U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Laboratório de Computadores - LCOM

“Paint - Minix Style”

Ferramenta de desenho colaborativo

Turma 2MIEIC5

João Neto - 201203873 - ei12013@fe.up.pt
Miguel Sandim - 201201770 - miguel.sandim@fe.up.pt
Prof. Pedro Ferreira do Souto

3 de Janeiro de 2014

Índice

1. Instruções de utilização do programa

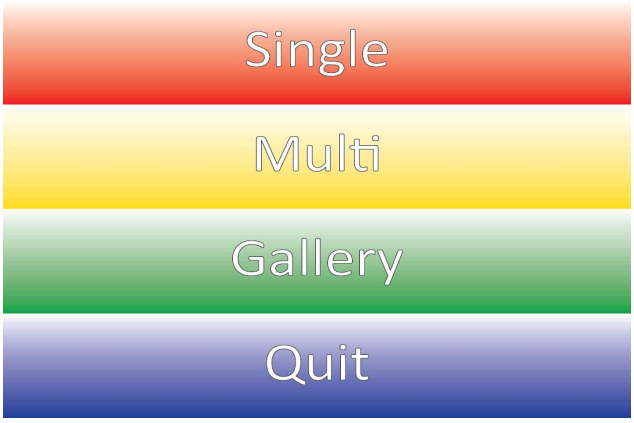
1.1. Ecrã inicial

Fig. 1 - Ecrã principal.

Ao iniciar o programa, é apresentado um ecrã de introdução do programa com informação sobre o projeto, os autores e o seu contexto. O utilizador pode avançar carregando em qualquer tecla.

1.2. Menu

De seguida, o utilizador é redirecionado para o menu principal onde poderá usar o rato para selecionar uma das 4 opções:



Single: inicia a ferramenta de desenho em modo de desenho singular;

Multi: inicia a ferramenta de desenho em modo de desenho colaborativo

Gallery: inicia o modo Galeria onde se podem ver os desenhos já gravados;

Quit: fecha o programa.

Fig. 2 - Menu.

1.3. Modo de desenho

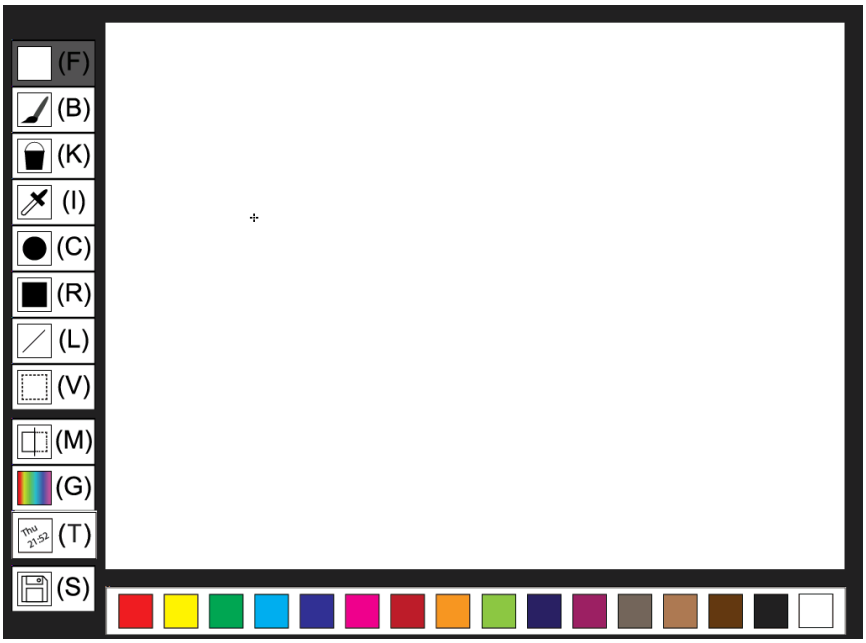


Fig. 3 - Modo de desenho.

Neste modo, o utilizador é convidado a desenhar livremente no ecrã, utilizando as ferramentas apresentadas. Junto ao ícone da ferramenta está, entre parêntesis, a tecla a utilizar para acionar a mesma ferramenta. Estão disponíveis as seguintes ferramentas:

1. "Blank" - Limpa o ecrã de desenho;

2. "Brush" - Permite pintar no ecrã uma linha utilizando o rato, com grossura alterável (pressionando as teclas '+' ou '-' no *numpad* ou junto à tecla *Enter*);

3. "Bucket" - Permite pintar o ecrã, de forma semelhante à ferramenta "Balde" do *paint* do *Windows*. O utilizador deverá selecionar no ecrã de desenho o local onde pintar;

4. Selecionador de cor - Ao clicar com o rato sobre uma cor (na tela de desenho), as ferramentas passam a trabalhar com essa cor. A cor de fundo do programa é alterada para a cor que está a ser usada;

5. Circulo - Desenha um círculo com dois cliques do rato, centro e extremidade;

6. Retângulo - Desenha um retângulo com dois cliques do rato, correspondentes a dois cantos opostos;

7. Linha - Desenha uma linha com dois cliques do rato, os pontos extremos do segmento de reta;

8. Selecionador de área - Permite selecionar uma área com dois cliques do rato, passando a área de desenho atual a estar restringida à selecionada pelo utilizador. O utilizador poderá selecionar outras ferramentas para aplicar nessa zona (de onde se excluem os efeitos, carimbo de hora e o “bucket”). Se o utilizador der um clique fora da zona de desenho selecionada, a área é anulada e o utilizador pode novamente desenhar em todo o ecrã (de notar que o botão fica ativo enquanto a nova área está funcional).

