TUTORIAL DE UTILIZAÇÃO DA BIBLIOTECA ALLEGRO PARA INICIANTES

Rafael Loosli Dias

Roberto Ferrari Júnior

Sumário

1. Introdução

O intuito deste tutorial é possibilitar que o aluno possa ter uma noção do funcionamento de certas diretivas voltadas à programação de jogos utilizando a biblioteca Allegro. Serão abordados conceitos básicos orientados a uma programação gráfica 2D, porém vale a pena ressaltar que existem outras APIs (*Application Programming Interface*) que possibilitam trabalhar com recursos gráficos mais poderosos. Contudo, a sistemática de utilização de certas funções é, de certa forma, semelhante entre os diferentes APIs disponibilizadas.

Para exemplificar a utilização da biblioteca Allegro, foi escolhida a IDE (*Integrated Development Enviroment*) Dev-C++, visto que a mesma é de fácil compreensão para utilização e possui assistência para a instalação de pacotes através da sua própria interface, facilitando a adição de novos plugins ou bilbiotecas, por exemplo. Contudo, vale ressaltar que a mesma possuí algumas complicações, como a falta de recursos para Debug de código, por exemplo.

É desejável que o aluno tenha conhecimento básico sobre a linguagem C++.

2. Instalação

Existem diversas possibilidades para a utilização da biblioteca Allegro na linguagem C++. Contudo, para facilitar o desenvolvimento, recomenda-se que seja utilizada uma IDE, ou ainda um Ambiente Integrado de Desenvolvimento, o qual possui artifícios para facilitar o trabalho do programador.

Existem diferentes IDEs que podem ser utilizadas. Neste tutorial, considera-se que o programador estará usando uma IDE com MinGW, sendo esta uma coleção de arquivos específicos para o funcionamento em Windows e que, combinadas com ferramentas da GNU, permitem a produção de programas nativos para o mesmo sistema operacional. São levantadas algumas das seguintes IDEs para utilização:

- Microsoft Visual C++ Express: Encontrada no link: http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express)
- CodeBlocks com MinGW: Encontrada no link: http://www.codeblocks.org/downloads/26.
- Dev-C++ com MinGW/GCC: Encontrada no link: http://www.bloodshed.net/dev/devcpp.html.

Após concluído o download da IDE desejada, é necessário também baixar a biblioteca Allegro que será utilizada. Para aqueles que optarem pela IDE Dev-C++, existe uma seção deste tutorial que demonstra como realizar esse procedimento de instalação da biblioteca através da própria interface da IDE.

A biblioteca em sua versão 4.2.2, a qual será utilizada neste tutorial, pode ser encontrada no link: http://sourceforge.net/projects/alleg/files/allegro-bin/4.2.2/ (Baixar o arquivo “allegro-mingw-4.2.2.zip”).

Para que seja possível rodar aplicações fora do ambiente da IDE, é necessário baixar a biblioteca (.dll) e colocá-la em seu devido diretório. Considerando a plataforma Windows, a biblioteca (“alleg42.dll”) deve ser copiada no diretório “\Windows\System32”.

2.1 Utilizando o Dev - C++

Após concluído o download da IDE (seguindo o link citado anteriormente), é preciso instalar e fazer os links para a biblioteca Allegro diretamente através da interface do Dev-C++. Para tal, basta iniciar o Dev-C++ e selecionar a opção "Tools >> Check for Updates/Packages", como ilustrado na Figura 1.

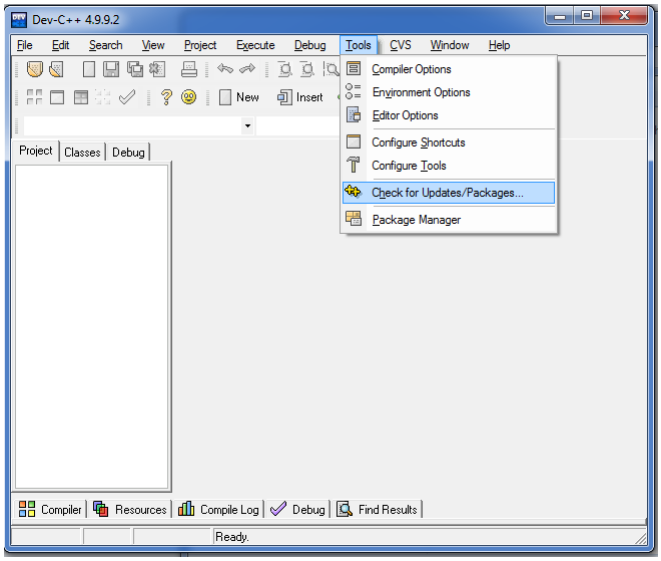

Figura 1. Após abrir o Dev-C++, selecionar a opção: *Tools >> Check for Updates/Packages.*

Uma nova janela irá se abrir. Esta possibilita a conexão com determinados servidores para downloads de pacotes para o Dev-C++. Logo, para o próximo passo, é preciso selecionar o devido servidor como descrito na Figura 2.

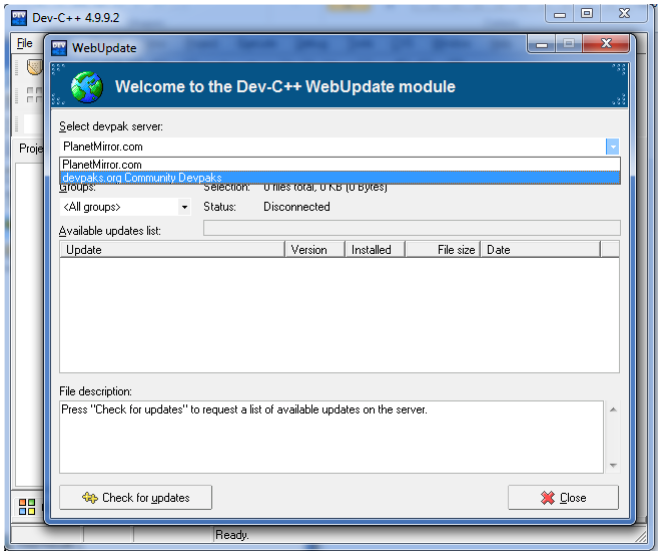

Figura 2. Selecionar a opção de servidor "*devpaks.org Community Devpack*".

Em seguida, clicar no botão "Check for updates", o que possibilita procurar no servidor selecionado todos os possíveis upgrades para a IDE Dev-C++.

Desta forma, serão exibidos diversos pacotes para instalação. Neste tutorial, será utilizado a biblioteca Allegro na versão 4.2.2, como descrito na Figura 3.

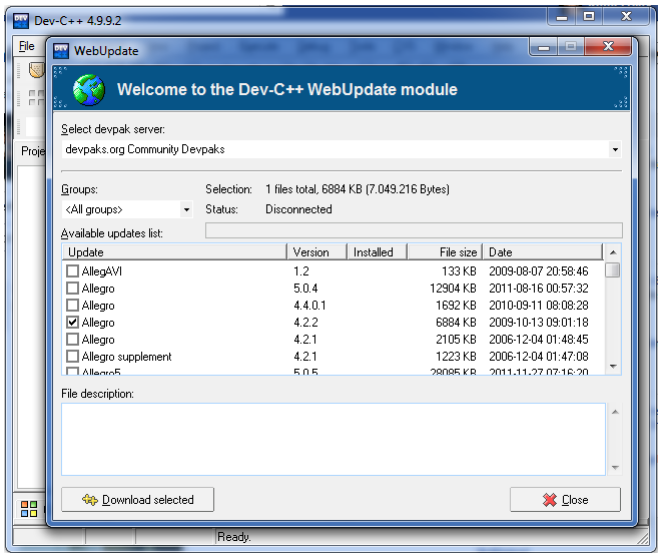

Figura 3. Após selecionada a opção de servidor "devpaks.org Community Devpaks", selecionar a biblioteca Allegro (versão 4.2.2) para download.

