Simple-Tk-template

Preface

This is a modern template to revise the original GUI of python **tkinter**.

This template is mainly for developing full-screen app with a dashboard in right side, a main window in left side.

Each python module inside stand for different function, I will list the following things in each modules:

1. Full path
2. Modules imported in internal
3. Classes and functions

Project Structure

```
PROJECT_ROOT
├── Tkinter_template
│   ├── Assets/
│   │   ├── default_menu.py
│   │   ├── default_dashboard.py
│   │   ├── extend_widget.py
│   │   ├── font.py
│   │   ├── image.py
│   │   ├── music.py
│   │   ├── project_management.py
│   │   ├── soundeffect.py
│   │   └── universal.py
│   └── base.py
├── musicplayer/
├── README.md
└── LICENSE
```

Demo Project Structure

Using this packages, your project should be like this one:

```
PROJECT_ROOT
├── images/
│   ├── bitmaps/
│   │   └── your_file
│   └── covers
│   │   └── your_file
```

```
├── sounds/
│   └── your_file
├── musics/
│   └── your_file
├── detas/
│   └── your_file
├── modules/
│   └── your_file
└── main.py
```

Modules

`base.py`

Mainly for skeleton built and define the template

Modules use: project_management

```
class Interface(title, icon=None, default_menu=False)
```

- **root**: root widget
- **side**: root widget side (Computer screen side)
- **isFullscreen**: (property) getter, return the state of fullscreen; setter, set the state of fullscreen and do fullscreen setting by same time
- **canvas**: main canvas widget (left side)
- **canvas_side**: main canvas widget side
- **top_menu**: the main menu bar (that on top and horizontal) (all menu command, cascade should pad on it)
- **default_menu**: the default menu connected to top_menu and it is a cascade that hava name Selective Function which provide a menu to pad on some default_menu.py method, but, you do not need to pad on by own.
    - If True the canvas and dashboard height will -20 otherwise -0, so if do not want dafault menu and want add other menu on top_menu remenber to make height -20 if you did not some function will malfunction
- **dashboard**: frame widget (right side)
- **dashboard_side**: frame widget side

```
app = Interface("foo")
app.dashboard['height'] = app.dashboard['height'] - 20
app.dashboard_side = int(app.dashboard['width']), int(app.dashboard['height'])
app.canvas['height'] = app.canvas['height'] - 20
app.canvas_side = int(app.canvas['width']), int(app.canvas['height'])
```

`Assets/default_dashboard.py`

Mainly for well-built function to pad on dashboard

Module used: project_management, font, image, universal, extend_widget

```
time_show(dashboard, side)
```

passing the dashboard and its side, adding the time in form **00:00:00**. using this function can adjust the show string var by `time_show` variable
need to using time_flush function

```
time_show(dashboard, side)
```

passing the dashboard and its side, adding the date in form **2022/09/09 (Mon)**. using this function can adjust the show string var by `date_show` variable
need to using time_flush function

```
time_flush()
```

update the date and time value
should put in while loop

```
table(canvas, dashboard, side)
```

pad on a small button on the dashboard at lower-right place. Its function is show the window only cover in the canvas. (include press and hover version)

```
class MusicPlayer(dashboard, music_obj)
```

package the Music class in music modules(below) into a more fancy function class

if you want to use this class, download the image source `musicplayer` folder in my github and put it in your project's `images` folder

- **d**: return passed parameter dashboard
- **dw**: return passed parameter dashboard's width
- **fs**: return a integer represent the font size internal use
- **mo**: return passed parameter music_obj, it should be a Music class object in music module
- **play_a_music(self, file)**: pass a mp3 file in path `musics`, and play a music, but seldom use in yourself
- **set_ball(self)**: configure the ball in duration bar
  should be put in loop