9. Efeito espelho - Coloca metade da imagem simétrica à outra metade, dando a ilusão de um espelho no meio da tela;

10. Efeito Magia - Altera o esquema de cores do desenho;

11. Carimbo de hora - Imprime a hora atual no formato HH - MM - SS DD - MM -YY no desenho;

12. Salvar - Guarda a imagem para poder ser posteriormente vista e editada.

Se o utilizador clicar na barra de cores, a cor selecionada é alterada para a cor que se encontra na localização do rato. A cor de fundo do programa é alterada para a cor que está a ser usada. Ao premir a tecla ESC o programa volta ao menu.

1.4. Modo galeria

Fig. 4 - Modo galeria com um desenho em visualização.

O modo galeria permite ao utilizador visualizar todos os desenhos que guardou e voltar a trabalhar neles.

Para navegar entre desenhos pode utilizar os botões laterais do ecrã ou com as setas do teclado (seta esquerda e seta direita).

Se pretender voltar a trabalhar num determinado ficheiro deve clicar no botão EDIT, que abrirá o modo de desenho singular com o ficheiro selecionado na tela. Se o utilizador gravar alterações neste desenho, estas substituirão o ficheiro do desenho antigo.

Ao clicar no botão BACK o utilizador retorna ao menu.

2. Arquitetura do programa

2.1. Estruturas de dados utilizadas

São utilizadas várias estruturas auxiliares no nosso programa, sendo estas:

➔ **SPRITE**

Esta estrutura destina-se a guardar elementos a serem pintados no ecrã, bem como as suas características.

Constituição:

- pixels (*array* de *unsigned shorts*): contém os pixeis de um objeto; seguindo o sistema de cores RGB com 16 bits (5-6-5);
- width (*unsigned int*): Comprimento, em pixeis, do objeto;
- height (*unsigned int*): Largura, em pixeis, do objeto.

➔ **BTN**:

Esta estrutura reconstitui um botão, com o estado premido e solto deste. É utilizada na barra de ferramentas lateral do modo de desenho, para permitir interação com as ferramentas.

Constituição:

- sprite_on (SPRITE): aparência do botão quando este está premido;
- sprite_off (SPRITE): aparência do botão quando este está solto;
- press_state (*int*): toma o valor de 1 ou 0, consoante a SPRITE a desenhar no ecrã seja a do botão premido ou não.

➔ **Draw_screen_area:**

Como o nome indica, esta estrutura designa uma área desenhável no ecrã. É utilizada no módulo de desenho, para limitar as interações do rato a uma área específica (ferramenta selecionador de área).

Constituição:

- h_dim (*unsigned int*): Dimensão horizontal da área;
- v_dim (*unsigned int*): Dimensão vertical da área;
- x_ul_corner (*unsigned int*): Coordenada X do canto superior esquerdo da área;
- y_ul_corner (*unsigned int*): Coordenada Y do canto superior esquerdo da área.

➔ **Stack:**

Implementação de uma pilha de *ints* (LIFO). Utilizada no algoritmo “Flood Fill” da ferramenta “Bucket”.

Constituição:

- buf (*array* de *ints*): *array* que simula a pilha;
- top (*int*): índice do elemento no topo da pilha;
- size (*int*): tamanho da pilha.

➔ **Date_info:**

Struct que contém informação sobre a hora e data num determinado instante. É constituída por 7 campos do tipo *unsigned long* sendo estes seconds (segundos), minutes (minutos), hours (horas), month (mês), year (ano), month_day (dia do mês) e week_day (dia da semana).

➔ **Header:**

Estrutura para manter a informação contida na primeira parte de um ficheiro BMP. Usada no carregamento de imagens no programa.

Constituição:

- type (*unsigned short int*): Identifica o formato BMP;
- size (*unsigned int*): tamanho do ficheiro em *bytes*;
- reserved1,reserved2 (*unsigned short int*): campos reservados;
- offset (*unsigned int*): distância à informação dos pixeis da imagem,em *bytes*.

➔ **Info Header:**

Estrutura para manter a informação contida na segunda parte de um ficheiro BMP. Usada no carregamento de imagens no programa.

Constituição:

- size (*unsigned int*): tamanho do header em bytes;
- width, height (*int*): largura e altura da imagem;
- planes (*unsigned short int*): número de planos de cor;
- bits (*unsigned short int*): número de bits por pixel;
- compression (*unsigned int*): tipo de compressão:
- imagesize (*unsigned int*): image size in bytes;
- xresolution, yresoultion (*int*): pixeis por metro;
- ncolours (*unsigned int*): número de cores;
- importantcolours (*unsigned int*): número de cores importantes.