Após selecionado o pacote especificado, clicar no botão "Download selected". Em alguns instantes, após finalizado o download, caso o mesmo seja executado corretamente, será exibida uma mensagem como descrito na Figura 4.

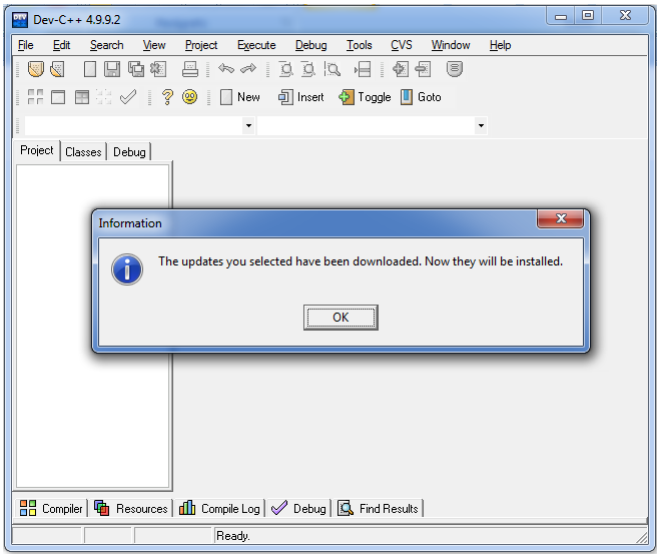

Figura 4. Mensagem de confirmação de download completo.

Após terminado o download, um assistente para instalação de pacotes para a IDE Dev-C++ será automaticamente aberto, como ilustrado na Figura 5. O mesmo irá auxiliar na tarefa de instalação da biblioteca Allegro desejada.

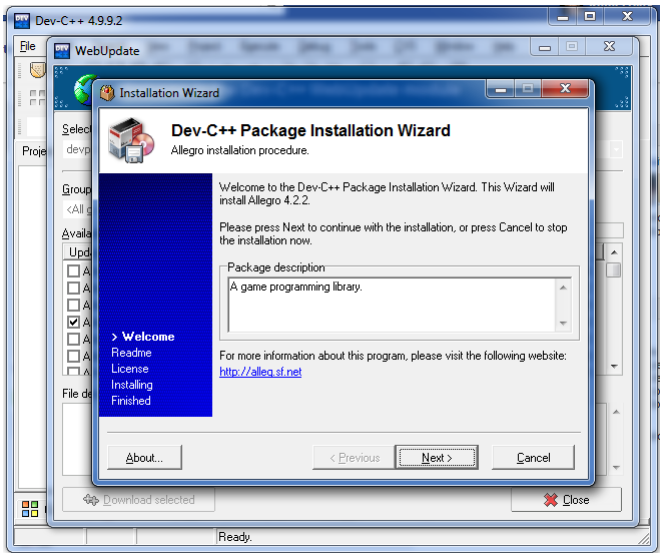

Figura 5. Assistente para instalação de pacotes automaticamente aberto durante o procedimento descrito.

É necessário prosseguir com a instalação até que a mesma seja concluída. Caso esse procedimento seja executado com êxito, será exibida a mensagem demonstrada na Figura 6.

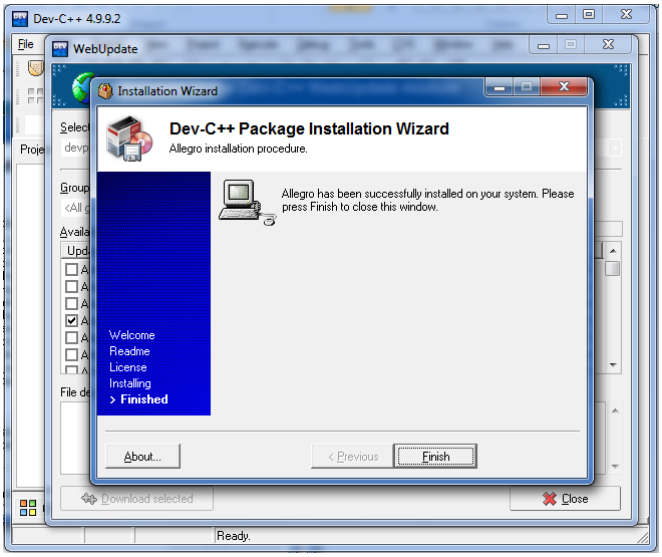

Figura 6. Mensagem de confirmação de êxito na instalação do pacote (neste caso, a biblioteca Allegro na versão 4.2.2)

Se todos os passos aqui demonstrados forem executados corretamente, o aluno estará pronto para começar a utilizar os recursos disponibilizados pela Allegro no ambiente de desenvolvimento provido pela Dev-C++.

3. Primeiro programa utilizando a Allegro

Após instalar corretamente a biblioteca Allegro para o uso no Dev-C++, deve-se então criar um novo projeto para iniciar o nosso primeiro programa. Para tal, basta selecionar a opção "File >> New >> Project", como apresentado na Figura 7.

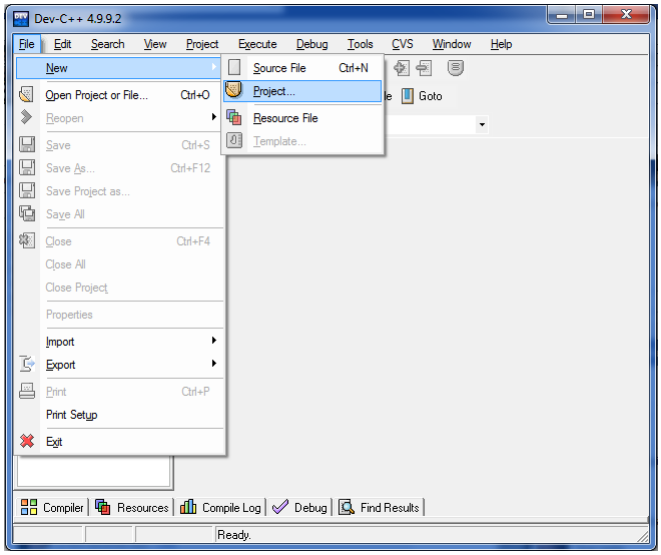

Figura 7. Selecionar a opção de criação de um novo Projeto

Assim, uma nova janela abrirá com as opções de Projeto disponibilizadas pelo Dev-C++. Logo, basta selecionar a aba "MultiMedia", marcar a opção "Allegro application (static)", dar o nome desejado ao projeto e, finalmente, clicar no botão "Ok". Tais procedimentos estão descritos na Figura 8.