`Assets/default_menu.py`

Mainly for well-built function to pad on menu, like defualt_menu

Module used: project_management, font, image

```
background_color(canvas)
```

adjust the canvas color by scale and button

```
menu.add_command(label="foo", command=lambda: background_color(canvas))
```

```
canvas_cover(canvas, canvas_side, side=None)
```

access the cover from path `images\covers`, select a cover to pad on canvas (parameter canvas) and the image size is specified by parameter canvas_side if parameter side is not provided, using canvas_side alternatively, it control pop up window size *internal call the select_cover method of project_management

```
time_flush()
```

update the date and time value
should put in while loop

`Assets/extend_widget.py`

Mainly for some widget that basic on tk widget, add some features on them

Module used: None.

```
class BindButton(char, root=None, **option)
```

for a button that can control it using a key binding,
like

```
btn = BindButton('Return', app.root)
```

can using enter key to control

- **char**: return char passed

```
class EffectButton(color, root=None, **option)
```

for a button that achieve hover features, when mouse is on it, it will change its bg and fg to the color passed *parameter color should be a tuple (bg, fg)*

- **color**: return color passed

Assets/font.py

Mainly for font relate manipulations.

Module used: None.

```
change_font(font)
```

change the main font (global variable) for your project

```
font_get(size, bold=False)
```

return tuple in form **(font_family, size, bold|None)**

```
font_span(text, fit_size, *, upper_bound=1000)
```

give a text and fit size return just can fit the width font size
beacuse the font size only decided by width, the height may be too large to exceed your expect. passing a upper bound font size to ensure the font size not go so big (upper bound value can evaluate height * 3/4)

```
measure(text, size)
```

give a text and font size, return the width will have

Assets/image.py

Mainly for image showing, particularly tk image. Because the original tkinter tk image has some bugs (image object will be buffer out if these source not store in some container), this modules also avoid this bug

```
```

```
tk_image(filename, width=None, height=None, *, dirpath=None,
get_object_only=False)
```

passing filename(only file not include path), size(width, height). If the image file path in search_path, don't need to specify dirpath, if not, you need. (filename is only file name whenever its path include or not include in search_path, if not include, its path should be passed to dirpath) return a tkImage object that can be directly used in `create_image`, etc.

```
mycanvas.create_image(0, 0, anchor='nw', image=tk_image(
    'image.png', 64, 64, dirpath='images\\main'
))
```

but if the parameter get_object_only is True, return TkImage object only (this class is in below)

```
class TkImage(whole_name, width, height)
```

less to use for general project
to store, manage image sources from calling function

tk_image

- **image_base**: a dictionary in the form `{TkImage objects(this class objects):Image.TkImage objects}`
- **whole_name**: return the image's whole path name
- **width**: return the image's width
- **height**: return the image's height
- **get_image(self)**: return the image's TkImage object source

`Assets/music.py`

Mainly for BGM aka background music manipulations.

Module used: None.

```
class Music()
```

to build a music class combind a lot of function for music

- **music**: (property) getter, return the music filename include extension; setter, pass a music name (the music file should be in musics folder) and play it, pass None can unload music
- **pause**: (property) getter, return the state of pause (a boolean value); setter, pass a boolean value to pause or unpause

```
Music().pause = True # will pause music
```

- **toggle(self)**: switch the state of pause the internal code

```
self.pause = not (self.pause)
```

- **set_volume(self, volume)**: pass a float or integer number to set the volume of music
- **judge(self)**: it will check that if not pause and the music is end, replay the music
  need to put in loop

`Assets/project_management.py`

Mainly for common use function in many project, using it to save time.

Modules use: font, image

```
create_menu(root)
```

create a menu and using the same style, form: `Menu(root, font=font_get(16), tearoff=0)`

```
new_window(title, icon=None, maxsize=None)
```

pop up a new window with the title specified by title parameter, icon specified by icon parameter and maxsize specified by maxsize parameter, it is also a tuple `(width, height)`
the window will be zoomed but not fullscreen

```
making_widget(widget)
```

return widget corresponde to widget name in tkinter, example:

```
making_widget("Canvas")(self.root, width=100, height=100)
```

so the function's purpose is to avoid too many python file

```
import tkinter module lose efficiency
```

```
canvas_obj_states(canvas, mode, *tag)
```

select a canvas and mode mode can be (hidden, normal delete), if a object in the canvas contain the tag specified by *tag, nothing happened to it, on the contrary if contain any tag in *tags depend on mode