2.2. Organização por módulos

O nosso programa encontra-se dividido em módulos que comunicam entre si de modo a garantir que o programa utilize de forma correta os periféricos e realize as funções desejadas.

O seguinte esquema sumariza estas relações:

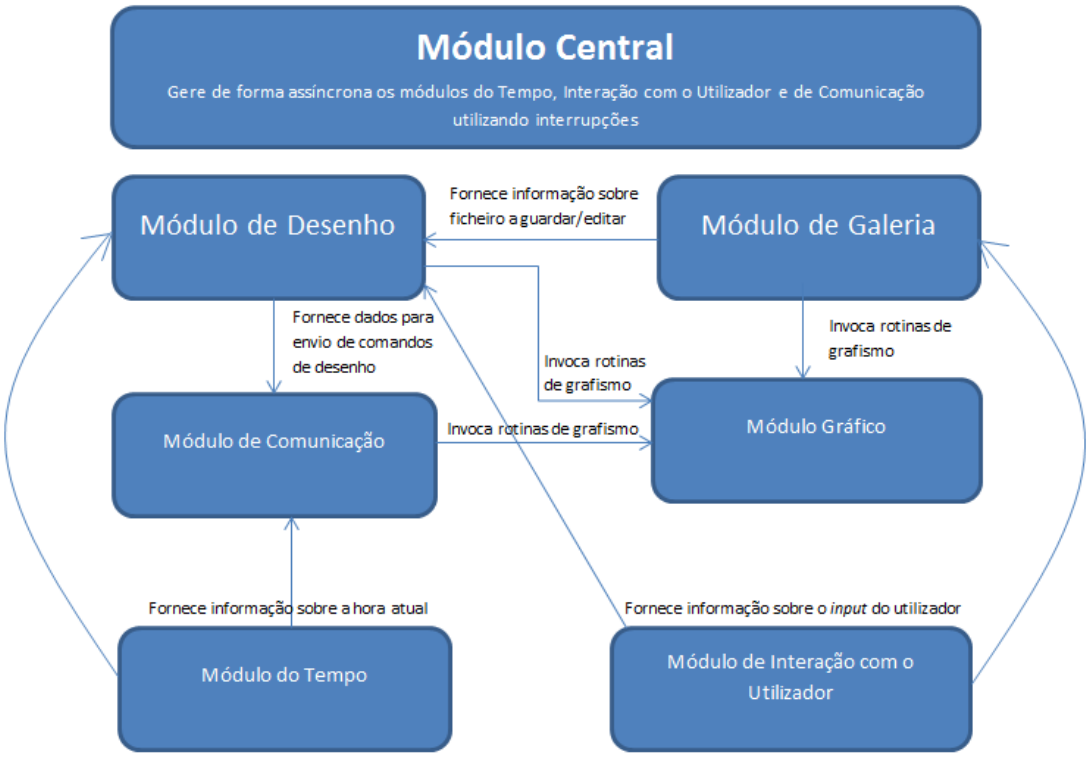


Fig. 5 - Esquema dos módulos e suas relações.

2.2.1. Módulo central

O módulo central (*central_module.h*) é o módulo principal do nosso programa, sendo responsável pela invocação de rotinas de alocação de memória para as diversas estruturas do programa. É também responsável pela gestão das interrupções, alterações de estado e eventos ocorridos no programa, assunto que é falado com maior precisão mais à frente.

2.2.2. Módulo de Desenho

O módulo de desenho (*draw_module.h*) é um dos principais módulos de alto nível na nossa ferramenta. É responsável pela gestão de eventos relativos à interação do utilizador durante o modo de desenho e utiliza-os na invocação das várias ferramentas e efeitos, utilizando máquinas de estados.

São utilizadas várias variáveis neste módulo nomeadamente:

➔ color_bar (SPRITE): estrutura com a barra de cores;

➔ button_array (*array* de estruturas do tipo BTN): lista de botões (ferramentas e efeitos) do modo de desenho;

➔ tool_current_state: variável que indica o estado de execução da ferramenta (usada na construção de círculos, retângulos e linhas, indicando se o utilizador está a fornecer o primeiro ou segundo ponto por exemplo);

➔ draw_screen (*array* de *unsigned shorts*): conjunto de pixeis que formam o desenho;

➔ tool_handlers (*array* de apontadores para funções): conjunto de *handlers* de desenho (ferramentas e efeitos);

➔ Variáveis inteiras tool_selected e color_selected: indicam respetivamente o índice da ferramenta em tool_handlers que está selecionada e a cor que está a ser utilizada;

➔ Flag serial_com_enabled: indica se o modo colaborativo está ativo e se o programa deve enviar comandos de desenho via porta série.

2.2.3. Módulo de Galeria

O módulo de galeria (*gallery_module.h*) é responsável pela gestão de eventos relativos à interação do utilizador quando este se encontra no “Gallery Mode” do programa. Também é responsável por colocar o conteúdo dos ficheiros de desenho em memória quando o utilizador navega no modo Galeria e mantêm em variáveis *static* o número de desenhos que existem e o número do próximo desenho a ser gravado.