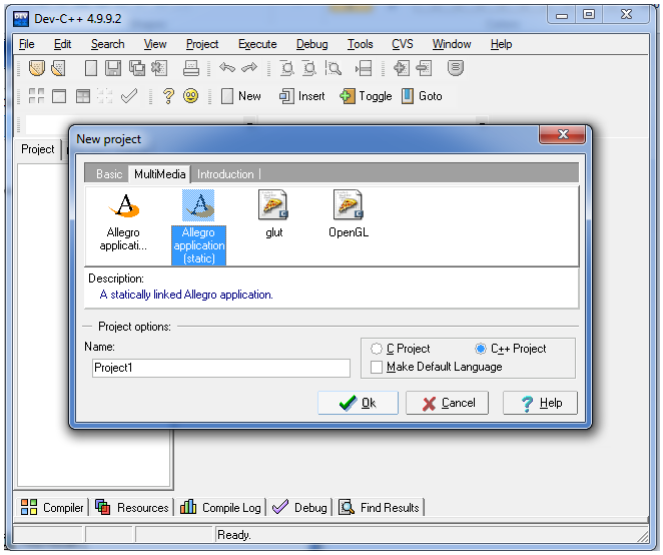

Figura 8. Criação de um novo projeto "Allegro application (static)".

Posteriormente, será perguntado onde o novo projeto deve ser salvo. O mesmo pode ser em qualquer diretório desejado pelo programador.

Finalmente, se feito corretamente, o novo projeto será criado e um pequeno trecho inicial de código é introduzido ao usuário. O código descrito deve ser semelhante ao que segue:

```
#include <allegro.h>

void init();
void deinit();

int main() {
	init();

	while (!key[KEY_ESC]) { /*Enquanto a tecla ESC não for pressionada, irá
				  executar o código especificado */

		/* Código a ser executado */
	}

	deinit();
	return 0;
}
END_OF_MAIN() /* Macro necessária para o funcionamento da Allegro
		 Sempre deve ser utilizada no final da função main*/

void init() {
	int depth, res;
	allegro_init(); /* Inicializa a Allegro */
	depth = desktop_color_depth(); /* Retorna a profundidade de cores usadas no
					  Desktop */
	if (depth == 0) depth = 32;
	set_color_depth(depth); /* Define a profundidade de cor que será usada */

	/* Seleciona o modo gráfico a ser exibido
	   Neste caso, será em janela (poderia ser FullScreen, por exemplo), com
	   tamanho 640 x 480 */
	res = set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
	if (res != 0) {
		allegro_message(allegro_error); /* Exibe mensagem em caso de erro */
		exit(-1);
	}

	install_timer(); /* Instala o handler de tempo da Allegro */
	install_keyboard(); /* Instala o handler de teclado da Allegro */
	install_mouse(); /* Instala o handler de mouse da Allegro */
	/* Outras possíveis inicializações */

}

void deinit() {
	clear_keybuf(); /* Limpa o buffer do teclado */
	/* Outras possíveis desinicializações */
}
```

Caso o programador compile e rode o programa então gerado, apenas uma janela será aberta, com a opção de fechá-la ao pressionar a tecla "ESC". Logo, esse trecho de código gerado somente traz algumas inicializações e chamadas para iniciar um programa com interface gráfica proporcionada pela biblioteca. A partir desse momento, o programador começará a desenvolver o seu próprio código.

4. Funções disponibilizadas pela Allegro

Existem diversas funções disponibilizadas pela biblioteca Allegro para permitir que o programador desenvolva adequadamente o seu jogo. Nessa seção, serão mostradas algumas

das funções mais usualmente utilizadas pelos desenvolvedores, mas vale ressaltar que o manual da biblioteca deve ser consultado no momento em que existir alguma dificuldade ou necessidade.

O manual de utilização da biblioteca Allegro (versão 4.2.2) pode ser encontrado em http://alleg.sourceforge.net/stabledocs/en/allegro.html.

4.1 Sintaxe da biblioteca e funções gráficas

Inicialmente, vale a pena ressaltar que, na Allegro, as coordenadas nas imagens são tratadas diferentemente de gráficos e outros elementos que possuímos em nosso mundo real. Dessa forma, a origem encontra-se no canto superior esquerdo ao invés do canto inferior esquerdo. A Figura 9 ilustra melhor o sistema de coordenadas utilizado.

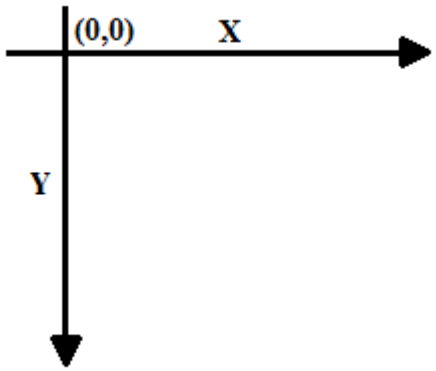

Figura 9. Sistema de coordenadas utilizado pela biblioteca Allegro

Para estudo, utilizaremos o seguinte trecho de código base:

```
#include <allegro.h>

int main() {
	allegro_init();
	install_keyboard();

	set_color_depth(32);
	set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);

	while (!key[KEY_ESC]) {
		/* Código a ser executado */
	}
	return 0;
}
END_OF_MAIN()
```

Logo, temos as seguintes chamadas:

- allegro_init(): Macro utilizada para inicializar a biblioteca Allegro. A mesma deve aparecer antes de qualquer outra chamada provida pela biblioteca.

- install_keyboard(): Instala o handler de teclado da Allegro. Assim, pode-se ter o controle de eventos e do comportamento das teclas do teclado na execução do programa. Um exemplo visível da necessidade dessa chamada encontra-se no loop "while (!key[KEY_ESC])" , visto que os comandos internos ao mesmo serão executados até o momento em que o usuário pressionar a tecla ESC.

- set_color_depth(32): Define a profundidade de cores que serão utilizadas no programa ou, em outras palavras, a quantidade de bits para representar as cores utilizadas. Logo, esta é primordial de ser feita antes de qualquer outra chamada gráfica. Possíveis valores são: 8 (default), 15, 16, 24 e 32 bits.

- set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0): Seleciona o modo gráfico que será exibido. Neste caso, será fixada uma janela de tamanho 640x480 para exibição. Contudo, poderia ser utilizado o parâmetro "GFX_AUTODETECT_FULLSCREEN", possibilitando que fosse forçada a exibição em tela cheia.

- key[KEY_ESC]): Verifica se a tecla ESC foi pressionada. No caso do código utilizado, a intenção era executar o trecho do programa quando tal tecla não fosse pressionada, assim basta-se negar a verificação com o símbolo '!".

- END_OF_MAIN(): Macro necessária para o funcionamento da Allegro, a qual deve ser colocada imediatamente depois da função main.

4.2 Adicionando uma string no display

Agora que já tem-se o programa básico, é viável incrementar o código especificando o comportamento desejado no programa. Assim, inicialmente, pode-se colocar uma chamada para exibir um determinado texto na janela criada, usando o comando:

```
      void textprintf_ex(BITMAP *bmp, const FONT *f, int x, int y, int color, int
bg, const char *fmt, ...);
```

Logo, os parâmetros utilizados são:

- O primeiro parâmetro (BITMAP *bmp) refere-se a um buffer onde será armazenada a informação para, posteriormente, ser visualizado na janela do programa. Já existe um ponteiro específico para bitmap utilizado pela Allegro para exibir as informações no display, sendo este denominado "screen".

- O segundo parâmetro (const FONT *f) é a fonte a ser utilizada para a exibição da string em questão. Por padrão, existe a fonte denominada "font".

- O terceiro e o quarto parâmetro (int x, int y) representam, respectivamente, a distância dos eixos X e Y em que a string deverá começar a ser escrita.

- O quinto parâmetro (int color) refere-se à cor em que a string deverá ser exibida. Normalmente, é utilizada a chamada makecol("R","G","B"), fazendo com que seja

possível escolher a cor através da especificação dos níveis de vermelho, verde e azul, respectivamente.Vale a pena ressaltar que esses valores de intensidade devem estar no intervalo entre 0 e 255.

