INSTITUTO FEDERAL MINAS GERAIS - *Campus Bambuí*
Lazarus
Prof. Marcos Roberto Ribeiro

Lista de Exercícios 05

Observações:

- Todos os formulários dos aplicativos devem ser desenhados;
- Todas as propriedades alteradas devem ser indicadas;
- Use alinhamento relativo sempre que possível.

---

1. Refaça os seguintes exemplos da aula sobre "Componentes Avançados e Criação de Interfaces no Lazarus":

   a) Exemplo com Image

   b) Exemplo com StringGrid e SpinEdit

   c) Exemplo com Dialogs

   d) Exemplo com Timer

2. Explique para que é utilizado e dê um exemplo prático para cada um dos componentes abaixo:

   a) *Image*;

   b) *StringGrid*;

   c) *Dialogs*;

   d) *SpinEdit*;

   e) *Splitter*;

3. Dê exemplos práticos para a utilização das propriedades do componente *Image*.

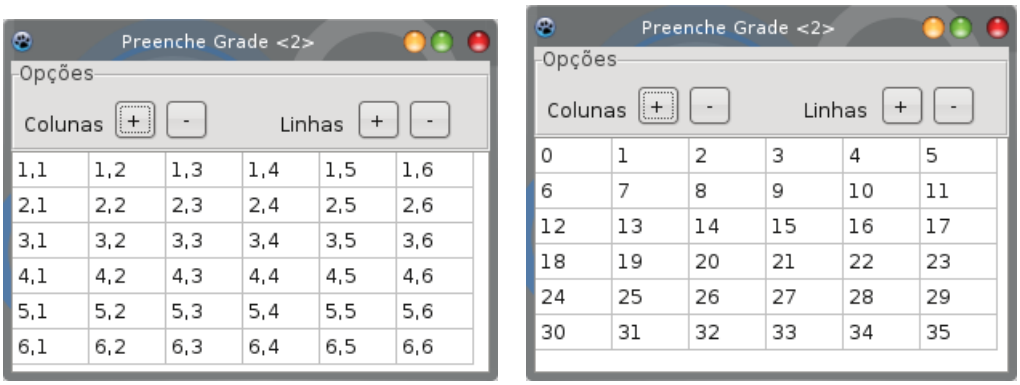
4. Crie uma aplicação no Lazarus capaz de exibir 5 figuras armazenadas em arquivo, use botões para navegar entre as figuras. Dicas:

   - Salve as cinco figuras na mesma pasta do projeto com nomes de 1 a 5;
   - Use o código `ExtractFilePath(Application.ExeName)` descobrir a pasta do programa;
   - Use o método *LoadFromFile* da propriedade *Picture*.

5. Dê exemplos práticos para a utilização das propriedades e eventos do componente *StringGrid*.

6. Implemente um programa no Lazarus como mostrado na Figura 1(a). Os botões "+" e "-" devem controlar o número de colunas e linhas, sendo que deve haver pelo menos uma linha e uma coluna. Cada célula da tabela deve exibir sua linha e coluna, isto deve ser feito na na inicialização do programa e sempre que houver alterações no número de linhas ou colunas.

7. Modifique o programa do exercício anterior para que as células da tabela exibam uma sequência numérica como mostrado na Figura 1(b).



(a) Versão 1. (b) Versão 2.

Figura 1: Programa Preenche Grade.

8. Crie um programa no Lazarus como mostrado na Figura 2. O *SpinEdit* define o número de valores presentes na grade. O total dos valores deve ser calculado sempre que houver uma alteração na grade (nos valores ou nos *checks*) ou no *SpinEdit*. O total deve considerar apenas os valores com os *checks* marcados. Números digitados incorretos não devem ser considerados.

