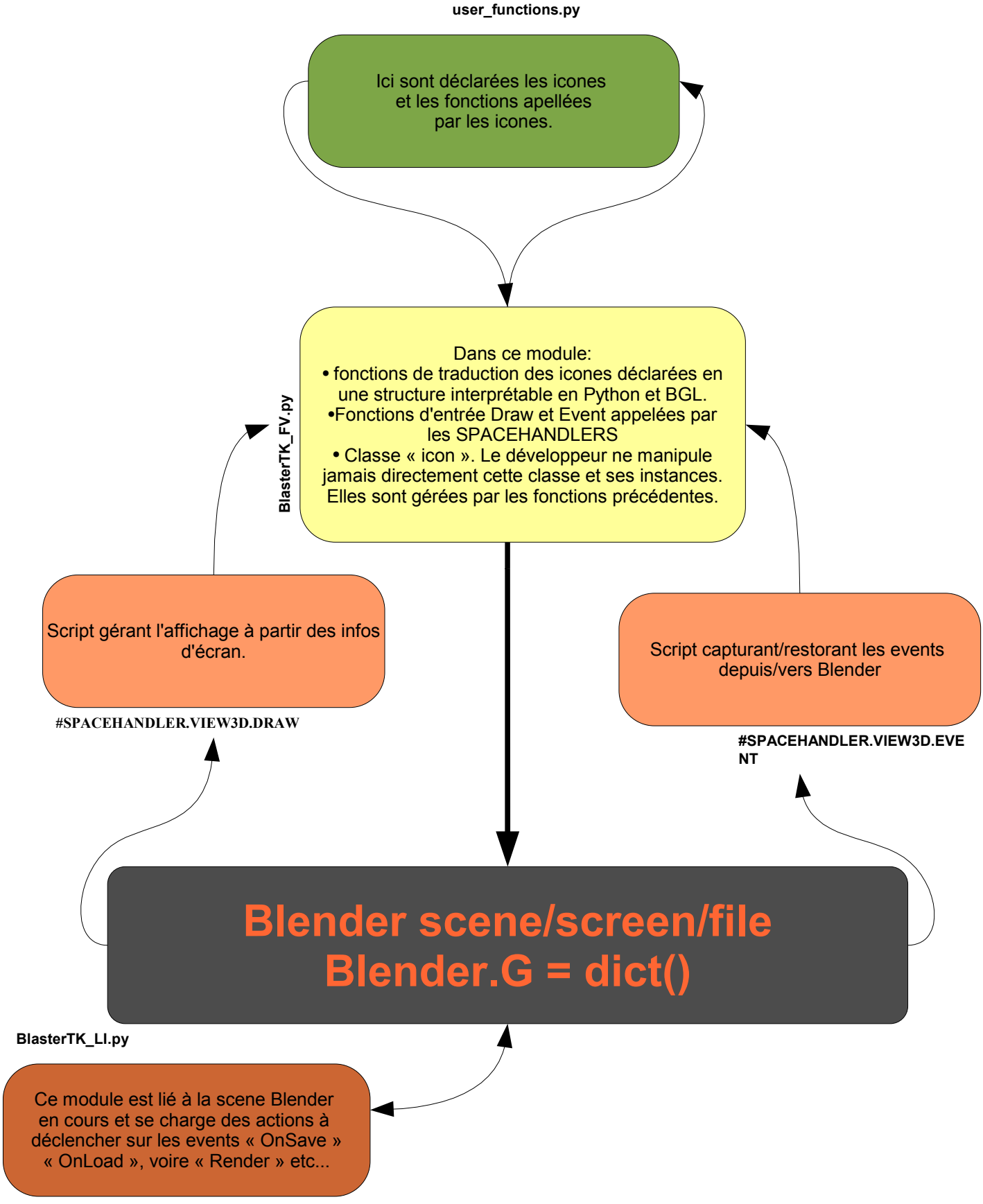
BlasterTK dev
DOCUMENTATION

Daniel Barbeau Vasquez

BlasterTK : Fonctionnement

BlasterTK est un ensemble de modules Python pour Blender qui permet de créer facilement des icones et de les placer dans la Vue 3D. Ces modules sont:

- BlasterTK_FV.py
- BlasterTK_LI.py
- user_functions.py



1. Créer une nouvelle icone.

Pour créer une nouvelle icone, il faut:

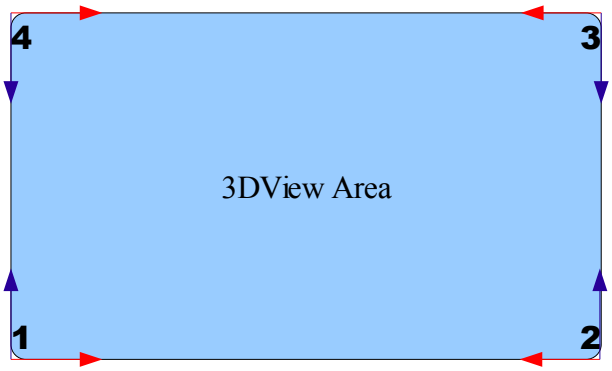
- a: Connaître le nombre de positions de l'icone (unique, binary, ternary ...),
- b: Créer autant d'images que de positions,
- c: Savoir l'emplacement de l'icone dans la vue 3D,
- d: Savoir si l'icone est liée à l'objet séléctionné.
- e: Est-ce que l'icone autorise un clic droit
- f: faut-il sauvegarder l'état de l'icone si celle-ci est masquée?
- g: Préparer les fonctions associées aux différentes positions de l'icone,
- h: Editer le fichier user_functions.py.