- Por fim, é colocada a string desejada para a exibição. A forma com que a string é utilizada é semelhante à chamada printf, onde pode-se também colocar valores referentes a variáveis do contexto em questão.

Logo, pode-se realizar o seguinte exemplo:

```
#include <allegro.h>

int main() {
	allegro_init();
	install_keyboard();

	set_color_depth(32);
	set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);

	while (!key[KEY_ESC]) {
		textprintf_ex( screen, font, 10, 10, makecol(255,255,255), -1, "Hello
World");
		textprintf_ex( screen, font, 10, 20, makecol(255,0,0), -1, "Exercicio
numero: %i", 1);
	}
	return 0;
}
END_OF_MAIN()
```

Assim, a saída desse programa será uma janela de tamanho 640X480 com as strings "Hello World" (na cor branca) e "Exercicio numero:1" (na cor vermelha), com o ponteiro para as mesmas iniciados respectivamente em (10,10) e (10,20).

Vale a pena ressaltar que, neste exercício, foi utilizada a fonte padrão disponibilizada pela Allegro. Contudo, caso seja necessário, é também possível criar e utilizar uma fonte customizável. Este assunto não será tratado no tutorial e ficará como exercício.

4.3 Criando e adicionando um objeto no display

Uma possível necessidade pode ser a de exibir ao usuário um objeto, como uma linha, um quadrado, um triângulo, etc. Logo, a Allegro possuí algumas primitivas para sanar este problema.

A chamada a seguir permite que seja desenhada um segmento de reta

```
	void line(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
```

A mesma possui os seguintes parâmetros:

- O primeiro parâmetro (BITMAP *bmp), como citado na seção anterior, refere-se a um buffer onde será armazenada a informação para, posteriormente, ser visualizado na janela do programa. Já existe um ponteiro específico para bitmap utilizado pela Allegro para exibir as informações no display, sendo este denominado "screen".

- O segundo e o terceiro parâmetro (int x1, int y1) definem o ponto de início do segmento de reta. Neste momento, vale a pena relembrar o sistema de coordenadas utilizado pela Allegro.

- O quarto e o quinto parâmetro (int x2, int y2) definem o ponto final do segmento de reta. Novamente, vale rever o sistema de coordenadas da biblioteca.

- O sexto parâmetro (int color), como já levantado anteriormente, refere-se à cor em que a string deverá ser exibida. Normalmente, é utilizada a chamada makecol("R","G","B"), fazendo com que seja possível escolher a cor através da especificação dos níveis de vermelho, verde e azul, respectivamente.Vale a pena ressaltar que esses valores de intensidade devem estar no intervalo entre 0 e 255.

Logo, tem-se o seguinte exemplo de segmento de reta:

```
#include <allegro.h>

int main() {
	allegro_init();
	install_keyboard();

	set_color_depth(32);
	set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);

	while (!key[KEY_ESC]) {
		line(screen,100,50,300,200, makecol(0,0,255));
	}
	return 0;
}
END_OF_MAIN()
```

Além da linha, existem outras primitivas, como o contorno de retângulo e um círculo preenchido, por exemplo.

```
	void rect(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
```

Os parâmetros desta chamada são semelhantes aos do segmento de reta, sendo os valores de x1, y1, x2, y2, delimitadores de dois dos vértices opostos do retângulo.

```
	void circlefill(BITMAP *bmp, int x, int y, int radius, int color);
```

Novamente, os parâmetros são semelhantes, contudo os valores de x e y são utilizados para determinar a coordenada do centro do círculo. Além disso, o valor de radius corresponde ao tamanho do raio desejado.

Logo, tem-se o seguinte exemplo:

```
#include <allegro.h>

int main() {
	allegro_init();
	install_keyboard();

	set_color_depth(32);
	set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);

	while (!key[KEY_ESC]) {
		rect(screen, 50, 50, 250, 150, makecol(255,255,255));
		circlefill(screen, 150, 100, 50, makecol(0,0,255));
	}
	return 0;
}
END_OF_MAIN()
```

Ao rodar o programa, percebe-se que foi desenhado um círculo de cor azul, com um raio equivalente a 50 unidades, dentro de um contorno de retângulo de cor branca.

4.4 Adicionando uma imagem Bitmap já existente

Caso haja a necessidade de adicionar um bitmap externo já existente em seu programa, o desenvolvedor deve seguir os seguinte passos:

Inicialmente, deve-se carregar o imagem desejada, utilizando-se a seguinte chamada:

```
	BITMAP *load_bitmap(const char *filename, RGB *pal);
```

- O primeiro parâmetro (const char *filename), corresponde ao caminho para a imagem Bitmap em questão. Caso a imagem esteja no mesmo diretório do projeto, basta o nome da imagem, mas se a mesma encontra-se em outra localidade, é preciso especificar o caminho (ex: "imagens\\imagem_exemplo.bmp", a "imagem_exemplo" encontra-se no diretório "imagens" que está localizado dentro do diretório do projeto corrente).
- O segundo parâmetro (RGB *pal) define a palheta de cores utilizada na imagem Bitmap em questão. Este valor pode ser passado nulo ("null"), já que, para este momento, não há necessidade de retornarmos a palheta de cores utilizadas na imagem.

Posteriormente, é necessário copiar a imagem Bitmap em um buffer de destino, utilizando a seguinte chamada:

```
	void blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int
dest_x, int dest_y, int width, int height);
```

- O primeiro parâmetro é buffer (imagem) de origem, enquanto o segundo é o buffer (imagem) de destino.

- O terceiro e o quarto parâmetro (source_x, int source_y) expressam a partir de qual coordenada do buffer (imagem) de origem deve ser copiada para o buffer (imagem) de destino. Assim, estes parâmetros representam o ponto inicial (origem) da imagem de origem que deve ser copiado para o novo buffer, permitindo então que a imagem seja cortada dependendo da necessidade do desenvolvedor.

- O quinto e o sexto parâmetro (int dest_x, int dest_y) identificam a posição da origem (coordenada (0,0)) do buffer (imagem) de origem em relação ao buffer (imagem) de destino. Logo, estes parâmetros são responsáveis pelo posicionamento da imagem a ser inserida no buffer de destino.

- O sétimo e o oitavo parâmetro (int width, int height) são responsáveis por indicar o tamanho do buffer (imagem) de origem que será copiado no buffer(imagem) de destino. Novamente, esse parâmetros podem ser utilizados para recortar uma determinada parte da imagem, já que especificam a coordenada máxima (pixel inferior direito) da imagem de origem que será copiada.

Por fim, é importante destruir o bitmap carregado para desalocar o buffer utilizado, usando a seguinte função:

```
void destroy_bitmap(BITMAP *bitmap);
```

- Permite desalocar o buffer que contem a imagem apontada pelo ponteiro BITMAP *bitmap.

Para exemplo, vamos tomar a imagem ilustrada na Figura 10, a qual deve ser colocada no diretório corrente do projeto (como especificado no desenvolvimento), e o trecho de código que segue.

Figura 10. Imagem utilizada no exemplo (Tamanho real 300x200)

```
#include <allegro.h>

int main() {
	allegro_init();
	install_keyboard();

	set_color_depth(32);
	set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
	BITMAP *imagem_ufscar = load_bitmap("ufscar.bmp",NULL);
	while (!key[KEY_ESC]) {
		blit(imagem_ufscar, screen, 120, 0, 100, 50, 300, 200);
	}
	destroy_bitmap(imagem_ufscar);
	return 0;
}
END_OF_MAIN()
```

Ao rodar o programa, o usuário verificará que a imagem de origem foi cortada em relação ao eixo X e posicionada com a origem na coordenada (300,200) do display.

