# Implementação do jogo Minesweeper usando a linguagem Handel-C

Leonel Neves

*Resumo* – **Pretendeu-se criar uma versão do famoso jogo do Microsoft Windows® "Minesweeper" para a placa RC100 da Celoxica, suportado por um ambiente gráfico cromaticamente rico (16bpp), organizado em hierarquia de janelas e *event driven*, à semelhança dos actuais sistemas operativos.**

**O programa começa por copiar os bitmaps da memória Flash para o bloco de SSRAM que é usado para dados, seguindo-se a inicialização das estruturas de dados do jogo. A partir desse momento, vão correr em paralelo os processos: *video driver*, *mouse driver*, escrita no ecrã e leitura do rato, gerador de números aleatórios, contadores de tempo, *window manager* e o jogo propriamente dito.**

**Optou-se por usar um dos blocos de SSRAM para conter os *pixels* que são continuamente alimentados à saída VGA. São lidos dois *pixels* de cada vez num ciclo de relógio, o que liberta o ciclo de relógio seguinte para que se possa escrever nova informação nessa RAM, se necessário.**

**O ponteiro do rato é definido numa ROM constituída por um *array* de valores indexados a um *colormap* de cores a 24bpp. Por motivos estéticos, resolveu fornecer-se sombra ao ponteiro, a qual está definida numa ROM adicional.**

**Poder-se-á, sem grande dificuldade de adaptação, utilizar este ambiente para correr qualquer programa para o referido *hardware*.**

*Abstract* – **This article describes an implementation of the well-known Microsoft Windows® game "Minesweeper", on the basis of the Celoxica RC100 board. The game runs in a rich and colorful environment (16bpp), which is event driven and organized as a window hierarchy, just like modern operating systems.**

**At the beginning, the program copies bitmaps previously stored in the Flash RAM to the SSRAM block used as a databank. Then, the game structures are initialized. After that, several processes run in parallel: the video driver, the mouse driver, output to screen, mouse processing, random numbers generator, time counters, window manager and the game itself.**

**One of the SSRAM blocks is used exclusively as video RAM, thus providing a continuous feed of pixels to the VGA output. In a single clock cycle two pixels are fetched, which frees the following clock cycle, making it possible to write new data into that RAM block.**

**The mouse pointer is defined as an array of indexes to a 24bpp colormap and is enhanced by a mild and pleasant shadow, also defined as an array. Both pointer and shadow are kept in ROMs.**

**This environment can be tailored to run any program for the RC100 hardware without to much effort.**

## 1. INTRODUÇÃO

Os utilizadores de sistemas digitais são cada vez mais exigentes relativamente à qualidade das interfaces Humano-Computador. Hoje em dia já é possível apreciar o elevado nível de complexidade do grafismo dos mais variados sistemas electrónicos, da indústria dos telefones celulares às consolas de jogos. Neste contexto, faz sentido pensar em soluções de visualização igualmente agradáveis destinadas ao uso de sistemas reconfiguráveis autónomos.

No decorrer do desenvolvimento deste trabalho, embora o produto final seja um simples jogo, pretendeu-se dar ênfase à capacidade de simular um ambiente gráfico de alto nível numa pequena placa electrónica. Para tal, foram exploradas as características específicas do *hardware*, como a capacidade de suportar processamento paralelo real, uma das vantagens dos sistemas que utilizam uma FPGA relativamente aos que usam um PIC (Processor Integrated Circuit, microcontrolador). Assim, de entre as linguagens disponíveis para especificar sistemas digitais, optou-se pelo Handel-C por ser de alto nível. O programa destina-se a correr na placa RC100 da Celoxica e foi desenvolvido com o *software* DK2 da Celoxica. No site [1] encontra-se informação detalhada sobre a linguagem, o *software* e a placa utilizados.

### *A. Objectivos*

O trabalho desenvolvido pretendeu, entre outros, responder aos seguintes desafios:

- disponibilizar um ambiente gráfico de alto nível para o jogo Minesweeper;
- construir uma plataforma adequada à solução pretendida e ao *hardware*;
- responder com sucesso à especificidade dos dispositivos periféricos rato e monitor VGA;
- integrar técnicas de optimização e de reutilização dos recursos da FPGA;
- utilizar eficientemente os blocos de SSRAM e a Flash RAM.

