registo C do RTC usando a função `read_from_rtc()` para limpar as flags e dependendo da origem da interrupção(update ou alarm) alerta o programa para o disparo do alarme. O programa usa a função `rtc_get_date()` para obter a data atual e depois colocar os registos de alarme em conformidade com o pretendido.

## Code organization/structure

### Timer.c

Contém funções para subscrever e cancelar subscrição dos interrupts, algumas funções do lab2 que não são usadas no projeto(obter configuração, mudar frequência,etc..) e função de interrupt handler.

Desenvolvido na aula por todos os membros do grupo.

## Keyboard.c

Contém funções para subscrever e cancelar subscrição dos interrupts, algumas funções do lab3 que não são usadas no projeto(ler status, dar print a um scancode,etc..) e função de interrupt handler.

Desenvolvido na aula por todos os membros do grupo.

## Mouse.c

Contém funções para subscrever e cancelar subscrição dos interrupts, função de interrupt handler, função para organizar os packets, função para escrever argumentos para o rato,funções para ativar e desativar o mouse data e função para atualizar a posição do rato/mira de acordo com o packet lido.

Desenvolvido maioritariamente na aula por todos os membros do grupo excetuando a função de atualizar a posição do rato que foi realizada por Rúben Viana.

## Videocard.c

Contém funções para iniciar e sair do modo de vídeo, algumas funções do lab5 que não são usadas no projeto(desenhar linha, retângulo, pattern,draw\_sprite,etc..) ,função de desenhar um pixel para o buffer, função de copiar o buffer para a memória de video(double buffering) e função para desenhar sprites do projeto.

Desenvolvido na aula por todos os membros do grupo excetuando a função de desenhar sprites do projeto que foi realizado por Tiago Barbosa.

## Rtc.c

Contém funções para subscrever e cancelar subscrição dos interrupts, funções para ativar e desativar interrupts de "update" e "alarm" ,função de interrupt handler, função para ler de registos de RTC e função para obter a data atual.

Contém uma struct Date para representar uma data com os campos ano, mês, dia, hora, minuto e segundo.

Desenvolvido por Tiago Barbosa.

## Sprites.c

Contém funções para criar e destruir um sprite. Contém uma struct Sprite que representa um sprite do nosso projeto com posição, dimensões, velocidade e pixmap correspondente.

Desenvolvido por Tiago Barbosa, Ruben Viana e Martim Henriques.

## Utils.c

Contém funções para obter o MSB e LSB e uma função para ler dados de um port.

Desenvolvido na aula por todos os membros.

## Assets

A pasta assets contém todos os assets necessários para o projeto que foram criados todos por nós utilizando como ferramenta o GIMP para converter imagens em XPM.

Desenvolvido maioritariamente por Martim Henriques com alguma ajuda de Alberto Cunha.