4.5 Inserindo diversos elementos gráficos

Até o momento, foram tratados elementos gráficos isoladamente. Contudo, na maioria das vezes, é necessário utilizar várias desses recursos juntamente para atingir o objetivo determinado pelo programa. Desta forma, existem algumas boas práticas de desenvolvimento que mostram-se interessantes de serem utilizadas.

Primeiramente, mostra-se interessante inserir todos esses elementos gráficos em um mesmo buffer e, somente no fim de cada iteração, copiar este buffer para o buffer destinado a visualização no display (denominado "screen"). Logo, inicialmente, é necessário criar um buffer (BITMAP *) onde tais elementos serão inseridos, utilizando a seguinte chamada:

```
	BITMAP *create_bitmap(int width, int height);
```

- Os parâmetros "int width" e "int height" especificam, respectivamente, a largura (eixo X) e a altura (eixo Y) do buffer a ser criado.

Em seguida, o desenvolvedor pode adicionar quaisquer elementos gráficos que desejar através da seguinte chamada já mencionada no tutorial:

```
	void blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int
dest_x, int dest_y, int width, int height);
```

Contudo, existe uma alternativa para essa chamada. Pode existir, por exemplo, a necessidade de não exibir certos elementos de uma imagem a ser carregada, já que a mesma possui alguns pixels com valores transparentes. Logo, a seguinte chamada pode ser utilizada como a anterior:

```
void draw_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y);
```

Os parâmetros são:

- O primeiro parâmetro (BITMAP *bmp) é o buffer destino, ou seja, o buffer onde a imagem de origem será copiada.
- O segundo parâmetro (BITMAP *sprite) representa a imagem de origem a ser copiada no buffer. Neste caso, o objeto pode possuir alguns pixels com valores transparentes, indesejáveis para a visualização.
- O terceiro e o quarto parâmetro (int x, int y), semelhantemente à chamada "void blit", são responsáveis por indicar o tamanho do buffer (imagem) de origem que será copiado no buffer de destino.

Assim, após inseridos todos os elementos gráficos em somente um buffer, o desenvolvedor pode copiar todo o conteúdo do mesmo para o buffer destinado à visualização no display (definido como "screen")

Finalmente, após a inserção dos elementos gráficos no buffer e a visualização dos mesmos, é necessário esvaziar os buffers utilizados. Para isto, temos a seguinte função:

```
void clear_bitmap(BITMAP *bmp);
```

Onde o parâmetro "BITMAP *bmp" representa o buffer a ser esvaziado (colocado o valor 0 para o bitmap).

Vale a pena ressaltar que esta prática de liberar o conteúdo contido no buffer é muitas vezes necessário durante o desenvolvimento do programa, visto que o conteúdo de cada iteração do loop principal continuaria em memória e, consequentemente, seria exibido com os demais elementos de uma outra iteração. Desta forma, poderia haver uma sobreposição indesejada de informações gráficas, sendo necessário então esvaziar o buffer a cada iteração.

Tendo em mente as práticas aqui sinalizadas, pode-se elaborar o seguinte exemplo de código:

```
#include <allegro.h>

int main() {
       allegro_init();
       install_keyboard();

       set_color_depth(32);
       set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
       BITMAP *buffer = create_bitmap(640,480);
       BITMAP *imagem_ufscar = load_bitmap("ufscar.bmp",NULL);
       while (!key[KEY_ESC]) {
         blit(imagem_ufscar, buffer, 0, 0, 100, 50, 300, 200);
         circlefill(buffer, 400, 250, 50, makecol(255,69,0));
          textprintf_ex( buffer, font, 360, 250, makecol(0,0,0), -1, "Hello
World");
          blit(buffer,screen,0,0,0,0,640,480);
          clear_bitmap(buffer);
       }
       destroy_bitmap(imagem_ufscar);
       destroy_bitmap(buffer);
       return 0;
}
END_OF_MAIN()
```

Neste código, foi inserida uma imagem já existente em disco, um círculo de cor laranja e uma string ("Hello World"), assim como ilustrado na Figura 11. Para tanto, foram utilizadas as práticas aqui mencionadas.

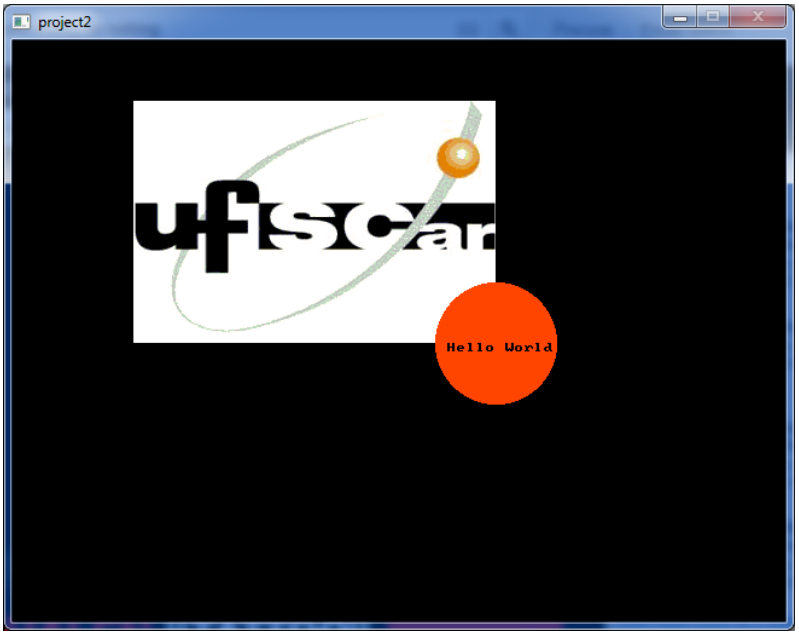

Figura 11. Saída para o código levantado

5. Utilizando o teclado

O teclado é um dos periféricos mais utilizados para a interação do computador com o usuário. Assim, a biblioteca Allegro provê alguns mecanismos para que o desenvolver possa utilizar essa ferramenta adequadamente.

As principais chamadas são:

```
install_keyboard();
```

- Permite que o handler de teclado provido pela Allegro seja instalado. Essa chamada deve constar antes de qualquer outra função de teclado utilizada ao longo do programa.

```
extern volatile char key[KEY_?];
```