### *B. Descrição global do trabalho*

A versão do jogo Minesweeper aqui implementada consiste numa matriz de 16x16 células clicáveis, como se pode ver na figura 1. No início de um novo jogo, são colocadas automaticamente 40 minas ocultas na área virtual correspondente às células, recorrendo a um gerador

de números aleatórios. Para dar início ao jogo, clica-se numa célula qualquer, momento em que o tempo começa a contar. Garante-se ao jogador que a primeira célula descoberta nunca tem mina. Cada célula descoberta contém a indicação do número de minas nas células vizinhas ou, se contiver uma mina, o jogo termina e o jogador perde. Se a célula clicada estiver numa área sem minas, o processo de descoberta é automático, de forma recursiva, para toda essa área. O jogador assinala uma mina clicando com o botão direito do rato na célula suspeita. O jogador deve descobrir todas as minas antes de se atingir o tempo limite, 999 segundos. O jogo termina com sucesso quando todas as minas, e apenas essas, estão assinaladas correctamente. Um novo jogo pode ser iniciado a qualquer momento clicando no botão que contém o desenho de um smiley. A intervenção do utilizador é feita através de um rato directamente ligado à placa. A figura 2 ilustra a interligação das componentes de *hardware* utilizadas.

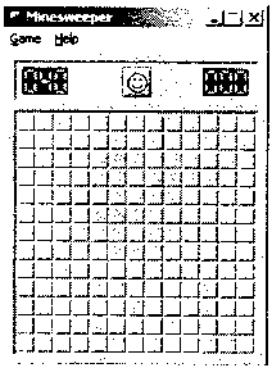


Fig. 1 – Minesweeper

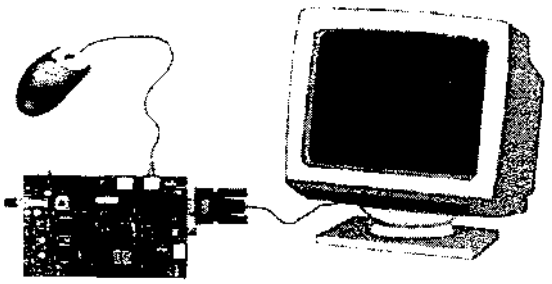

Fig. 2 – Interligação da placa RC100 e periféricos

O único dispositivo de interacção disponível ao utilizador é o rato, pois é esse que é usado quando se joga num computador.

## II. ORGANIZAÇÃO DO PROGRAMA

Este programa faz uso intensivo de bitmaps, os quais já devem estar na memória Flash da placa (a partir do endereço 0) antes de sua execução. No início, o programa copia os bitmaps da Flash para o bloco 1 da SSRAM. De seguida, inicializa as estruturas de dados do jogo. É então que arrancam em paralelo os processos:

- video driver;
- mouse driver;
- escrita no ecrã e leitura do rato;
- gerador de números aleatórios;
- contadores de tempo;
- window manager e o jogo propriamente dito.

Todos estes processos são, basicamente, ciclos infinitos independentes. A comunicação entre eles, como se verá mais adiante, efectua-se ou através da RAM ou através de variáveis globais. Esta é a parte activa da função *main*:

```
copyFlash2RAM();
setupNewGame();
par { // execução em paralelo:
  RC100VideoDriver(&Video);
  RC100PS2MouseDriver(&Mouse, RC100_MOUSE_PORT);
  runScreenAndMouse();
  runRandomGen();
  runTimer();
  runWindowManagerAndGame();
}
```