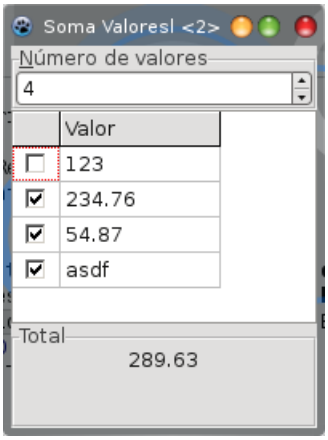
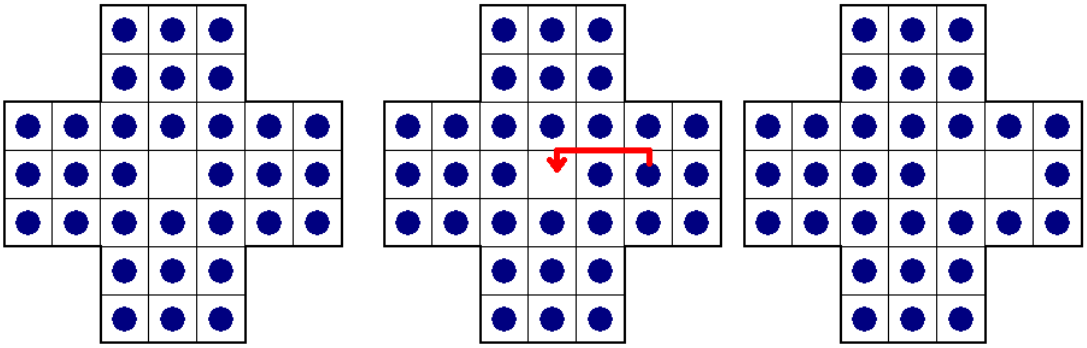


Figura 2: Programa Soma Valores.

9. Crie um programa no Lazarus para receber e validar a data de nascimento do usuário. Utilize 3 *SpinEdits* para receber dia, mês e ano. A validação deve ser feita para que o usuário informe os meses de 1 a 12, dias de 1 a 31, sendo que o programa não deve permitir um número maior de dias para o mês selecionado. No mês de fevereiro, permita no máximo 29 dias.

10. Desenvolva um programa no lazarus capaz de visualizar figuras. O programa deve ter um botão "Abrir" que mostra uma caixa de diálogo onde o usuário seleciona a figura a ser aberta.

11. Implemente o jogo "Resta 1" no Lazarus. O jogo Resta 1 é jogado em um tabuleiro como o mostrado na Figura 3(a). O objetivo do jogo é eliminar as peças de forma que reste apenas uma. A peças são eliminadas pulando uma sobre a outra nas direções vertical e horizontal, como mostra a Figura 3(b).



(a) Tabuleiro. (b) Jogada.

Figura 3: Jogo Resta 1.

Para implementar tal jogo vamos utilizar uma grade para representar o tabuleiro, obedecendo aos seguintes passos:

(a) Projete um formulário como mostrado na Figura 4(a). O botão "Sair" deve fechar o formulário;

(b) Crie um procedimento **NovoJogo()** para inicializar o tabuleiro como mostrado na Figura 4(b). Este procedimento deve ser chamado quando o programa iniciar e pelo botão "Novo Jogo". A peças são representadas com "O" (o maiúsculo) e a posição central fica com um "." (ponto) indicando que está vazia. O procedimento deve alterar o **Caption** do formulário para "Resta 1 (Jogando)". Dica: Preencha as três colunas, as três linhas e depois o ponto central;

(a) Formulário. (b) Novo Jogo.

Figura 4: Programa Jogo Resta 1.

(c) Para realizar uma jogada, o usuário precisa clicar em uma posição inicial e em uma posição final. Portanto, a cada clique, é preciso saber se a jogada está começando ou terminando. Para fazer isto, declare as variáveis globais **JogadaIniciada**, **LinIni** e **ColIni**.