- Verifica se determinada tecla do teclado foi pressionada. Existem diferentes possibilidades de teclas a serem verificadas, sendo estas especificadas na Figura 12.

```
KEY_A ... KEY_Z,
KEY_0 ... KEY_9,
KEY_0_PAD ... KEY_9_PAD,
KEY_F1 ... KEY_F12,

KEY_ESC, KEY_TILDE, KEY_MINUS, KEY_EQUALS,
KEY_BACKSPACE, KEY_TAB, KEY_OPENBRACE, KEY_CLOSEBRACE,
KEY_ENTER, KEY_COLON, KEY_QUOTE, KEY_BACKSLASH,
KEY_BACKSLASH2, KEY_COMMA, KEY_STOP, KEY_SLASH,
KEY_SPACE,

KEY_INSERT, KEY_DEL, KEY_HOME, KEY_END, KEY_PGUP,
KEY_PGDN, KEY_LEFT, KEY_RIGHT, KEY_UP, KEY_DOWN,

KEY_SLASH_PAD, KEY_ASTERISK, KEY_MINUS_PAD,
KEY_PLUS_PAD, KEY_DEL_PAD, KEY_ENTER_PAD,

KEY_PRTSCR, KEY_PAUSE,

KEY_ABNT_C1, KEY_YEN, KEY_KANA, KEY_CONVERT, KEY_NOCONVERT,
KEY_AT, KEY_CIRCUMFLEX, KEY_COLON2, KEY_KANJI,

KEY_LSHIFT, KEY_RSHIFT,
KEY_LCONTROL, KEY_RCONTROL,
KEY_ALT, KEY_ALTGR,
KEY_LWIN, KEY_RWIN, KEY_MENU,
KEY_SCRLOCK, KEY_NUMLOCK, KEY_CAPSLOCK

KEY_EQUALS_PAD, KEY_BACKQUOTE, KEY_SEMICOLON, KEY_COMMAND
```

Figura 12. Imagem retirada do Manual da biblioteca Allegro (versão 4.2.2). Possíveis teclas para verificação disponibilizadas pela biblioteca.

Um exemplo de utilização dos recursos de teclado provido pela Allegro pode ser encontrado em diversos exercícios propostos neste tutorial. Nota-se que, para sair do laço principal da biblioteca, colocamos uma verificação para a tecla “ESC” pressionada. Desta forma, a aplicação Allegro deixa de executar pela ação do usuário ao pressionar a tecla, como pode ser novamente visto no trecho de código abaixo.

```
#include <allegro.h>

int main() {
       allegro_init();
       install_keyboard();

       set_color_depth(32);
       set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);

       while (!key[KEY_ESC]) {
         // Até que a tecla "ESC" seja pressionada, a aplicação permanece no laço
         // principal.
       }

       return 0;
}
END_OF_MAIN()
```

6. Utilizando o mouse

Para a utilização do mouse, a biblioteca Allegro possui algumas chamadas para facilitar o desenvolvedor na utilização do mesmo. Dentre as mais utilizadas, estão:

```
int install_mouse();
```

- Permite a instalação do handler do mouse provido pela biblioteca Allegro. Esta chamada deve constar no código antes de qualquer outra função de mouse utilizada ao longo do programa.

Após instalado o handler para o mouse, é necessário fazer com que o mesmo seja visualizado no display para que o mesmo passa ser usado devidamente. Logo, é utilizada a seguinte chamada:

```
void show_mouse(BITMAP *bmp)
```

- O parâmetro "BITMAP *bmp" é o buffer onde o cursor do mouse deve ser desenhado. Como já citado, já existe um ponteiro específico para bitmap utilizado pela Allegro para exibir as informações no display, sendo este denominado "screen".

Desta forma, é possível utilizar o mouse através da visualização do cursor. Além disso, ainda existem alguns recursos proporcionados pela biblioteca para que esse recurso de interação com o usuário seja melhor aproveitado.

Pode-se verificar a posição do cursor em relação às coordenadas do display, através das variáveis globais:

```
      extern volatile int mouse_x; // Retorna a posição relativa ao eixo X

      extern volatile int mouse_y; // Retorna a posição relativa ao eixo Y
```

Além disso, existe uma variável que indica o estado de cada botão, denominada "mouse_b", onde cada bit especifica um determinado botão do mouse. O bit "1" guarda as informações de estado do botão esquerdo e o bit "2"do botão direito. O melhor uso da mesma é explicado através do seguinte trecho de código:

```
#include <allegro.h>

int main() {
    int x;
    int y;
    int x1;
    int y1;

    allegro_init();
    set_color_depth(32);
    set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
    BITMAP *buffer = create_bitmap(640,480);
    install_keyboard();
    install_mouse();
    show_mouse(screen);
    while (!key[KEY_ESC]) {
        if (mouse_b & 1)
        {
           x = mouse_x;
           y = mouse_y;
        }
        if (mouse_b & 2){
           x1 = mouse_x;
           y1 = mouse_y;
        }
        circlefill(buffer, x, y, 50, makecol(255,255,255));
        circlefill(buffer, x1, y1, 50, makecol(0,0,255));
        blit(buffer,screen,0,0,0,0,640,480);
        clear_bitmap(buffer);
    }
       return 0;
}
END_OF_MAIN()
```

No exemplo acima, vemos que existem duas verificações no loop, sendo estas "`mouse_b & 1`" e "`mouse_b & 2`". Através destas, podemos respectivamente verificar se o usuário pressionou o botão esquerdo ou direito do mouse. Além disso, caso alguma das verificações for atendida, os valores das posições *x* e *y* do cursor são armazenadas através da verificação das variáveis globais "`mouse_x`" e "`mouse_y`".

Por fim, utilizando as informações do clique e posicionamento do cursor dadas pelo usuário, são desenhados dois círculos. O primeiro, com a cor branca, é desenhado na posição designada pelo clique do botão esquerdo, enquanto que o segundo, de cor azul, é desenhado na posição do clique do botão direito. As imagens abaixo ilustram dois possíveis posicionamentos dos círculos orientados pelo usuário.

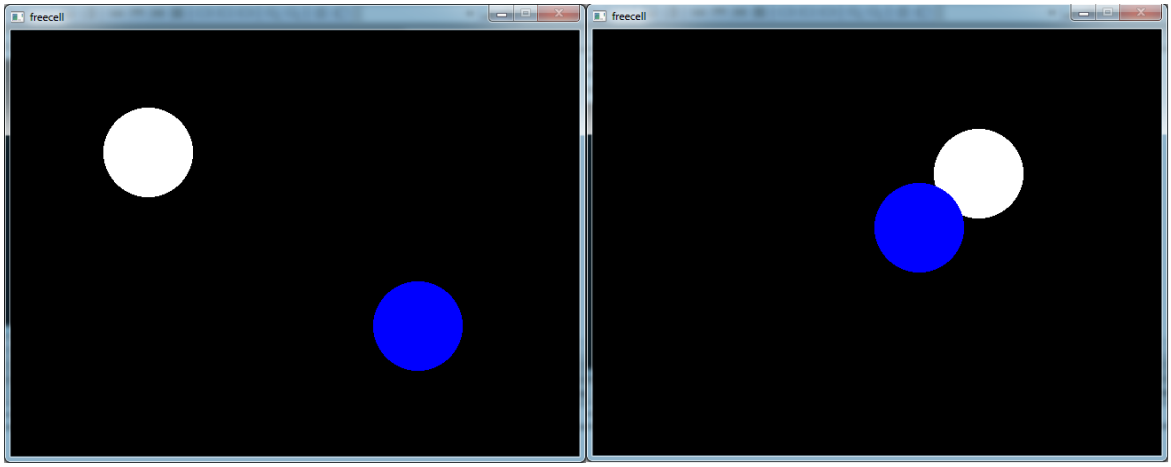

Figura 13 Possíveis posicionamentos das circunferências orientados pelo clique e movimentação do mouse.

7. Aplicação: FreeCell

É sugerida a implementação do jogo FreeCell para aplicação dos conceitos de estrutura de dados e utilização dos recursos da biblioteca Allegro apresentados neste tutorial.

Neste jogo de baralho, o conceito de Pilha é amplamente utilizado. As cartas são movimentadas de forma a respeitar regras específicas, contudo sempre mantendo os conceitos principais da estrutura. Estes, em alto nível, estão relacionados com a inserção e remoção de elementos. Em uma Pilha, tanto a inserção de novos elementos quanto a remoção são feitas no topo da estrutura. Desta forma, o último elemento inserido será sempre o primeiro removido.