As variáveis utilizadas neste módulo são:

➔ file_draw (buffer de *unsigned short*): estrutura que contém o desenho a apresentar no momento;

➔ total: número total de desenhos;

➔ file_number: número do próximo desenho a ser gravado.

2.2.4. Módulo de Comunicação

O módulo de comunicação (*com_module.h*) é responsável pelo envio e recepção de comandos de desenho. Para tal, são usadas máquinas de estado que transmitem a evolução da construção (recepção) de um determinado comando, sendo que se o comando chegar ao seu estado final este é processado e desenhado.

Durante a comunicação são usados FIFOs de 16 *bytes* em conjugação com *polling* através da COM1.

Este módulo usa a biblioteca de funções de baixo nível *serial.c* (funções que lidam com a porta de série).

As variáveis mais relevantes utilizadas neste módulo são:

- ➔ received_char: contém o último *byte* recebido;
- ➔ handlers (*array* de apontadores para funções): conjunto de *handlers* construtores de comandos;
- ➔ command_number: contém o índice do *array* handlers para o *handler* de construção de comando que está em curso;
- ➔ current_state: estado da construção do comando em curso.

2.2.5. Módulo Gráfico

O módulo gráfico (*graphic_module.h*) é responsável pelo desenho dos diferentes ecrãs nos diferentes estados do programa, e dos diferentes objetos (tais como a *colorbar*, a barra de ferramentas, o cursor do rato e a biblioteca gráfica de letras e números), implementando o conceito de *double buffering*

Este opera no modo 0x117 da VBE, trabalhando com pixeis de dois *bytes* (u*nsigned short*), com o sistema de cores RGB (5 bits para o *Red*, 6 bits para o *Green* e 5 bits para o *Blue*) e dimensões 1024 por 768 pixeis.

Cabe também a este módulo o *loading* dos diferentes elementos gráficos a partir dos *bitmaps* e a gestão de memória para os alocar (usando *malloc* e *free*). Estão também presentes neste módulo as funções de gravação e leitura de ficheiros de desenho.

Este módulo usa a biblioteca de funções gráficas de baixo nível *video_gr.h.*

As variáveis mais relevantes utilizadas neste módulo são:

- ➔ double_buf: *buffer* auxiliar usando para a visualização dos elementos gráficos. Na "escrita" de uma *frame* no *buffer*, todos os elementos gráficos são desenhados um a um neste *buffer* e copiados para a memória gráfica usando a função drawBufferInVRAM().

- ➔ letters: *array* de objetos do tipo SPRITE, que contém o alfabeto, algarismos de 0 a 9 e o caracter hífen.

- ➔ menu: SPRITE com o ecrã de menu;

- ➔ gallery: SPRITE com o ecrã do modo galeria.

2.2.6. Módulo de Interação com o Utilizador

O módulo de comunicação (*user_interaction.h*) é no fundo uma camada de mais alto nível que lida com o teclado e com o rato.

É basicamente constituído por duas funções que atualizam o estado do teclado (updateKeyboardStatus()) e o estado do rato (updateMouseStatus()). Estas duas funções são chamadas no Módulo Central quando existem interrupções relativas ao teclado e ao rato, e através dos seus *interrupt handlers* é obtida informação sobre o estado atual destes dois periféricos e colocada em variáveis internas do módulo.

São recolhidas as seguintes informações: que tecla foi premida/largada, e se esta tem o *makecode* ou *breakcode* com dois *bytes* (significando que é uma "seta", por exemplo) - no caso do teclado; a posição do rato no ecrã de acordo com o modo gráfico selecionado e suas dimensões e o estado (premido/não premido) dos três botões localizados no rato.

As variáveis mais relevantes utilizadas neste módulo são:

Teclado

- ➔ last_key_code: Contém o makecode da última tecla premida ou libertada;

- ➔ two_byte_flag: Flag que indica se o *makecode* na variável anterior é antecedido de 0xE0 (significando que tem um *makecode* de 2 *bytes*);

- ➔ press_key_flag: Indica se a tecla foi premida ou libertada.

Rato

- ➔ x_position: Indica a posição em x do cursor no ecrã, de acordo com o modo gráfico selecionado. É arbitrado que a coordenada horizontal cresce da esquerda

para a direita;

- ➔ y_position: Indica a posição em y do cursor no ecrã, de acordo com o modo gráfico selecionado. É arbitrado que a coordenada vertical cresce de cima para baixo;

- ➔ RB_pressed: Flag que indica se o botão direito do rato está premido ou não;

- ➔ MB_pressed: Flag que indica se o botão médio do rato está premido ou não;

- ➔ LB_pressed: Flag que indica se o botão esquerdo do rato está premido ou não;

Este módulo usa as bibliotecas de funções de baixo nível *mouse.h* (funções que lidam com o rato) e *keyboard.h* (funções que lidam com o teclado).

