WORDLCOM

T05G03 LCOM – L.EIC

Trabalho realizado por:

André Lima – up202004260@edu.fe.up.pt

Guilherme Almeida – up202006137@edu.fe.up.pt

Jorge Sousa -up202006140@edu.fe.up.pt

Mariana Lobão – up202004260@edu.fe.up.pt

Índice

WORDLCOM

Wordlcom é o produto do nosso trabalho realizado na unidade curricular de LC. Desenvolvido em C e inspirado no jogo popular Wordle, recriamos no Minix tanto a interface gráfica como a jogabilidade do mesmo, utilizando o conhecimento adquirido ao longo do semestre.

O objetivo será encontrar a palavra alvo*, em inglês e com 5 letras, num máximo de 7 tentativas**. A cada tentativa, após carregar no "Enter", o jogo irá validar a mesma e apresentar, para cada letra, um dos três cenários:

I. Fundo **cinzento** – A letra não está presente na palavra alvo;
II. Fundo **amarelo** – A letra está presente na palavra alvo, mas numa posição diferente;
III. Fundo **verde** – A letra está presente na palavra alvo, no mesmo local onde aparece na tentativa.

*A *wordlist* pode ser alterada através de um argumento, na execução do programa, sendo possível alterar a linguagem e o número das palavras no jogo.

**O número de linhas e colunas do tabuleiro podem ser alteradas a partir do código.

FUNCIONALIDADES IMPLEMENTADAS

Dispositivo	Funcionalidade	Interrupt(I) / Polling(P)
Timer	Contabilizar o tempo de jogo e temporizar eventos. Desenhar o ecrã a 20fps.	I
Keyboard	Permitir ao utilizador jogar letras no tabuleiro e alterar o estado do jogo (“game” ou “menu”)	I
Mouse	Permitir ao utilizador selecionar o quadrado onde colocar uma letra e selecionar opções do menu	I
Video Card	Mostrar a UI e os gráficos relacionados com o jogo, para o utilizador conseguir jogar de uma forma confortável e user-friendly	N/A
Serial Port	Permitir jogos *multiplayer*, comunicando entre dois computadores	I

TIMER

Utilizado para contabilizar o tempo de jogo e para temporizar eventos.

A função *timer_init()* permite-nos inicializar o timer de modo a ser utilizado no jogo, estipulando a sua frequência, subscrevendo interrupções e criando eventos com base nas mesmas, que serão tratadas por um *event_handler*.

A função *timer_cleanup()* trata de desubscrever as interrupções, retornar à frequência original e dar *unregister* dos eventos relacionados com o timer.

KEYBOARD

Utilizado para permitir que o utilizador escreva letras no tabuleiro, preenchendo cada tentativa.

A função *keyboard_init()* inicializa o teclado para ser utilizado devidamente durante o jogo, subscrevendo as interrupções e criando eventos a ser tratados por um *event_handler*.

A função *keyboard_cleanup()* desubscreve as interrupções e dá *unregister* dos eventos relacionados com o teclado.

A função *kbd_map_code_to_char()* recebe um *makecode* e transforma-o num caracter legível, através de pesquisa linear numa tabela com as traduções *makecode-character* necessárias neste projeto.

A função *keyboard_handle_byte()* recebe um byte to *makecode* e chama a função *kbd_map_code_to_char()*. Esta mesma função distingue se o *makecode* tem um ou dois bytes e opera de acordo com essa distinção.

MOUSE

Utilizado para permitir o utilizador mexer o rato e clicar em opções do menu e nos quadrados específicos do tabuleiro.

A função *mouse_init()* recebe dois pares de bytes referentes à largura e altura do ecrã de modo a poder inicializar o rato de forma correta. Tal como nas outras funções, subscreve interrupções do periférico e cria eventos relacionados com o mesmo. Chama a função *mouse_enable_stream_mode()*, que permite ao rato enviar interrupções contínuas, e a função *mouse_enable_date_reporting()*, que permite receber informações acerca dos movimentos e cliques do rato. Ambas as funções foram criadas pelo grupo, não tendo sido utilizadas as funções da *lcf*.

A função *mouse_cleanup()* trata de desubscrever as interrupções do rato e dar *unregister* aos eventos do mesmo, revertendo o rato no estado inicial.