Como sugestão de implementação, a estrutura de Pilha deve ser implementada inicialmente, possibilitando que definições mais específicas desta estrutura sejam detalhadas em classes mais especializadas. Considerando as características das movimentações do jogo FreeCell, podemos ter Pilhas Intermediarias e Pilhas Definitivas. Além disso, podemos também considerar os Espaços para Movimentação de Cartas como Pilhas Auxiliares, já que a única restrição imposta por estes é o tamanho limite de uma carta. Abaixo temos a ilustração das definições específicas das Pilhas propostas.

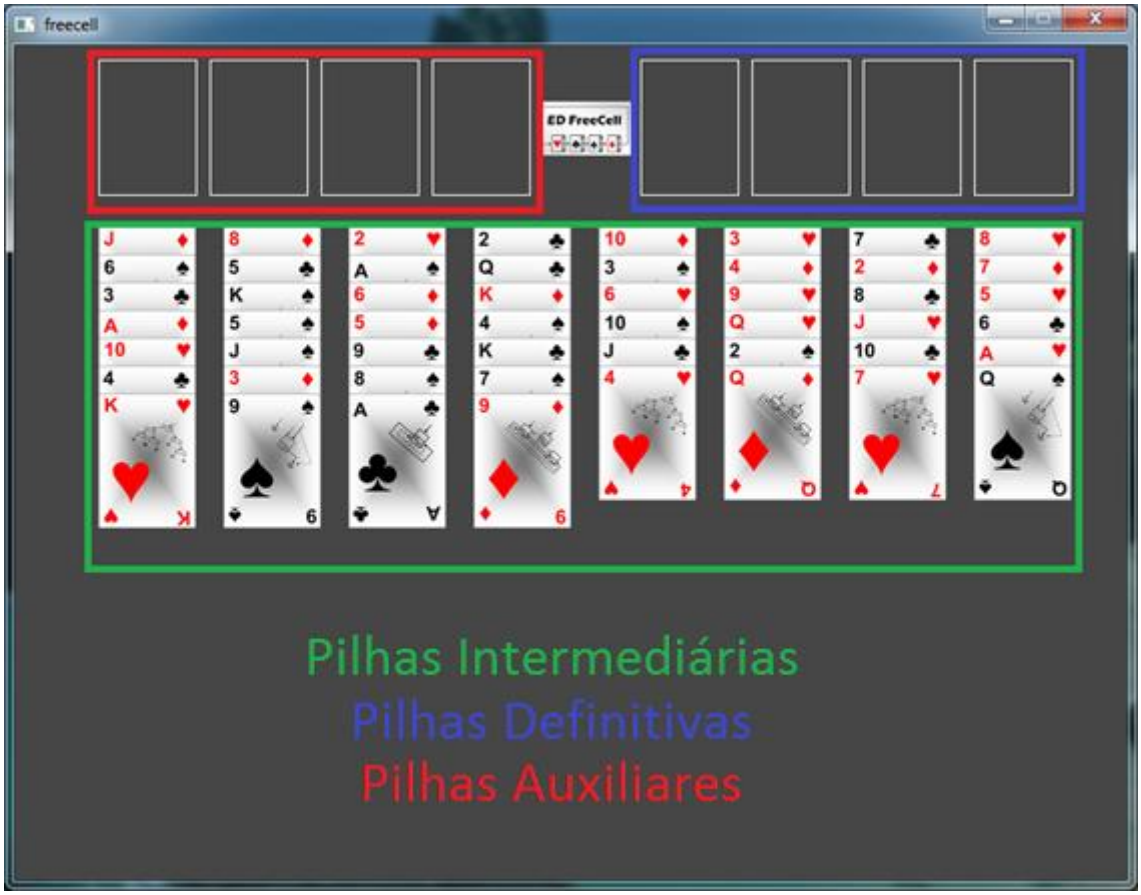

Figura 14 Levantamento de Pilhas com características específicas do jogo FreeCell.

Para cada estrutura específica de Pilha, temos as seguintes características:

- **Pilhas Intermediárias**: Possuem características específicas para inserção. Uma nova carta só pode ser inserida caso sua cor seja oposta e com valor estritamente anterior ao da pilha do topo. Além disso, é possível que mais de uma carta seja movimentada caso existam espaços suficientes para movimentação, considerando espaços vagos nas Pilhas Auxiliares e em outras Pilhas do Jogo. Nesta situação, devem ser verificadas se todas as cartas para movimentação estão devidamente ordenadas, considerando que poderiam ser feitas diversas movimentações equivalentes para inserir todas as cartas desejadas na Pilha destino.

- **Pilhas Definitivas**: Possuem características específicas para inserção. Se a pilha estiver vazia, somente uma carta com valor "Ás" pode ser inserida. Caso já existam cartas na pilha, somente pode ser inserida uma nova carta caso a mesma for do mesmo naipe e, necessariamente, possuir um valor seguinte ao da carta do topo.

- **Pilhas Auxiliares**: O tamanho máximo da Pilha deve ser de uma carta. Desta forma, a única possibilidade de inserção deve ser se a mesma estiver vazia (nenhuma carta na estrutura). Para facilitar a implementação, é dado como sugestão os seguintes procedimentos para movimentação.

1. São removidas *n* cartas da Pilha de origem.
2. As *n* cartas são, momentaneamente, inseridas em uma estrutura auxiliar.
3. Se as *n* cartas puderem ser inseridas na Pilha destino, a movimentação é concluída.
4. Se alguma carta não puder ser movimentada, as *n* cartas inseridas na estrutura temporária são devolvidas à Pilha de origem.

Em relação à organização do código, é sugerido que não sejam misturadas as informações relativas às estruturas utilizadas, lógica do jogo e interface gráfica. Desta forma, a relação entre essas entidades pode ser feita da seguinte forma:

- O jogo simplesmente utiliza as estruturas de Pilha implementadas. Dessa forma, as lógicas de inserção e remoção devem estar contidas na própria definição da estrutura.
- O jogo informa à interface gráfica o estado atual das estruturas (posicionamento das cartas nas estruturas de Pilha).
- A interface gráfica informa ao Jogo a origem e o destino das movimentações entre as estruturas de Pilha.

7.1. Interface do Jogo

Aplicando os conceitos apresentados neste Tutorial, podemos estruturar a interface para o jogo FreeCell.

Para facilitar a implementação, é sugerido que todas as informações gráficas sejam armazenadas em um *buffer*, sendo este, posteriormente, adicionado ao *display*. Neste sentido, podemos pensar em uma quantidade finita de elementos gráficos que, sobrepostos, formam o jogo como um todo.

Incialmente, podemos pensar na mesa do jogo. Esta, por sua vez, é representada por um retângulo preenchido com a cor desejada, assim como já apresentado neste tutorial. Logo, neste momento, podemos utilizar a função "`rectfill`" da Allegro, adicionando a mesa criada ao *buffer*.

```
rectfill(buffer, 0, 0, buffer->w, buffer->h, makeacol(0,255,0));
```

Posteriormente, temos que pensar como adicionar as cartas que estão empilhadas nas Pilhas Intermediárias do jogo. Neste momento, basta carregarmos às imagens referentes a cada carta do jogo, armazenado estas em um buffer "`BITMAP`".

```
BITMAP *asPaus = load_bitmap("imagens\\Paus\\asPaus.bmp", NULL);
```

Agora que já temos as informações visuais relativas às cartas em memória, como adicioná-las na mesa? Como dito anteriormente, podemos pensar nas Pilhas como a sobreposição de imagens de cartas dispostas em cima da mesa.