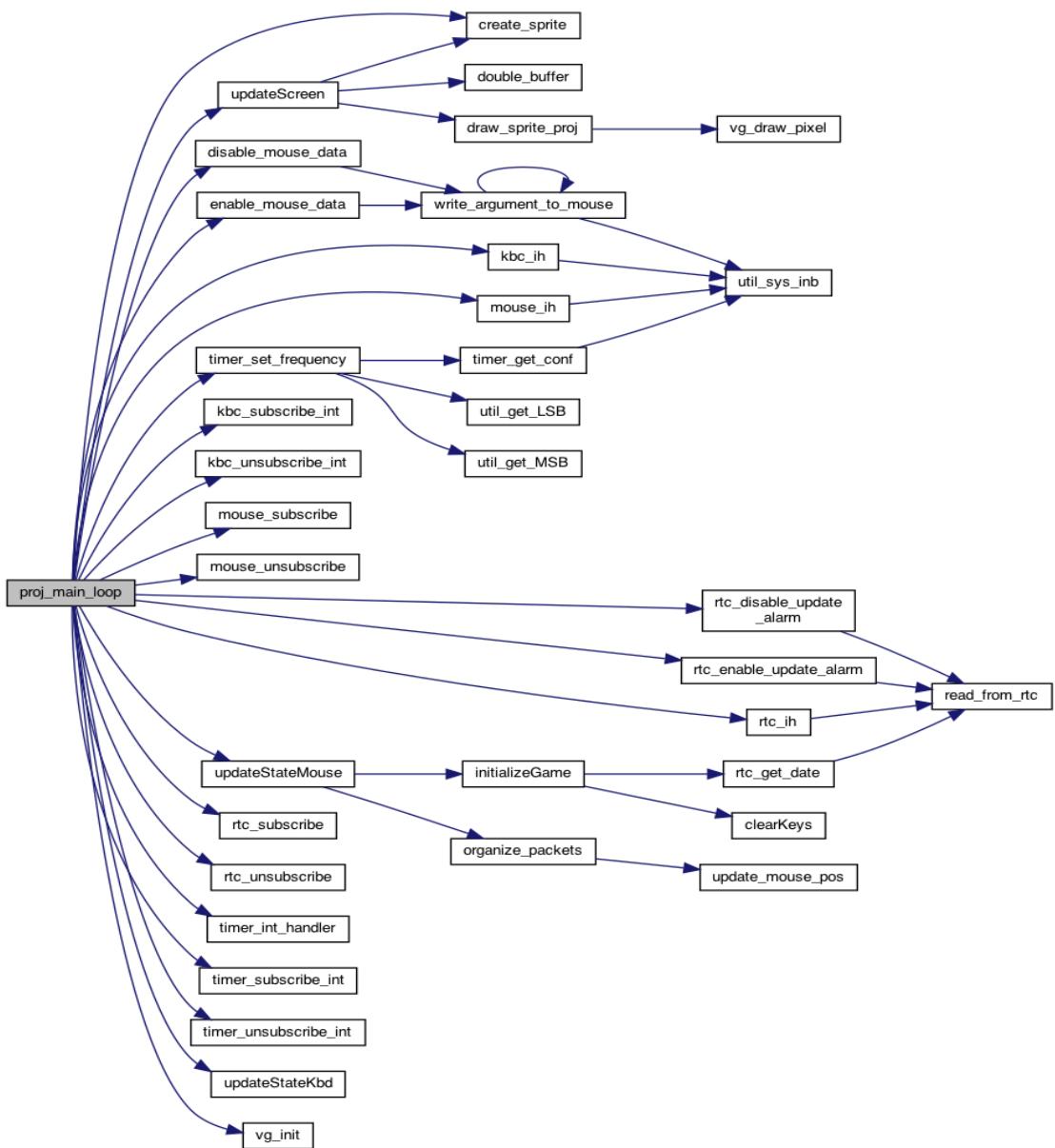
## Proj.c

Contém a função principal proj\_main\_loop() que trata de subscrever interrupções, iniciar modo de vídeo ,etc e contém o loop driver\_receive para tratar de interrupções. Contém também as várias funções já acima referidas que tendo em conta uma máquina de estados alteram o estado do programa de acordo com as várias interrupções.

Desenvolvido por Martim Henriques, Ruben Viana e Tiago Barbosa.

Módulo	Peso Relativo
Timer	8%
Keyboard	8%
Mouse	15%
VideoCard	15%
RTC	6%
Sprites	3%
Utils	2%
Assets	13%
Proj	30%

## Call Graph



# Implementation Details

## RTC

A implementação do RTC foi simples tendo sendo necessário a criação de uma função para ler registos de RTC para facilitar a obtenção da data. Para a criação do alarme foi necessário primeiro perceber que os registos estavam em hexadecimal o que demorou um tempo , mas quando percebemos apenas foi necessário acrescentar 1 minuto à data atual e colocar esses valores nos registos do alarme.

## Colisões

Para verificar as colisões entre dois objetos tivemos que entender um pouco como podíamos verificar se dois retângulos estavam sobrepostos visto que os sprites são todos retângulos. Utilizamos como referência:

[https://developer.mozilla.org/pt-BR/docs/Games/Techniques/2D\\_collision\\_detection](https://developer.mozilla.org/pt-BR/docs/Games/Techniques/2D_collision_detection)

Apenas tivemos que acrescentar um offset ao algoritmo presente nesse site para evitar que a colisão ocorresse quando os sprites ainda não se tivessem tocado visualmente, visto que, alguns sprites não preenchiam totalmente com pixeis de cores visíveis o seu próprio "retângulo".

## Lançamento de pokebolas

Para realizar o lançamento de uma pokebola na direção da mira quando o botão principal do rato for pressionado tivemos que pensar um pouco e optámos por após o click calcular a distância entre a mira e o jogador e colocar o atributo Xspeed e Yspeed da pokebola de acordo com a direção da mira. Para a velocidade ser constante, ao colocarmos o atributo Xspeed a ser a diferença de x entre jogador e mira dividimos essa diferença pela distância previamente calculada. Tivemos que fazer isto pois no início quanto mais longe a mira estivesse do jogador, mais depressa a pokebola viajava após ser lançada.

# Conclusão

No geral a ideia inicial do projeto foi atingida tendo ao longo da sua realização sendo acrescentadas algumas ideias como por exemplo os sprites usados e até a parte de existir um alarme para acabar o jogo. Também ficaram coisas pretendidas por realizar devido à falta de tempo como por exemplo a porta de série que iria permitir ao nosso jogo ser multijogador.

Fica para desenvolver no futuro, para além da possibilidade de multijogador, uma opção para mostrar os highscores do nosso jogo e uma opção que permita mudar de mapa e de dificuldade acrescentando mais goombas ou criando um alarme superior a 1 minuto.

Sentimos assim que no geral conseguimos atingir os objetivos da unidade curricular já que conseguimos implementar os vários dispositivos lecionados nas aulas e conseguimos integrá-los de uma forma eficiente para "trabalharem" todos juntos permitindo assim a criação do nosso jogo que utiliza variadas funcionalidades de cada dispositivo.