## III. MODELAÇÃO E ESTRATÉGIAS ADOPTADAS

### *A. Suporte para a interface gráfica*

Para se conseguir libertar o jogo da necessidade de estar sincronizado com o feixe de varrimento do ecrã, optou-se, logo no início do projecto, por utilizar o bloco 0 de SSRAM exclusivamente como RAM de vídeo. Para obter gráficos de elevada qualidade e ainda manter disponível uma elevada largura de banda para a escrita de pixels na RAM, para além da obtida nos períodos de *blanking* do feixe de varrimento, resolveu guardar-se em cada endereço, de 32 bits, a informação de 2 pixels, cada um com 16 bits distribuídos numa variante da forma RGB565. Na macro `runScreenAndMouse()`, sempre que o feixe de varrimento está nas colunas ímpares, é lido um endereço, tendo o cuidado de verificar situações excepcionais como o fim visível das linhas e o fim visível do ecrã. Os pixels correspondentes são armazenados num *buffer* (a variável `video_data`), o qual é consultado no momento de enviar um pixel para a saída VGA após ser convertido para RGB888. Por sua vez, quando invocada no decorrer do jogo, a função `writePixel(x, y, cor)` faz a escrita na RAM tendo em conta o processo de leitura. Assim, o pixel de coordenadas (X,Y) é mapeado no endereço N = X / 2 + Y x BG_WIDTH / 2, em que BG_WIDTH é o número de pixels por linha, ficando o pixel guardado na parte baixa ou alta em função do bit 0 de X, aquele que indica se a coluna é par ou ímpar (figura 3).

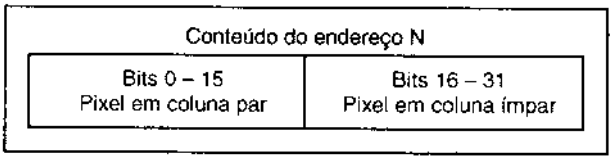


Fig. 3 – Conteúdo de um endereço da RAM de vídeo

Como não pode acontecer mais do que um acesso à RAM em cada ciclo de relógio, a macro `runScreenAndMouse()` controla a variável global `vram_ocupada`, a qual é consultada em `writePixel(...)` antes desta escrever o pixel. De notar que a escrita de um pixel consiste em ler o conteúdo de um endereço de memória, combinar os bits

lidos com os bits novos (simples substituição) e, por fim, escrever o resultado no mesmo endereço de memória.

O ponteiro do rato não é armazenado na RAM de vídeo. A sua apresentação no ecrã é conseguida fazendo a leitura de um pixel do *buffer* e combinando-o com um valor lido da matriz de pontos do cursor, definida estaticamente. De seguida afecta-se cada componente de cor do pixel com um coeficiente de transparência, cujo valor vem de uma matriz estática, o qual fornece ao ponteiro uma sombra visualmente agradável, 75% transparente. As coordenadas do ponteiro do rato e o estado dos botões são armazenados numa estrutura de dados (a variável global `mouse_data`) adequada às necessidades do programa. De notar que não é obrigatório o *hot spot* do cursor coincidir com o canto superior esquerdo da respectiva matriz, o que oferece diversas possibilidades criativas. A matriz contém índices para uma *look-up table* com as cores que o cursor pode apresentar, já em RGB888. Ambas as matrizes referidas são constituídas por simples vectores de valores.

### *B. Interface gráfica orientada aos eventos*

As interfaces gráficas *event driven* pressupõem a existência de uma estrutura computacional capaz de analisar o estado do sistema a intervalos mais ou menos regulares e identificar as diferenças entre dois estados consecutivos. Estas diferenças são os eventos. A figura 4 ilustra este processo, o qual é implementado no procedimento macro `runWindowManagerAndGame()`.