Pensando na estrutura de Pilha, podemos percorrer cada carta inserida, colocando na mesa a primeira carta inserida na estrutura, seguida pela segunda, terceira e assim por diante. Desta forma, basta utilizarmos a função "`blit`" descrita anteriormente, incrementando, para cada carta da Pilha, o valor do posicionamento da mesma no eixo *Y*. A Figura 15 ilustra a sobreposição de cartas de uma mesma Pilha.

Figura 15. Cartas de uma mesma Pilha sobrepsotas.

Este procedimento deve ser realizado para cada Pilha Intermediária do Jogo, então podemos escrever um procedimento em que deslocamos as cartas de cada Pilha também no eixo *X* (Figura 16). Abaixo segue uma sugestão de implementação desse algoritmo.

```
int posX = 60;   //Posição x estipulada para a primeira carta
int posY = 130;  //Posição y estipulada para a primeira carta
Carta carta;

// Para cada Pilha Intermediária (8)
for (int i=0; i<8 ; i++)
{
  // Se houver cartas na Pilha
  if (!pilhasIntermediarias[i].isVazia())
  {
      // Para cada carta inserida na Pilha
      for (int j=0; j<pilhasIntermediarias[i].getNumeroElementosInseridos() ; j++)
      {
          // Obtém as informações da carta. IMPORTANTE: A carta não é
          // removida da Pilha.
          carta = pilhasIntermediarias[i].getElemento(j);
          // Desenha a carta na posição desejada.
          desenhaCarta(carta, posX + ESPACO_HORIZONTAL*i, ESPACO_VERTICAL*j+ posY);
      }
  }
}
```

Figura 16. Cartas de Pilhas distintas sobrepostas.

Na função "`desenhaCarta`", basta colocarmos a chamada para a função "`blit`", passando como parâmetro o "`BITMAP`" da carta e a posição (*X, Y*) que a esta deve ser desenhada na mesa.

As demais cartas do jogo, presentes nas Pilhas Auxiliares e Definitivas, seguem a mesma lógica de desenho das Pilhas Intermediárias. A única diferença é que, para as Pilhas Auxiliares e Definitivas, não precisamos deslocar no eixo *Y* para desenhar cada carta inserida na estrutura, já que é dada como visualização somente a carta do topo.

Depois de inseridos todos esses elementos em um *buffer*, este é finalmente adicionado ao *display*.

```
blit( buffer, screen, 0, 0, 0, 0, buffer->w, buffer->h );
```

Neste momento, temos todos os elementos do jogo desenhados na mesa! Estamos quase prontos para começar a jogar o FreeCell! Contudo, ainda falta pensarmos como deve ser feita a interação com o usuário.

7.1.1. Detectando movimentação do usuário

É dado como sugestão a utilização do mouse, utilizando como referência as informações dadas neste tutorial. Contudo, é também possível fazer a interface através do teclado.

Considerando a utilizando do mouse, basta verificarmos, no laço principal da Allegro, se o botão esquerdo do mouse foi pressionado. Este procedimento pode ser feito através da verificação da variável global "`mouse_b`".

Caso seja verificado que o usuário pressionou o botão esquerdo do mouse, basta encontrarmos a posição *X* e *Y* do cursor. Neste momento, as variáveis globais "`mouse_x`" e "`mouse_y`" devem ser verificadas.

Como descobrirmos em que Pilha o clique foi disparado? Basta delimitarmos um espaço, considerando os eixos *X* e *Y*, em que cada Pilha deve ocupar na mesa do jogo.

```
// Exemplo para a detecção do clique em uma das Pilhas Intermediárias.
// O espaço está delimitado em um intervalo do eixo X e Y
if((mouse_x>230 && mouse_x<320 && mouse_y>130))
{
   // Verificado o clique dentro deste espaço (Pilha)
   // IMPORTANTE: Aqui devemos gravar a informação de que o usuário clicou nesta
Pilha
   rectfill(buffer, 230, 120, 320, TAMANHO_VERTICAL-5, COR_SELECAO);
}
```

Vejam que, para facilitar ainda mais a experiência do usuário no jogo, desenhamos um retângulo abrangendo todo o espaço delimitado pela pilha. Desta forma, fica claro qual Pilha foi feito o clique (imagem do jogo em execução abaixo).

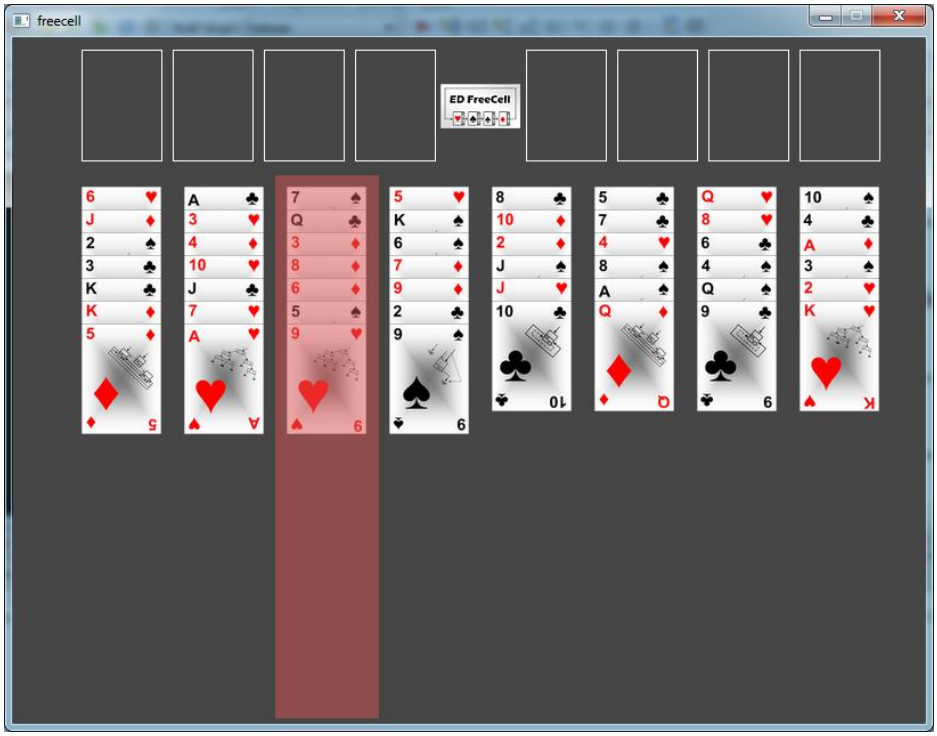

Figura 17. Exemplo de movimentação do usuário.

Da mesma forma que aos demais elementos gráficos colocados na mesa do jogo, o retângulo desenhado para reforçar a movimentação também é adicionado ao *buffer*.

Pronto! Desta forma já conseguimos detectar qual a Pilha foi selecionada pelo jogador.

Como a movimentação de cartas possuí uma origem e um destino, basta aplicarmos a detecção para ambas situações. Logo, tendo um movimento de origem e destino, podemos

passar a informação da Interface Gráfica para a lógica do jogo, estando esta responsável por continuar o resto do processamento das movimentações.

Descarregue o ED FreeCell:
http://edcomjogos.dc.ufscar.br/tutoriais/FreeCell.rar
executável, imagens e .dll (se já não estiver, coloque a .dll no diretório "Window\System32\").