- **delete mode**: delete it
- **normal mode**: make the state to normal
- **if `H` in tags, it will not be show, like pernament hidden**
- **hidden mode**: make the state to hidden
  example:

```
canvas_obj_states(self.canvas, 'hidden', 'cover')
# using this function, only object contain cover tag will
# keep same, other objects' state will be hidden
```

```
select_cover(canvas, side, file)
```

using file (only filename) to be cover, pad on canvas, and the size depends on side parameter

```
canvas_reduction(canvas, canvas_side, music_obj, cover=None, music=None)
```

to like regenerate the canvas (most use in changing the tab, like from main page change to setting page can use this function), here is the procedures:

1. delete all objects on the canvas parameter
2. move to the original point (sometime width or height may be move with scrollbar)
3. if the music_obj parameter is not False, music set to None and if some music parameter specified play the music use given music_obj (`music_obj.music = music`)
4. cover padded if cover parameter is gived, internal using this module function *select_cover*
5. unbind a sequence of common keys event include mouse wheel, button-1, double-button-1, return and space.
   if you using others key bind events, you can add corresponding unbind function in this function.
   (because I didn't find a function in tkinter that can unbind all key event)

`Assets/soundeffect.py`

Mainly for sound effects, such like a short sound play.

Modules use: universal

```
```

```
play_sound(filename)
```

play a soundeffect, and filename parameter only comtain file name and do not include extension, the search path base is `sounds` so filename don't add `sounds`.
example: if want to play the soundeffect with path `sounds/mainsound/boom.mp3` using this

```
play_sound('mainsound/boom') #play the sound
```

`Assets/universal.py`

Mainly for some common use function and not include canvas manipulations

Modules use: project_management, image

```
delete_extensio(filename)
```

remove the file extension, internal code is:

```
return filename[:filename.rfind('.')]
```

```
str_tuple_date_change(arg)
```

pass a data format, it can be

1. tuple in form (YYYY, MM, DD), change to string YYYY-MM-DD
2. string in form YYYY-MM-DD, change to tuple (YYYY, MM, DD)
3. datatime.data object in form date(YYYY, MM, DD) change to string YYYY-MM-DD

```
parse_json_to_property(app, json_file)
```

passing app parameter that usually is your main project example:

```
from Tkinter_template.base import Interface
from Tkinter_template.Assets.universal import parse_json_to_property as pjtp
if __name__ == '_main__':
    main = Interface('foo', 'bar.ico')
    main.pjtp(main, 'datas/setting.json')
```

it will make app parameter adding some attributes, the json_file parameter is the json file path you want to parse
each attribute will point to a Var class in original tkinter and the type of Var denpends on its value in json
example:

```
{
    volume: 0.8,
    fullscreen: true
}
```

```
parse_json_to_property(app, json_file)
```

passing app parameter that usually is your main project example:

using this function, app.volume = DoubleVar(value=0.8), app.fullscreen = BooleanVar(value=True)
int -> IntVar()
float -> DoubleVar()
str -> StringVar()
boolean -> BooleanVar() and if value is array, it will be flatten to a sequence of value (only support single layer), each attribute will name to `f{key}_{count}` count starts to 0
example:

```
{
    images: ["1.jpg", "2.jpg"]
}
```

using this function app.images_0 = StringVar(value='1.jpg'), app.images_1 = StringVar(value='2.jpg')
if value is object, not support

```
MoveBg(canvas, canvas_side, rate, source_folder, abadon_folder: tuple)
```

achieve move background function which the move is downeard, passing canvas parameter (add movebg effect on it), canvas_side parameter (the same canvas side), rate parameter (to control move bg size), source_folder parameter (the source of move bg images, will be os.walk to extract image file), abadon_folder (the source do not use to move bg images, must be list or tuple)

- **c**: return canvas passed
- **cs**: return canvas_side passed
- **r**: return rate passed
- **imgName**: contain tuple: (whole_path and not include filename, filename) it controled by parameter source_folder and abadon_folder
- **create_obj(self, number=1)**: create move bg instance in the number given by number parameter
- **flush(self)**: use this to animate move bg
  should be put in loop

LICENSE

LICENSE