Brinkando v0.0.2

Uma passagem pelo código da v0.0.2

```
import gi
```

Importa a biblioteca GI para o projeto.

```
gi.require_version("Gtk","3.0")
from gi.repository import Gtk
```

Importa e define o uso da versão 3.0 do GTK (a biblioteca principal do projeto, a que de fato traz os recursos para criação de telas).

```
class MyWindow(Gtk.Window):
    def __init__(self):
        super().__init__(title="BrinkAnDo v0.0.2")
        self.set_default_size(1280, 720)

        .
        .
        .
```

Esse trecho define uma classe arbitrariamente chamada de MyWindow, que é baseada na classe “Window”, do GTK. De início, é definido o método __init__, executado quando a classe é instanciada (ou seja, executado quando é criado um objeto a partir dessa classe; leia-se objeto, nesse contexto, como uma nova tela).

O método super() é usado para referenciarmos a classe mãe (a classe na qual a nossa se baseia, ou seja, a Window, do GTK). A partir disso, ainda na linha onde se usa super(), é chamado o __init__ da classe mãe, que recebe a informação do título da janela (title=”…”).

Na sequência, define-se o tamanho padrão da janela (informando primeiro a largura, e depois a altura). É preciso notar que ao definir/modificar propriedades da janela (um objeto específico a ser criado a partir da nossa classe), usa-se *self*. Isso é uma referência ao objeto, que representa a nossa janela.

```
        # base grid for layout
        self.grid = Gtk.Grid.new()
```

```
        self.grid.set_row_spacing(4)
        self.grid.set_column_spacing(4)
```

Definimos uma variável dentro do objeto (por isso escreve-se self.*nomeDaVariavel*) chamada *grid*. Essa variável recebe, da biblioteca GTK, um novo objeto da classe Grid. Na sequência, definimos algumas propriedades desse novo grid (espaçamento entre linhas e entre colunas igual a 4).

```
        # a grid which will hold the selected commands and other items
        # currently meant to be an item of each one of the menu buttons
        self.cmdGrid = Gtk.Grid.new()
        self.cmdGrid.set_row_spacing(4)
        self.cmdGrid.set_column_spacing(4)
```

Semelhantemente, aqui é definida outra variável, chamada de *cmdGrid*, que também recebe um objeto da classe Grid, do GTK. São definidos então os espaçamentos entre linhas e entre colunas.

```
        self.grid.attach(self.cmdGrid, 0, 0, 10, 2)
```

Para a variável *grid*, pertencente ao objeto que representa nossa tela, é usado o método attach. Segundo a documentação do GI:

attach(*child*, *left*, *top*, *width*, *height*) [source]

Parameters

- **child** (Gtk.Widget) – the widget to add
- **left** (int) – the column number to attach the left side of **child** to
- **top** (int) – the row number to attach the top side of **child** to
- **width** (int) – the number of columns that **child** will span
- **height** (int) – the number of rows that **child** will span

Adds a widget to the grid.

The position of **child** is determined by **left** and **top**. The number of "cells" that **child** will occupy is determined by **width** and **height**.

O método attach recebe 5 parâmetros:

- *child*: o elemento que passará a ser filho do elemento base na estrutura. O elemento base é nesse caso o nosso *grid*. Isso quer dizer que *child* estará **dentro** de *grid*.

- *left*: o posicionamento de um elemento dentro de um grid é definido pelo número da coluna e pelo número da linha na qual se localiza, como x e y num plano cartesiano, com a diferença de que o elemento não é necessariamente um ponto; tem dimensões a serem definidas. *left* representa o número da coluna onde se localiza o lado esquerdo do elemento.
- *top*: semelhantemente, esse parâmetro representa o número da linha onde se localiza o topo do elemento.
- *width*: representa a largura, em número de colunas, do elemento. Usamos *left* para definir onde o elemento começa, da esquerda para a direita, e com *width*, definimos o quão largo será, no mesmo sentido.
- *height*: representa a altura, em linhas, do elemento.