As interrupções do rato são trabalhadas pela função *mouse_handle_interrupt()*, que verifica qualquer erro de paridade e dá *parse* dos bytes obtidos pelo rato. A posição relativa do mesmo, bytes 1 e 2 do pacote, é utilizada para posteriormente desenhar o

rato no ecrã. O byte 0, que contém informação sobre cliques, é utilizado para operações que envolvam cliques por parte do utilizados, através de *handling* de eventos.

VBE

Utilizada para permitir o desenho de gráficos relacionados com o jogo.

A função *vbe_init()* inicializa a placa gráfica de modo a ser utilizada, chamando as funções *vbe_set_mode()* e *vbe_map_mem()*, que colocam a VBE em modo 11A, com modo de cor 5:6:5 utilizando cores diretas e resolução 1280x1024, e mapeiam a memória virtual, respetivamente.

A função *vbe_cleanup()* termina o uso da VBE, retornando ao modo de texto e dando *free()* à memória utilizada.

Utilizamos a técnica de *double buffering* para evitarmos lentidão no processo de desenho dos elementos no ecrã. Com isto, conseguimos apenas desenhar, no nosso caso, 30 vezes por segundo, em vez de a cada interrupção de um periférico. O buffer secundário guarda informação temporária e, quando estiver pronto, chamamos a função *vbe_flush()*, que copia a informação para o buffer principal, via *memcpy()*, que é então desenhado.

As funções *vbe_fill(), vbe_draw_pixel(), vbe_draw_line(), vbe_draw_rectangle()* e *vbe_draw_xpm()*, permitem-nos, respetivamente, colorir o ecrã numa só cor, desenhar um pixel, uma linha, um retângulo e um *xpm* previamente colocado em memória.

Em relação aos *xpms* utilizados, foram gerados pelo grupo, com a resolução que achamos adequada para o propósito.

SERIAL PORT (NÃO ACABADO)

Utilizado para permitir a comunicação entre PC's permitindo jogabilidade *multiplayer*.

A função *serial_init()* inicializa a porta série de modo a ser utilizada devidamente, subscrevendo as interrupções da mesma, limitando o fluxo de bits e gerando um *template* da estrutura de bits que passa pela porta.

A função *serial_cleanup()* termina o uso da porta série, eliminando a *queue* utilizada no programa e dando *unregister* aos eventos relacionados com a mesma.

A função *serial_send_packet()* envia informação, byte a byte chamando a função *serial_send_byte()*, pela porta série.

É de notar que este periférico, apesar dos esforços realizados, não foi implementado na sua totalidade. O código será submetido, daí as referências às funções desenvolvidas, no entanto não será compilado nem executado.

ORGANIZAÇÃO DO CÓDIGO

- devices/kbc.c
    - Implementa operações de ler e escrever comandos para o KBC. Utilizado para trabalhar com o rato e o teclado.
    - Peso no trabalho: 7%
    - Trabalhado por: André Lima

- devices/keyboard.c
    - Implementa operações com o teclado. De um modo geral, lê inputs do utilizador via teclado e faz a conversão para caracteres legíveis. Inclui *interrupt handlers*.
    - Peso no trabalho: 7%
    - Trabalhado por: Jorge Sousa

- devices/mouse.c
    - Implementa operações com o rato. Converte bytes emitidos pelo periférico aquando de ações do utilizador e converte-os para eventos emitidos para o programa. Inclui *interrupt handlers*.
    - Peso no trabalho: 7%
    - Trabalhado por: Guilherme Almeida

- devices/serial.c
    - Implementa operações relacionadas com a porta série. Permite leitura e envio de dados através da porta série para comunicar entre dois computadores. Inclui *interrupt handlers*. (não executado)
    - Peso no trabalho: 7%
    - Trabalhado por: Jorge Sousa e André Lima

- devices/timer.c
    - Implementa operações com o *timer*. Permite alterar a frequência do *timer*. Inclui *interrupt handlers*.
    - Peso no trabalho: 7%
    - Trabalhado por: Guilherme Almeida

- devices/vbe.c
    - Implementa operações com a placa gráfica, como desenho de elementos no ecrã. Inclui *interrupt handlers*.
    - Peso no trabalho: 7%
    - Trabalhado por: Mariana Lobão

- events/events.c
    - Permite criar, eliminar e dar *dispatch* a eventos.
    - Peso no trabalho: 6%
    - Trabalhado por: André Lima

- game/state.c
    - Permite alterar o estado do jogo.
    - Peso no trabalho: 6%
    - Trabalhado por: Guilherme Almeida