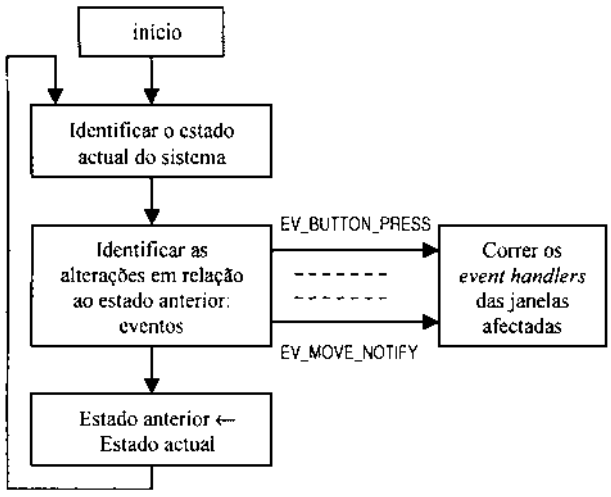


Fig. 4 – Mecanismo de base da gestão por eventos

Os eventos podem ser, por exemplo, movimentos do rato, botões pressionados e entrada do cursor numa janela. Por sua vez, uma janela é uma entidade conceptual que corresponde a uma área rectangular do ecrã e têm associada um conjunto de atributos, incluindo uma lista de sensibilidade a eventos. As janelas são organizadas numa hierarquia, a começar na janela de nível inferior, chamada de raiz, que cobre todo o ecrã e da qual todas as outras descendem. Ao nível da implementação, esta organização tem por base a estrutura `Window` que tem todos os campos necessários à execução do programa:

```
typedef struct {
  unsigned 3 id;        /* id da janela          */
  unsigned 3 idparent; /* id da janela pai      */
  unsigned 12 wx, wy;   /* posição rel. ao pai */
  unsigned 12 wx_abs, wy_abs;  /* pos. abs.    */
  unsigned 12 wwidth, wheight; /* dimensões    */
  void (*action[NUM_EVENTOS])(Event*);
                    /* lista de sensibilidade */
} Window;
```

A parte do programa que faz a gestão das janelas chama-se, apropriadamente, gestor de janelas (*window manager*). As estruturas `Event`, `Window` e `WManager` contêm a informação relevante neste contexto. Estes conceitos estão bastante desenvolvidos em "Xlib Programming Manual" [2], dos quais este trabalho apenas implementa um subconjunto muito pequeno. Normalmente os mecanismos de geração de eventos e de gestão de janelas são independentes, mas dada a necessidade de optimizar os recursos da FPGA, optou-se por juntá-los.

### *C. O jogo Minesweeper*

Uma vez definida a plataforma computacional sobre a qual corre o jogo e por observação do jogo original (figura 1) foram identificadas as áreas funcionais mapeáveis em janelas. A figura 5 contém a localização e a identificação das janelas:

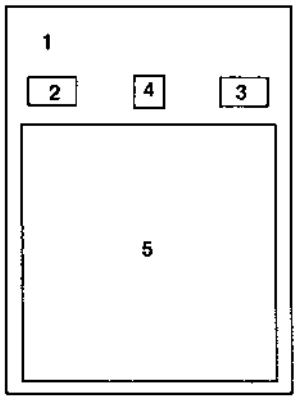


Fig. 5 – Áreas funcionais

1. janela do jogo;
2. contador de minas;
3. contador de tempo;
4. botão para reiniciar;
5. matriz de 16x16 células com minas escondidas.

Estas áreas correspondem à hierarquia que começa na janela raiz (índice 0 por definição), sobre a qual existe a janela do jogo (índice 1) e, sobre esta, as restantes. Os *event handlers* foram escritos de acordo com as funções a desempenhar no decorrer do jogo, de acordo com as acções do utilizador. Por exemplo, para o botão de início de jogo (janela índice 4) foram implementados:

```
static void smileyPressEV(Event *ev);
static void smileyReleaseEV(Event *ev);
static void smileyEnterEV(Event *ev);
static void smileyLeaveEV(Event *ev);
```

Por defeito, é invocada a função `nullFunc` quando uma determinada acção não está incluída na lista de sensibilidade da janela. Foram também escritos os procedimentos macro e as funções necessárias à implementação das regras do jogo descritas anteriormente. Por exemplo, quando é necessário escrever o valor de uma

célula na respectiva matriz, guardada na RAM de dados, invoca-se a seguinte função:

```
static void setCellValue(unsigned 4 i,
                         unsigned 4 j,
                         unsigned 4 val)
{
   RC100WriteSSRAM1(CELL_DATA_ADDR+adju(i,j,
                    AddressWidth),
                    adju(val, DataWidth));
}
```