(d) Modifique o procedimento **NovoJogo()** para que **JogadaIniciada** seja **false**, pois, quando o jogo começa, o usuário não iniciou nenhuma jogada;

(e) Crie o procedimento **Jogada(L, C: Integer)** para exibir apenas uma mensagem dizendo se a jogada esta começando ou terminando (por enquanto, não precisa mexer com as peças). Dica: utilize a variável global **JogadaIniciada**;

(f) Codifique o evento necessário para que o procedimento **Jogada()** seja chamado quando o usuário clicar na grade. Passe **GridResta1.Selection.Top** como parâmetro para **L** e **GridResta1.Selection.Left** como parâmetro para **C**;

(g) Modifique o procedimento **Jogada()** considerando os cliques do usuário, ou seja, quando houver um clique em uma célula com "O", o procedimento deve:

- Mudar **JogadaIniciada** para **True**;
- Armazenar a linha e a coluna clicadas em **LinIni** e **ColIni**, respectivamente.

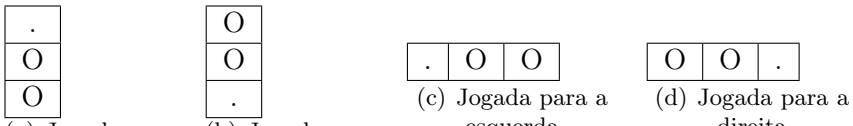
(h) Crie um procedimento **JogadaTermina(L, C: Integer)** que servirá para finalizar uma jogada.

(i) Modifique novamente o procedimento **Jogada()**. Se há um clique sobre "O", uma jogada começa (isto já foi implementado). Se o clique não for sobre "O", é preciso fazer alguns testes:

- Se o clique não for sobre "O", é preciso testar se **JogadaIniciada** é **True**;
- Se **JogadaIniciada** for **True**, então **JogadaIniciada** deve receber **False**;
- Depois de mudar **JogadaIniciada** para **False**, é preciso testar se o clique foi sobre "." e, em caso afirmativo, chamar o procedimento **JogadaTermina()**;

(j) Agora, modifique o procedimento **JogadaTermina()** para finalizar uma jogada corretamente. Uma jogada pode ser finalizada se a posição original possui peça, a posição de destino está vazia e a posição do meio possui peça. Existem quatro possibilidades:

- Jogada para cima: **C = ColIni**, **L = LinIni - 2**, mostrado na Figura 5(a);
- Jogada para baixo: **C = ColIni**, **L = LinIni + 2**, mostrado na Figura 5(b);
- Jogada para a esquerda: **L = LinIni**, **C = ColIni - 2**, mostrado na Figura 5(c);
- Jogada para a direita: **L = LinIni**, **C = ColIni + 2**, mostrado na Figura 5(d).



(a) Jogada para cima (b) Jogada para baixo (c) Jogada para a esquerda (d) Jogada para a direita

Figura 5: Possibilidades de jogadas no jogo Resta 1.

Além do posicionamento, é preciso verificar as peças de cada posição. Isto é, posição do meio preenchida ("O") e posição final vazia (".") (por que não testar a posição inicial?). Dica: É possível verificar apenas as posições e depois verificar as peças.

Após a conclusão da jogada, as posições de origem e do meio devem ficar vazias e a posição de destino preenchida.

(k) Crie a função **FimJogo(): Boolean** para verificar se o jogo terminou. Esta função deve percorrer a grade e verificar se ainda existem jogadas a serem realizadas. Se houverem possíveis jogadas a função deve retornar **false**, senão deve retornar **true**. Dica: percorra a grade da segunda até a penúltima linha (para jogadas verticais) e da segunda até a penúltima coluna (para jogadas horizontais);

(l) Modifique o procedimento **Jogada()** chamando a função **FimJogo()** no seu final. Se **FimJogo()** retornar **true** mostre uma mensagem para o usuário mostrando que o jogo terminou e modifique o **Caption** do formulário para "Resta 1 (Fim de jogo)";

(m) Crie a função **ContaPecas(): Integer** que conte o número de peças na grade. Modifique novamente o procedimento **Jogada()** para que sejam exibidas o número de peças restantes no final do jogo.