- game/wordlist.c
    - Permite ler a *wordlist* de um ficheiro e selecionar uma palavra aleatoriamente.
    - Peso no trabalho: 4%
    - Trabalhado por: André Lima e Mariana Lobão

- game/game.c
    - Permite gerir o jogo, validar ações do jogador e desenhar elementos no ecrã. Dividido em *model*, *controller* e *view*.
    - Peso no trabalho: 6%
    - Trabalhado por: André Lima e Mariana Lobão

- game/local_player.c
    - Permite gerir o estado do jogador e guardar as ações do mesmo. Dividido em *model*, *controller* e *view*.
    - Peso no trabalho: 6%
    - Trabalhado por: André Lima e Guilherme Almeida

- graphics/graphics.c
    - Permite desenhar todos os elementos do jogo no ecrã.
    - Peso no trabalho: 6%
    - Trabalhado por: André Lima e Mariana Lobão

- queue/queue.c
    - Implementa a estrutura de dados “fila”, para ser utilizada na porta série.
    - Peso no trabalho: 6%
    - Trabalhado por: Guilherme Almeida

- scheduler/scheduler.c
    - Permite executar tarefas espontâneas ou recorrentes após um determinado intervalo de tempo.
    - Peso no trabalho: 6%
    - Trabalhado por: Jorge Sousa

- vector/vector.c
    - Implementa a estrutura de dados “vetor”, para ser utilizado ao longo do projeto.
    - Peso no trabalho: 6%
    - Trabalhado por: André Lima

- protocol/protocol.c
    - Possibilita o modo de jogo *multiplayer* entre duas máquinas virtuais.
    - Peso no trabalho: 6%
    - Trabalhado por: André Lima e Jorge Sousa

DETALHES DA IMPLEMENTAÇÃO

Utilizamos a função *driver_receive()* apenas no ficheiro *main.c*, uma vez que este irá "apanhar" todas as interrupções e traduzi-las para os eventos correspondentes, colocando-os num vetor de eventos a ser tratado por um *event handler.* Utilizamos, portanto, uma estrutura *event driven*.

CONCLUSÕES

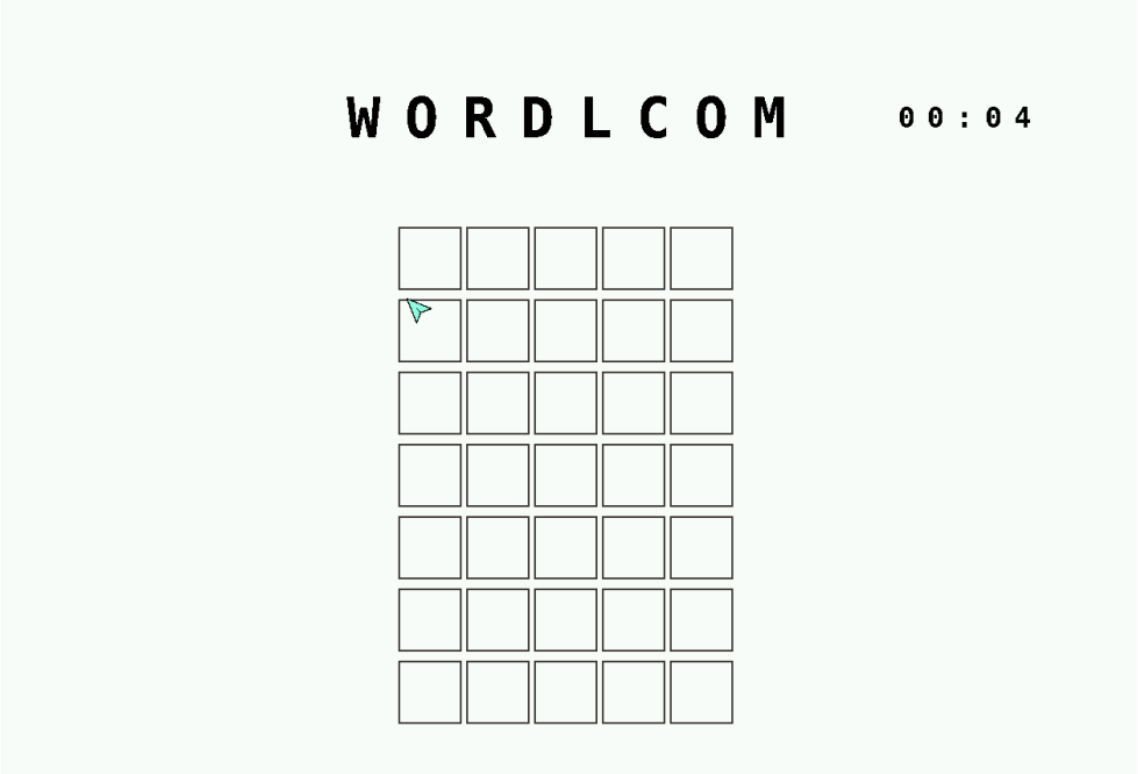
Não conseguimos implementar a porta série na sua totalidade no tempo estipulado para o desenvolvimento do projeto, apesar dos esforços. O código será entregue na submissão, apesar de não ser compilado na execução.