Como se pode ver, teve-se o cuidado de não armazenar na FPGA informação que podia estar na RAM. A estrutura seguinte é a que contém os estado do jogo e é mantida na FPGA por ser pequena.

```
struct {
  unsigned 10 mcount, tcount;
   /* contadores: minas, tempo. */
  unsigned 4 mcount_digits[3];
   /* digitos vermelhos: minas. */
  unsigned 4 tcount_digits[3];
   /* digitos vermelhos: tempo. */
  unsigned 2 state;  /* estado do jogo.    */
  unsigned 3 smiley; /* índice do smiley. */
  unsigned 1 inside;
   /* cursor sobre as células e janela activa?*/
  unsigned 4 press_i, press_j;
   /* coordenadas da célula pressionada.      */
} game;
```

Durante o processo de limpeza automática recursiva de uma área de células sem minas, foi necessário implementar um stack em RAM, no qual são guardadas as coordenadas das células que precisam ser analisadas em cada ciclo. A variável `stack_p` foi inicializada com o último endereço da RAM de dados.

```
static void push(unsigned DataWidth val)
{
  RC100WriteSSRAM1(stack_p, val);
  stack_p--;
}
static void pop(unsigned DataWidth *val)
{
  stack_p++;
  RC100ReadSSRAM1(stack_p, *val);
}
```

Os bitmaps e o cursor foram obtidos a partir de ficheiros de imagens GIF e JPEG e convertidos para os formatos usados no programa usando um programa muito simples construído de propósito. Esse programa também gera as directivas `#define` com as características dos bitmaps. A sua descrição está fora do âmbito deste artigo mas é disponibilizado juntamente com este trabalho [3].

### *D. Estratégias gerais de optimização*

Para optimizar o programa, nomeadamente, minimizar a ocupação da FPGA, usou-se, sempre que possível, a SSRAM1. Não podendo usar aquela RAM, declararam-se as variáveis como sendo dos tipos RAM e ROM. De notar que a FPGA é, ao mesmo tempo, o recurso mais importante e o mais limitado da placa RC100.

Também foram colocadas em prática as principais recomendações da Celoxica. Por exemplo, foram utilizadas expressões Handel-C partilhadas por oposição às expressões macro; foi completamente eliminado o uso de divisões; substituíram-se todos os ciclos `for` por ciclos `while`; substituíram-se as expressões aritméticas complexas por outras mais simples a correr em paralelo; eliminaram-se os *combinational cycles*, ainda que o acesso à RAM de vídeo pareça ter um na função `writePixel`; todas as expressões condicionais foram transformadas em operações entre valores de 1 bit; não se utilizaram dispositivos e periféricos desnecessários, por exemplo, o teclado, sendo o rato suficiente. Adicionalmente, isolou-se o envio de informação para a saída VGA da geração dessa informação, o que eliminou a necessidade de manter a escrita de píxels sincronizada com o *hardware*. O procedimento macro `runScreenAndMouse` podia ser mais optimizado, pois inclui cadeias `if-else` muito longas.

## IV. CONCLUSÕES

Embora tendo como ponto de partida o simples exemplo "Mouse" incluído na documentação da ferramenta DK2, este trabalho constitui um exemplo bastante completo de como implementar um sistema gráfico de alto nível na placa RC100, usando a nosso favor as características do *hardware*. Neste sentido, os objectivos propostos foram integralmente cumpridos. Pela maneira que foi desenvolvido este programa, será possível adaptar o ambiente de execução a qualquer tarefa que necessite de interacção com o utilizador. Por fim, há que reconhecer que uma desvantagem deste sistema é a elevada ocupação da FPGA, cerca de 40% apenas com o ambiente gráfico. No entanto, o utilizador terá uma experiência idêntica ao uso de um moderno computador.

## AGRADECIMENTOS

Ao Prof. Valery Sklyarov (Universidade de Aveiro), na sua constante disponibilidade e valiosa orientação durante as aulas, deu um precioso estímulo para a publicação deste artigo.

## REFERÊNCIAS

[1] URL: http://www.celoxica.com/

[2] Adrian Nye, "Xlib Programming Manual", O'Reilly & Associates, Inc, 1989.

[3] URL: http://sweet.ua.pt/~leonel/work_aulas/cr_minesweeper/