2.2.7. Módulo do Tempo

O módulo do tempo (*time_module.h*) é no fundo uma camada de mais alto nível que lida com o Real Time Clock (RTC).

Seguindo uma lógica análoga à usada na construção do Módulo de Interação com o Utilizador, este módulo contém uma função (updateRTCStatus()) que é chamada no Módulo Central quando existem interrupções relativas ao RTC, atualizando a hora na variável interna do tipo Date_info current_time, que pode ser posteriormente acedido a partir de uma função “get” pelos outros módulos.

Este módulo usa a biblioteca de funções de baixo nível *rtc.h*.

3. Detalhes de implementação

3.1. Utilização de interrupções

No contexto do que já tinha sido abordado nos *labs*, a maioria dos periféricos foram utilizados em modo *interrupt*, o que permite uma gestão assíncrona dos eventos dos diversos periféricos.

Apenas a porta de série foi utilizada em modo *polling*, sendo que para recepção de dados vindos da UART foram utilizadas as interrupções do *timer* (é feita uma tentativa de *update* a cada cinco interrupções se o utilizador estiver em modo colaborativo de desenho), o que contribuiu para a inclusão deste dispositivo na gestão simultânea dos periféricos.

3.2. Estados do programa e eventos

De acordo com o especificado, toda a arquitetura do programa foi baseada em estados, que podem ser alterados caso um evento dos periféricos ocorra (em especial, uma interação com o rato/teclado). Cada estado possui uma ou mais funções que gerem eventos vindos do *hardware*, que permitem associar a cliques e permissão de teclas a ações no programa.

Alguns destes *handlers* possuem ainda valor de retorno. Este valor de retorno permite a transição de estado de programa caso, por exemplo, o clique seja num botão ou seja pressionada uma tecla designada para sair ou entrar num outro ecrã.

Foi definido que se este valor de retorno fosse positivo, o programa avançava para um estado seguinte, e se este fosse negativo, avançava para o estado anterior.

Foram designados os seguintes estados e os seguintes *event handlers* na nossa aplicação:

- ➔ “Intro”: Estado inicial do programa, em que apenas se vê um ecrã de boas vindas.
  - keyboardIntroEvent(): Retorna 1 se for premida alguma tecla, o que possibilita avançar o estado do programa para “Menu”.

- ➔ “Menu”: Estado principal do nosso programa, em que o utilizador pode interagir com o rato, escolhendo ir para o modo de desenho (singular ou colaborativo), ir para o modo de galeria, ou sair.
  - mouseMenuEvent(): Retorna 1, 2, 3 ou -1 caso exista um clique

respetivamente no botão de desenho singular, colaborativo, modo galeria ou de sair.

- ➔ “Desenho Singular”: Estado em que o utilizador é convidado a desenhar em modo singular.
  - mouseDrawEvent(): Possibilita alterações internas dentro deste modo, não retorna nada.
  - keyboardDrawEvent(): Possibilita alteração de ferramenta a usar, retorna -1 se for pressionada na tecla ESC, o que muda o estado do programa para “Menu”.

- ➔ “Desenho Colaborativo (Multi)”: Estado semelhante ao anterior e com os mesmos *handlers*, mas que envia e recebe comandos de desenho via porta de série.

- ➔ “Galeria”: Estado em que o utilizador pode navegar entre os desenhos já criados e editar algum desejado.
  - keyboardGalleryEvent(): Possibilita mudar entre desenhos utilizando as teclas “seta direita” e “seta esquerda” do teclado, não retorna nada;
  - mouseGalleryEvent(): Retorna 1 caso exista um clique no botão “EDIT”, o que avança o programa para o estado “Desenho Singular” e retorna -1 caso exista um clique no botão “BACK”, o que retrocede o estado do programa para “Menu”.

3.3. Algoritmo para comunicação externa (porta de série)

Como foi referido anteriormente, a nossa ferramenta permite desenho colaborativo, isto é, desenho em conjunto em duas máquinas usando comunicação por porta série.

O envio de comandos ocorre se o programa se encontrar no estado “Desenho colaborativo”, enviando-se um comando por cada ação do utilizador na tela. Os atributos de cada ação gráfica (se existirem), tais como coordenadas de aplicação, cores, dimensões, etc ... são convertidos em BCD, e separados em 2 *chars* (MSB e LSB) para serem enviados. No caso da cor, como o número em base 10 pode ir até um máximo de 5 algarismos (0xFFFF), são enviados 3 *chars*: LSB, MSB e MMSB (contendo este último, o dígito de maior potência).

Na recepção, os LSB/MSB/MMSB são juntos e convertidos para a sua forma natural, para o efeito gráfico poder ser feito.

Esta decisão de conversão em BCD permitiu a existência de *chars* que nunca serão

atributos/argumentos de um comando, (por exemplo 0xAF, 0xA2, ..) e que poderão ser iniciadores de comandos.