No código da aplicação, adicionamos *cmdGrid* dentro de *grid*, definindo suas dimensões: começa em (0, 0) e cresce 10 colunas para a direita e 2 linhas para baixo.

```
        # the box which will hold the selected commands
        self.cmdBox = Gtk.FlowBox()
        self.cmdBox.set_valign(Gtk.Align.START)
        self.cmdBox.set_max_children_per_line(10)
        self.cmdBox.set_min_children_per_line(10)
        self.cmdBox.set_selection_mode(Gtk.SelectionMode.NONE)
```

Cria-se outra variável, *cmdBox*, destinada a conter os comandos selecionados pelo usuário. Ela recebe um FlowBox. Na sequência, são definidas algumas propriedades: alinhamento vertical (no início da caixa que o contiver), máximo de elementos filhos por linha (nesse caso setado para 10), mínimo de elementos filhos por linha (também setado para 10) e modo de seleção (setado para nenhum).

O flowbox é uma caixa destinada a conter vários elementos dentro de si, que se organizarão num fluxo, que pode ser controlado pela pessoa desenvolvedora. Por exemplo, pode ser um fluxo horizontal, onde a caixa vai dispondo os elementos um ao lado do outro até que atinja o limite lateral, onde pode haver uma quebra de linha, e o restante dos elementos formarão uma linha logo abaixo.

```
        # a cute little car icon
        cmdCarIcon = Gtk.Image.new_from_file("assets/little-car.png")
        self.cmdGrid.attach(cmdCarIcon, 0, 0, 1, 2)
```

Da classe Image, do GTK, é criado um novo objeto a partir de uma imagem disponível na pasta assets do projeto. Esse objeto contendo a imagem é então adicionado ao *cmdGrid*, com posição e tamanho definidos.

```
        self.cmdGrid.attach_next_to(self.cmdBox, cmdCarIcon, 1, 9, 2)
```

Segundo a documentação do GI:

attach_next_to(*child*, *sibling*, *side*, *width*, *height*) [source]

Parameters

- **child** (Gtk.Widget) – the widget to add
- **sibling** (Gtk.Widget or None) – the child of **self** that **child** will be placed next to, or None to place **child** at the beginning or end
- **side** (Gtk.PositionType) – the side of **sibling** that **child** is positioned next to
- **width** (int) – the number of columns that **child** will span
- **height** (int) – the number of rows that **child** will span

Adds a widget to the grid.

The widget is placed next to **sibling**, on the side determined by **side**. When **sibling** is None, the widget is placed in row (for left or right placement) or column 0 (for top or bottom placement), at the end indicated by **side**.

Attaching widgets labeled [1], [2], [3] with **sibling** == None and **side** == Gtk.PositionType.LEFT yields a layout of '3 [2]'[1].

O método attach_next_to posiciona um elemento próximo a um outro, fazendo com que este seja sua referência permanente de posicionamento. Com isso, a linha de código acima posiciona *cmdBox* próximo a *cmdCarIcon* (a variável que contém o objeto Gtk.Image com o ícone mencionado anteriormente). Especificamente, *cmdBox* é posicionado ao lado 1 (direito) de *cmdCarIcon*, tomando 9 colunas e 2 linhas de tamanho.

Da documentação do GI:

class **Gtk.PositionType**(*value*)

Bases: GObject.GEnum

Describes which edge of a widget a certain feature is positioned at, e.g. the tabs of a Gtk.Notebook, the handle of a Gtk.HandleBox or the label of a Gtk.Scale.

LEFT = 0

The feature is at the left edge.

RIGHT = 1

The feature is at the right edge.

TOP = 2

The feature is at the top edge.

