U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Laboratório de Computadores

EIC0020
Ano Letivo 2014/2015

T3G03
Diogo Marques (up201305642@fe.up.pt)
Pedro Miguel Melo (up201305618@fe.up.pt)

"ARKANIX"

ÍNDICE

DESCRIÇÃO

Jogo para um ou dois jogadores, cujo objetivo é eliminar blocos utilizando uma bola e uma barra controlada pelo jogador. Um jogador pode escolher controlar a barra com o rato ou o teclado.

No modo si*ngleplayer*, o jogador terá que ultrapassar três níveis, tendo um número máximo de três tentativas (vidas). Um jogador ultrapassa um nível eliminando todos os blocos existentes no ecrã. Caso o jogador fique sem tentativas ou complete o último nível, é mostrada a mensagem de "*Game Over*" ou "*Congratulations*!", respetivamente, podendo ou não entrar na tabela de *highscores* (com as melhores pontuações obtidas). O jogador perde uma tentativa sempre que a bloca tocar no chão.

No modo *multiplayer*, vence o jogador que conseguir eliminar mais blocos num determinado intervalo de tempo (modo contra-relógio). O jogo termina também quando um dos jogadores consegue limpar todos os blocos existentes no seu campo de jogo antes do tempo previsto. A particularidade deste modo é o facto de ser jogado em *split screen*, em que cada jogador pode ver o seu jogo e as jogadas do adversário lado a lado no próprio ecrã, bem como a pontuação actual de cada um .

DISPOSITIVOS

- ➔ Timer - responsável por controlar a placa gráfica, verificar a ocorrência de eventos, processar eventos aleatórios durante o jogo (power-ups); esta implementação não exige nenhuma alteração na configuração do timer, pelo que o jogo recorrerá aos interrupts do Timer 0 na sua velocidade por defeito de 60 interrupções/segundo (igual à taxa de refrescamento do ecrã, 60Hz).

  Método utilizado: interrupções

- ➔ Placa gráfica - para mostrar o ambiente do jogo, animações e texto. Serão utilizadas três camadas (*layers*) de forma a evitar perdas de *performance* ao desenhar novamente todo o ambiente de jogo: uma camada de fundo (para os menus e imagens de fundo do cenário), uma camada para o cenário (blocos, pontuação e vidas), e uma camada activa (que é atualizada constantemente em cada interrupção do *timer* para desenhar a barra e a bola quando necessário). O objetivo é correr o jogo à taxa máxima de 60 frames por segundo para conseguir um movimento fluído da bola e do rato.

  Método utilizado: *double buffering* e *page flipping.*

- ➔ Teclado - para controlar a barra, reconhecer atalhos de teclado frequentes (aceder ao menu pausa, sair do jogo) e inserir nomes na tabela de *highscores*.

  Método utilizado: interrupções**.**

- ➔ Rato - para controlar a barra e para navegar nos menus. Será implementado um cursor para facilitar a navegação, cujo código será partilhado com o da barra.

  Método utilizado: interrupções.

➔ Porta série - para permitir o modo *multiplayer em split screen;* devido às limitações deste periférico, não será possível enviar em tempo real o ecrã do adversário sob forma de imagem; a única informação a ser transmitida entre os dois computadores será as estruturas de dados do jogo (Sprite, Block); em vez disso, o computador do jogador enviará informações sobre a posição da barra, da bola e dos blocos restantes para o adversário; o computador do adversário (recetor) será responsável por desenhar no ecrã os sprites do jogador de acordo com as informações (posição, número) que recebeu pela porta de série.

Método utilizado: polling.

➔ RTC - para contar o tempo (no modo multiplayer, contra-relógio) e definir um alarme que é ativado e que gera uma interrução quando o tempo do jogo acabar; registar data no preenchimento na tabela de *highscores;*

Método utilizado: interrupções e polling.

VERSÕES

Versão 1:

- ❏ menus (imagem de fundo, cursor do rato, máquina de estados)
- ❏ teclado (controlo da barra, atalhos de teclado)
- ❏ rato (controlo da barra, navegação nos menus)
- ❏ nível de teste (não precisa de ser jogável, deve apenas mostrar os blocos e ser possível controlar a barra)

Versão 2 (demonstração na aula):

- ❏ lógica do jogo (colisões, movimento da bola, pontuação)
- ❏ níveis restantes, desta vez devem ser jogáveis
- ❏ ocorrência de eventos aleatórios (power-ups)
- ❏ *highscores* (mostrar pontuações, inserir nome, ler e escrever num ficheiro de texto)

Versão 3:

- ❏ implementar porta série
- ❏ implementar modo *multiplayer* em *splitscreen*

ESTRUTURA DE FICHEIROS

```
|-- src
    |-- arkanix.c
    |-- arkanix.h
    |-- block.c
    |-- block.h
    |-- bmp.c
    |-- bmp.h
    |-- kbd.c
    |-- kbd.h
    |-- mouse.c
    |-- mouse.h
    |-- rtc.c
    |-- rtc.h
    |-- serial.c
    |-- serial.h
    |-- level.c
    |-- level.h
    |-- timer.c
    \-- timer.h
```

MÓDULOS

Lógica - Diogo Marques / Pedro Miguel Melo

arkanix.c, **arkanix.h** - contêm a função main() , a lógica do jogo (dos dois modos, a implementação e funções de inicialização e destruíção da struct *ArkanixState,* que guarda o estado do jogo.

Blocos - Pedro Miguel Melo

block.c, **block**.h - contêm a struct Block e as funções com ela relacionadas, assim como várias instâncias pré-definidas desta, que representam os blocos a eliminar pelo jogador em cada nível. Os blocos apresentam características diferentes, conferindo um grau de dificuldade extra ao jogo.

Níveis - Pedro Miguel Melo

level.c, **level**.h - contêm os algoritmos para gerar os três niveis do modo *singleplayer* e o único nivel do modo *multiplayer*. Existe também uma *struct Level* que guarda um array de blocos e um bitmap com a imagem de fundo do nível.

Placa de vídeo - Pedro Miguel Melo

bmp.c, **bmp**.h - contêm funções necessárias para a leitura de ficheiros BMP. Destaca-se a função *read*, que devolve uma matriz de píxeis a partir de um ficheiro de entrada, cujo nome é passado como argumento, e a função *draw*, que desenha num *layer* a imagem lida com a função *read*. Escolheu-se o formato 24 bpp (R8 G8 B8), por ser um formato mais intuitivo e compativel. A única desvantagem do seu uso está no espaço que é necessário alocar na memória para guardar os bitmaps do jogo.

video.c, **video**.h - contêm as funções necessárias para inicializar o modo de vídeo, desenhar bitmaps, fontes e blocos.

Rato - Diogo Marques

mouse.c, **mouse**.h - contêm todas as funções necessárias ao correto funcionamento do rato e a implementação da struct *Mouse* que guarda o estado do cursor (coordenadas x e y da posição, o bitmap do cursor, o tamanho do cursor, uma flag *updated* que indica se o rato recebeu um interrupt desde a última vez que foi atualizado, e o estado dos três botões.

Teclado - Diogo Marques

keyboard.c, **keyboard**.h - contêm todas as funções necessárias ao correto funcionamento do teclado (leitura de scancodes e movimento da barra).

Timer - Diogo Marques

timer.c, **timer**.h - contêm todas as funções necessárias ao correto funcionamento do timer e a implementação das funções struct *Timer*, bem como o registo do número de interrupções geradas até ao momento.