Significado	Iniciador de Comando	Possui argumentos ?
Círculo	0xA1	Sim
Retângulo	0xA2	Sim
Linha	0xA3	Sim
"Bucket"	0xA4	Sim
Carimbo de hora	0xA5	Sim
"Blank"	0xA6	Não
Efeito espelho	0xA7	Não
Efeito Magia	0xA8	Não

Tabela 1 - Tabela contendo os iniciadores de comando de cada comando e indicação dos que tem argumentos

Argumentos de cada comando:

- ➔ Círculo: coordenada x do centro (LSB+MSB), coordenada y do centro (LSB+MSB), raio (LSB+MSB) e cor (LSB+MSB+MMSB);

- ➔ Retângulo: coordenada x do canto superior esquerdo (LSB+MSB), coordenada y do canto superior esquerdo (LSB+MSB), dimensão horizontal (LSB+MSB), dimensão vertical (LSB+MSB) e cor (LSB+MSB+MMSB);

- ➔ Linha: coordenada x do ponto inicial (LSB+MSB), coordenada y do ponto inicial (LSB+MSB), coordenada x do ponto final (LSB+MSB), coordenada y do ponto final (LSB+MSB), espessura da linha (raio dos círculos que a formam) (LSB+MSB) e cor (LSB+MSB+MMSB);

- ➔ "Bucket": coordenada x do ponto (LSB+MSB), coordenada y do ponto (LSB+MSB) e cor a usar (LSB+MSB+MMSB);

- ➔ Carimbo de hora: coordenada x do ponto (LSB+MSB), coordenada y do ponto (LSB+MSB), número de segundos, número de minutos, número de horas, dia do mês, mês atual, ano atual e cor a usar (LSB+MSB+MMSB);

3.3.1. Algoritmo para recepção de dados

A receção de comandos efetua-se seguindo o seguinte algoritmo:

- ➔ Verificar, a cada 5 interrupções do timer, se existe algum dado no *Receiver Register* para receber;

- ➔ Se sim, processar os dados que vão sendo recebidos até não haver dados para receber ou se ter processado um total de MAX_CHARS_PER_UPDATE caracteres (que neste momento tem o valor de 2000) ou ocorrer um erro de *overrun*, *framing*, *parity*.

Cada vez que se realiza um *command update*, é invocado o *handler* do módulo de comunicação que trata de receber um *char* da porta série. Esse *char* é então processado pelo mecanismo de máquinas de estado que se encontra à frente explicado.

3.4. Uso de máquinas de estado em conjugação com *handling*

3.4.1. Uso em operações de desenho

Tal como foi descrito anteriormente, durante o desenho são usados um conjunto de *handlers* (um para cada ferramenta), e uma variável que indica o índice no vetor de apontadores para *handlers* da ferramenta em execução. Sempre que ocorre algum evento relacionado com o rato no ecrã de desenho atual, o *handler* da ferramenta em questão é executado (salvo com a ferramenta “blank”, os efeitos e a ferramenta “guardar”).

É guardado numa variável também o estado de execução da ferramenta.