BOTTOM = 3

The feature is at the bottom edge.

```
        # the box which will hold the command options (buttons)
        self.cmdFlowbox = Gtk.FlowBox()
        self.cmdFlowbox.set_valign(Gtk.Align.END)
        self.cmdFlowbox.set_max_children_per_line(2)
        self.cmdFlowbox.set_min_children_per_line(2)
        self.cmdFlowbox.set_selection_mode(Gtk.SelectionMode.NONE)
```

Esse trecho é semelhante ao que define *cmdBox*. Um novo flowbox é criado, recebendo o nome de *cmdFlowbox*, destinado a conter os comandos dentre os quais o usuário poderá escolher. Algumas propriedades são definidas: alinhamento vertical ao fim da caixa que o contiver, máximo e mínimo de elementos filhos por linha igual a 2 e sem modo de seleção.

```
        # each of the command buttons
        self.leftBttn = Gtk.Button()
        self.leftBttn.set_image(Gtk.Image.new_from_file('assets/turn-left-arrow.png'))
        self.cmdFlowbox.add(self.leftBttn)

        self.rightBttn = Gtk.Button()
        self.rightBttn.set_image(Gtk.Image.new_from_file('assets/turn-right-arrow.png'))
        self.cmdFlowbox.add(self.rightBttn)
```

```
        self.uTurnBttn = Gtk.Button()
        self.uTurnBttn.set_image(Gtk.Image.new_from_file('assets/u-turn-arrow.png'))
        self.cmdFlowbox.add(self.uTurnBttn)

        self.forwardBttn = Gtk.Button()
        self.forwardBttn.set_image(Gtk.Image.new_from_file('assets/forward-arrow.png'))
        self.cmdFlowbox.add(self.forwardBttn)
```

Aqui, são criadas quatro variáveis, todas elas sendo objetos da classe Button, do GTK, e cada qual contendo uma imagem diferente. Essas imagens são setas apontando para diferentes sentidos, identificando os comandos possíveis. Além disso, cada um dos botões é adicionado ao *cmdFlowbox*.

```
        # fill the cmdBox with drop-down menus
        for i in range(20):
            self.cmdSelect = Gtk.MenuButton()

            menu = Gtk.Menu()
            menu.attach(self.cmdFlowbox, 0, 0, 0, 0)

            self.cmdSelect.set_popup(menu)

            self.cmdBox.add(self.cmdSelect)
```

Esse *for loop* é responsável por, a cada iteração, criar um objeto da classe MenuButton, do GTK, e associá-lo a um outro objeto Menu, que contém um *cmdFlowbox* (definido anteriormente). Dessa forma, quando o elemento botão for pressionado, exibirá num pop-up o elemento Menu, com aquele flowbox que contém todos os comandos dentre os quais o usuário pode escolher. Por fim, esse botão é adicionado ao *cmdBox*. Esse conjunto de instruções é repetidamente executado até atingir 20 execuções.

```
        self.add(self.grid)
```

Por fim, o *grid*, elemento que contém todos os demais, é adicionado ao *self* (o objeto que representa a janela).

```
win = MyWindow()
win.connect("destroy", Gtk.main_quit)
win.show_all()
Gtk.main()
```

Nesse trecho, têm-se as seguintes ações:

- Criação de fato de um objeto baseado na classe que definimos até aqui. Ou seja, criação de uma janela.
- Para a janela criada, é assicioado o sinal destroy com a função main_quit. Isso torna possível fechar a janela ao clicar no tradicional botão “X” ao seu topo. Esse botão, quando clicado, envia o sinal reconhecido como “destroy”, enquanto a main_quit é a função que gerencia o fechamento da janela.
- A janela é exibida na tela.
- Uma chamada é feita à função main do GTK. Isso faz com que o loop principal seja executado e permaneça em execução até que haja uma chamada à função main_quit.

Referências

- Documentação da API do PyGObject, seção dedicada às classes: Classes - Gtk 3.0 (lazka.github.io)
- Documentação da API do PyGObject, seção dedicada às funções: Functions - Gtk 3.0 (lazka.github.io)