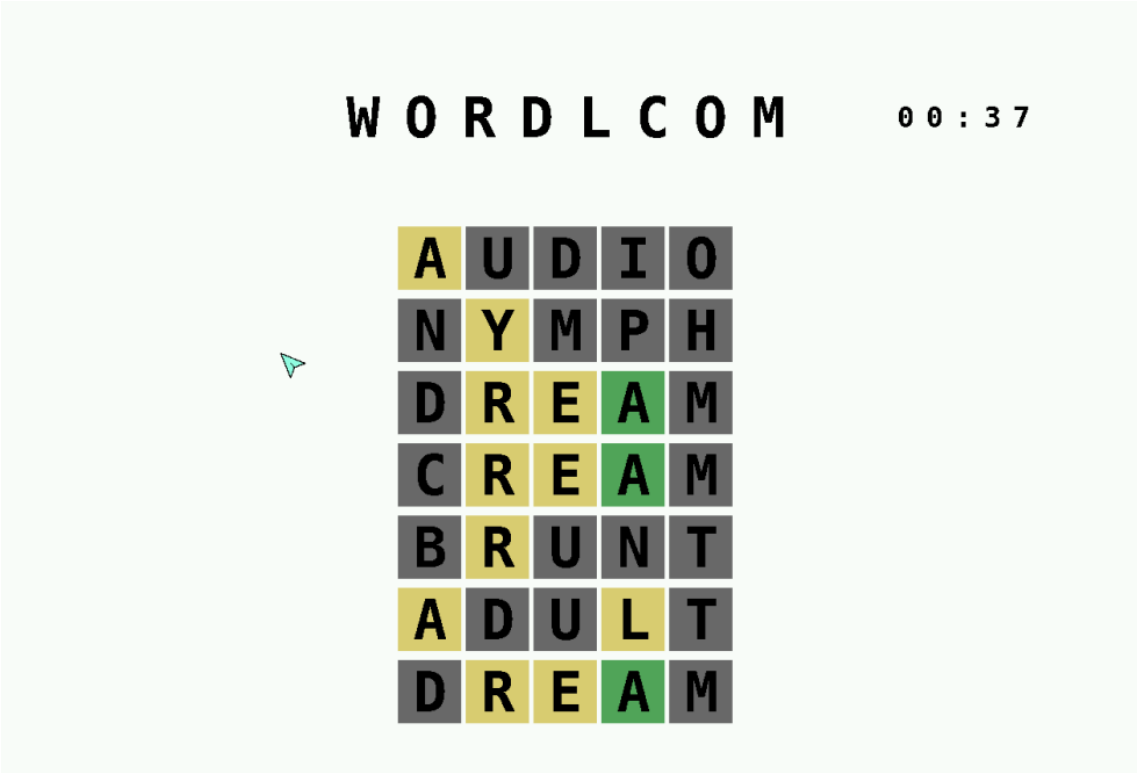
Não foi estipulado no planeamento, no entanto, no futuro, gostaríamos de apresentar no ecrã a lista de todas as letras que ainda não teriam sido utilizadas pelo jogador durante as suas tentativas, em que, aquando da sua jogada, iriam desaparecendo letras dessa mesma lista.

INSTRUÇÕES DE INSTALAÇÃO

Para o código executar de forma fluída, devemos utilizar o comando *make optimized.*

IMAGENS DE EXECUÇÃO




WORDLCOM
00:37
AUDIO
NYMPH
DREAM
CREAM
BRUNT
ADULT
DREAM