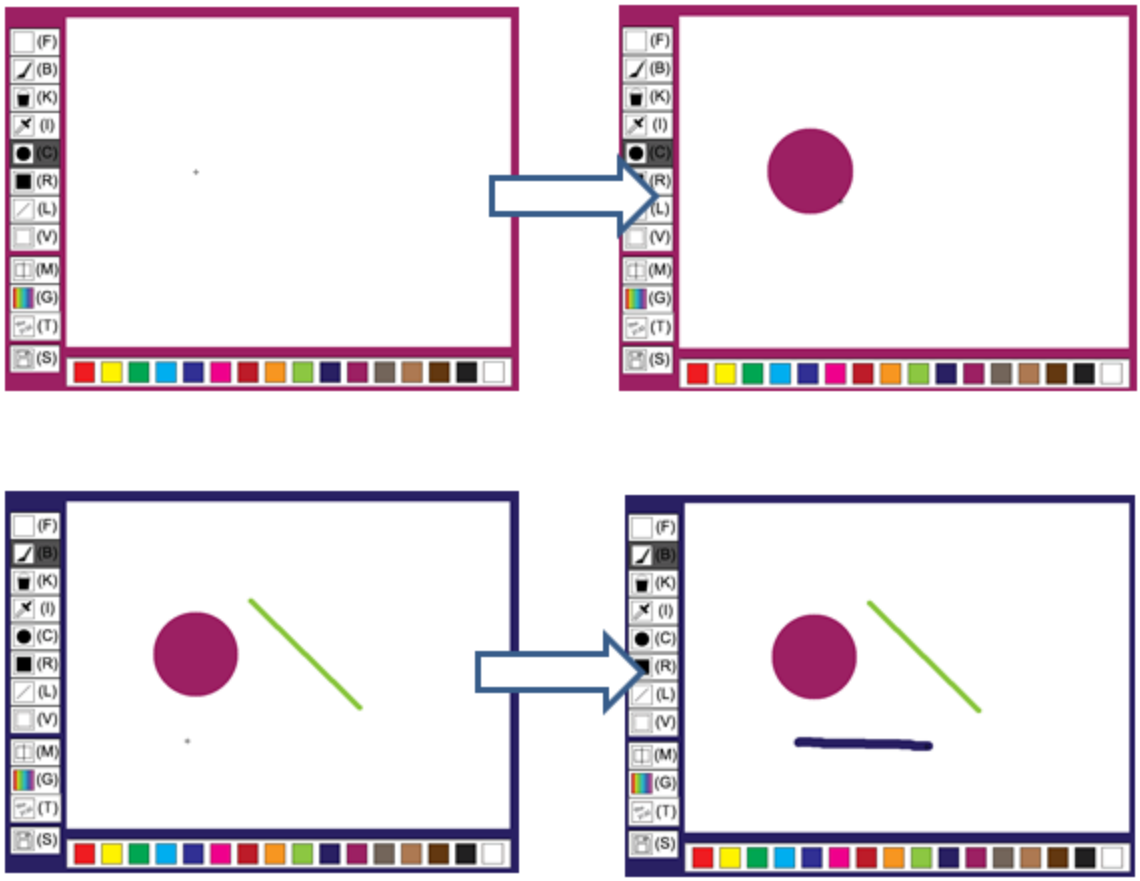


Fig. 6 - Esquema que ilustra as alternâncias de estado em várias ferramentas.

Como se pode ver no esquema acima, no primeiro caso a alternância entre um estado “st0” e “st1” permite o desenho de um círculo, em que o primeiro clique corresponde ao centro e o segundo o limite do círculo relativamente ao ponto inicial. A alternância entre estes dois estados e o arrastar do rato (com o botão esquerdo premido) permite criar alguns efeitos visuais apelativos no caso da ferramenta círculo e retângulo.

No segundo caso (ferramente *brush*), não existe alternância de estados, sendo que há uma transição de um estado inicial “st0” para um estado “st1” e a permanência neste estado “st1” enquanto a ferramenta está ativa e o rato está a ser premido e movimentado.

A memorização dos pontos anteriores e sincronização com novas coordenadas é feita a partir de variáveis *static* presentes nos *handlers* de cada ferramenta.

3.4.2. Uso em operações de envio de comandos

Durante a recepção de comandos vindos de outra máquina é também aplicado o conceito de máquina de estados de uma forma muito semelhante à descrita em 3.4.1.

Tal como foi referido anteriormente, no módulo de comunicação existe um *array* de *handlers* e uma variável que indica qual desses *handlers* está ativo neste momento (isto não é válido para comandos que não têm argumentos, isto é, que apenas contém um *char* "iniciador de comando" (nomeadamente os efeitos e ferramenta *blank*), já que estes são processados mal é recebido o seu *char* iniciador).

É guardado também numa variável o estado atual de recepção do comando.

Sempre que é recebido um *char* "iniciador de comando", é atualizado o *handler* a utilizar e os restantes dados recebidos são tratados segundo uma máquina de estados que associa a cada dado recebido a sua variável correta (tendo em conta a ordem de recepção dos dados). No estado final de cada comando, este é processado e o efeito gráfico associado é desenhado na tela.

Se a meio de de recepção de um comando é recebido um "iniciador de comando" (o que significa que os restantes *chars* do comando se perderam), o comando a ser "construído" é abandonado e é ativado o *handler* para este novo comando que vai ser recebido.

3.5 Uso de linguagem *assembly*

Foi implementado um *interrupt handler* de interrupções do RTC em *assembly*, que retorna 1, 2, 3 ou 0 caso a interrupção ocorrida seja uma *UIE Interrupt*, *Periodic Interrupt*, ou *Alarm Interrupt* ou nenhuma válida.

3.6 Algoritmos Gráficos

3.6.1. Desenho de linha

O algoritmo que estamos a usar para desenhar a linha é substancialmente mais rápido que o algoritmo de Bresenham. Segundo o autor é 10 a 12 % mais eficaz num processador Pentium 3, e pode ser encontrado em: http://willperone.net/Code/codeline.php;

Começa por calcular a diferença em x e em y, se alguma delas for nula estamos na presença de um caso especial, linha vertical ou horizontal, respectivamente. No tratamento destes casos verifica-se qual o sentido de crescimento da linha e pintamos os pixeis nessa direção até encontrarmos a extremidade.

No caso mais genérico de uma linha oblíqua é verificado o sentido de crescimento (cima/baixo e esquerda/direita), e utiliza-se um de dois métodos, desenhar linha com inclinação superior a 45º ou desenhar linha com inclinação inferior a 45º.

3.6.2. Desenho de Círculo

Para desenhar o círculo utilizamos o “algoritmo dos oitavos” disponibilizado pelo regente da Unidade Curricular nos slides das aulas teóricas.

Funciona muito rapidamente porque com uma verificação apenas pinta 8 pixels, fazendo 8 simetrias.

3.6.3. Desenho de Objectos

Para o desenho de objetos, como botões, barra de cores, menus, etc… foi criada uma função que usa o *memcpy* para tornar o programa mais eficiente, por ser uma função muito utilizada.

O algoritmo percorre o objecto a partir do canto inferior direito, o ultimo pixel, recua o numero de pixeis equivalente a sua largura e envia para memória os pixeis que recuou. Repete o numero de vezes igual ao numero de linhas do objecto.

Esta inversão deve-se ao facto de ao passar as imagens para memoria estas irem com as linhas invertidas.

3.6.4. Ferramenta do pincel

O pincel funciona com linhas sucessivas, mas em vez de desenhar um pixel em cada ponto, desenha um círculo, com centro na localização do rato e diâmetro igual ao valor da grossura da linha.

3.6.5. Carregamento de bitmaps de 16 bits

Para efetuar o carregamento dos *bitmaps* foi primeiro necessário estudar o formato. Este contém um *header* com informação sobre o ficheiro, nomeadamente o seu tamanho e distância em pixeis para o início do *array* de cores. Possui um segundo *header* com informação sobre a imagem, sendo apenas necessárias a altura e largura.

Depois de obtida esta informação é possível saltar a partir do início do ficheiro o número de *bytes* indicados no *offset* até ao início da imagem e carregar com uma única instrução para um *array* de *unsigned short*.

Para guardar os *headers* foram usadas duas estruturas auxiliares descritas em http://paulbourke.net/dataformats/bmp/.

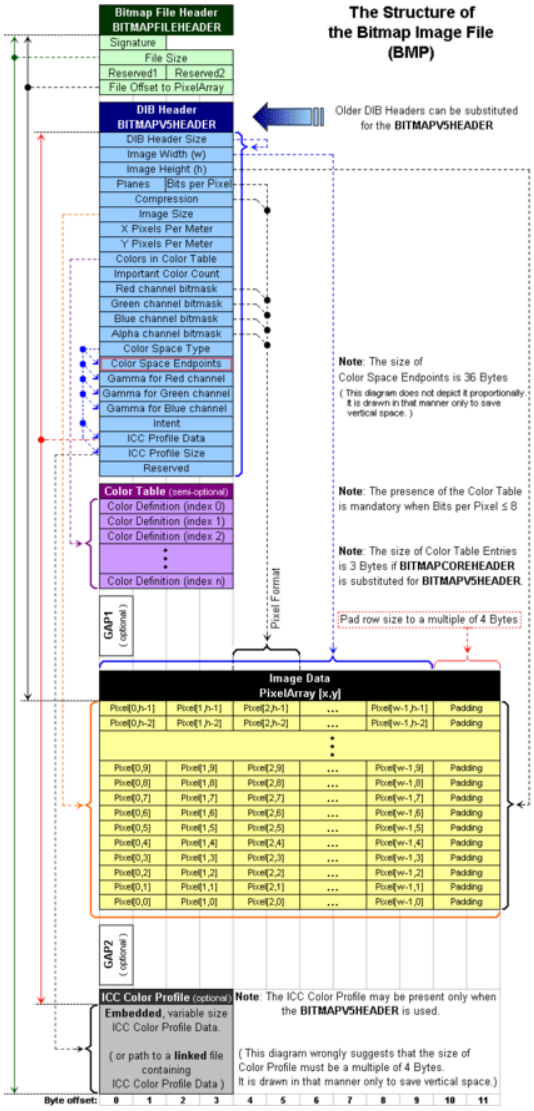


Fig. 7 - Formato BMP (http://en.wikipedia.org/wiki/BMP_file_format)

4. Diagrama de Invocação de Funções

Devido ao facto dos vários diagramas gerados terem uma dimensão grande, forem disponibilizados no diretório “doxy”, juntamente com o doxyfile.

5. Estado final do projeto

Relativamente ao descrito na especificação do projeto, praticamente todos os módulos e funcionalidades foram implementadas com sucesso.

Não foram no entanto implementadas as seguintes funcionalidades:

- ➔ Contador no ecrã do desenho (que contaria o tempo desde que se começou o desenho), por se ter considerado uma *feature* adicional, já que o *timer* já está a ser utilizado no trabalho para outro propósito.

- ➔ Função em *assembly* para efeito espelho, tendo sido feita uma versão em C.

6. Auto Avaliação

Considera-se que houve cooperação geral no desenvolvimento de todos os módulos.

Contudo, ficou a cargo de João Neto o desenvolvimento dos algoritmos gráficos de desenho, módulo gráfico, módulo galeria e módulo de desenho.

Por sua vez, ficou a cargo de Miguel Sandim a junção das bibliotecas de baixo nível criadas nos *labs* no projeto, e a sua interação no programa (nomeadamente módulo de interação com o utilizador, módulo do tempo, módulo de transferência). Ficou também a cargo de Miguel, a organização por eventos e máquinas de estado.

Relativamente à documentação, foi da responsabilidade de cada um dos membros do grupo a documentação dos seus módulos.

Acerca do peso relativo de participação, o grupo considera que o Miguel Sandim participou com mais horas no projecto, pelo que o João Neto se auto avalia com 45% e o Miguel Sandim com 55%.

Segundo o grupo, a contribuição para o resultado final foi igual entre todos, pelo que todos os membros se auto avaliam com 50%.