Pour les trois permiers points, il faut formuler une remarque: Lors de la conception des icones il vaut mieux penser a ne pas créer d'icone trop large. On peut considérer que 20x20px est une taille raisonnable. De plus il faut que les images soient au format PNG.

Concernant le point « c », il faut être informé de la chose suivante: La position de l'icone sur l'écran est déterminée par:

- le coin de l'écran choisissez.
- le décalage par rapport au bord de cet écran.

Les coins d'écran sont indexés et cet indexation modifie l'orientation des coordonées:



Concernant les points « b » et « d », il faut savoir qu'il y a un lien entre le nom de l'icone, celui des images et le nom de fonctions appelées. De plus selon le nombre de positions de l'icone, il y a une nomenclature à respecter.

L'ajout d'une icône consiste à son adjonction à la liste G[" **LOI** "] en faisant:

```
G[ " LOI "].append(tuple_paramètres)
```

Cette liste est transformée internement par BlasterTK et elle est répartie en deux dictionnaires :

- G[" **SObj** "][*nom_objet*]["**icons** "][*nom_de_l_icone*] = *instance_de_icon*
- G[" **Icons** "][*nom_de_l_icone*] = *instance_de_icon*

Ces deux dictionnaires sont ensuite analysés lors des évènements et redraw.

La classe **icon** accepte plusieurs paramètres obligatoires qui doivent être donc délcarés par *tuple_paramètres*.

tuple_paramètres = (*str(TYPE), str(NOM), int(index_COIN), tuple(int(MargeX), int(MargeY)), bool(lien_Objet), bool(ClickDroit), bool(Sauver)****)*

Exemple:

```
G["LOI"].append( (" binary ", " Blastme ", 3, (5,5), True, False, False) )
```

1.1 Construction des Space Handlers.

Il faut deux space handlers pour que le module BlasterTK fonctionne correctement: un pour capturer les évènements, l'autre pour dessiner les icones à l'écran.

1.1.1 #SPACEHANDLER.VIEW3D.DRAW

Ce script est capital et il doit obligatoirement contenir EXACTEMENT les lignes suivantes:

```
#SPACEHANDLER.VIEW3D.DRAW
####The following block is compulsary for any BlasterTK screen
####              IMPORT OF BLASTER MODULES AND FRIENDS     ####
import Blender, sys
def __init__(): #expands python path to Blaster dir
        script = Blender.Get("scriptsdir") + "/blaster"
        if sys.path.count(script) == 0:
                script = Blender.Get("scriptsdir") + "/blaster"
                sys.path.append(script)
        else:
                pass
        return
__init__()
import BlasterTK_FV
from Blender import G
####Entering BlasterTK drawing
BlasterTK_FV.main_draw()
####       END OF IMPORT OF BLASTER MODULES AND FRIENDS     ####
```

La fonction __init__ () étend le path de Python jusqu'au répertoire de Blaster. Sans cela, le module ne fonctionnera pas!

La fonction BlasterTK_FV.main_draw() est une fonction minimale qui se contente de lire les deux dictionnaires d'icones spécifiques à Blaster et de dessiner les icones en fonction. Si ses capacités ne sont pas suffisantes, elle peut être remplacée par une plus spécifique.

Néanmoins, cette fonction initialise également OpenGL pour la vue 3D concernée, sans quoi, les icones seraient dessinées au mauvais endroit.

Elle fait aussi appel à la fonction `BlasterTK_FV.modif_detect()` pour repérer d'éventuels changements dans le contenu de la scène.

1.1.2 #SPACEHANDLER.VIEW3D.EVENT

Ce script est beaucoup plus simple mais doit être lancé après le .DRAW.
Il peut se contenter des choses suivantes:

```
#SPACEHANDLER.VIEW3D.EVENT
import BlasterTK_FV
BlasterTK_FV.main_event()
```

Comme pour main_draw(), main_event() est une simple fonction pouvant suffire à capturer les clics gauches et droits sur des icones et d'éxécuter les fonctions nécessaires.

1.2 Déclarartion de nouvelles icones

1.2.1 Cas d'une icône unaire: « unique »

Une telle icône n'appelle qu'une seule fontion (sauf si elle accepte un clic droit), mais nécessite deux images.
La fonction porte le nom `nom_de_l_icone`, et les deux images associées sont nommées `nom_de_l_icone.png` et **un**`nom_de_l_icone.png`.

1.2.2 Cas d'une icône binaire: « binary »

Appelle deux fonctions et nécessite deux images.
Les fonctions s'appellent `nom_de_l_icone()` et **un**`nom_de_l_icone()`
`Les images s'appellent nom_de_l_icone.png et` **un**`nom_de_l_icone.png`

1.2.3 Cas d'une icône ternaire: « ternary »

Appelle trois fonctions et nécessite trois images.
Les fonctions s'appellent `nom_de_l_icone()`, **bis_**`nom_de_l_icone()` et **un**`nom_de_l_icone()`
Les images s'appellent `nom_de_l_icone.png`, **bis_**`nom_de_l_icone.png` et **un**`nom_de_l_icone.png`

1.2.4 Cas d'une icône acceptant le click droit

Dans ce cas, il ne faut pas d'image supplémentaire. Il suffit de rajouter une fonction portant le `nom_de_l_icone` préfixée par **rclk:**
rclk`nom_de_l_icone()`

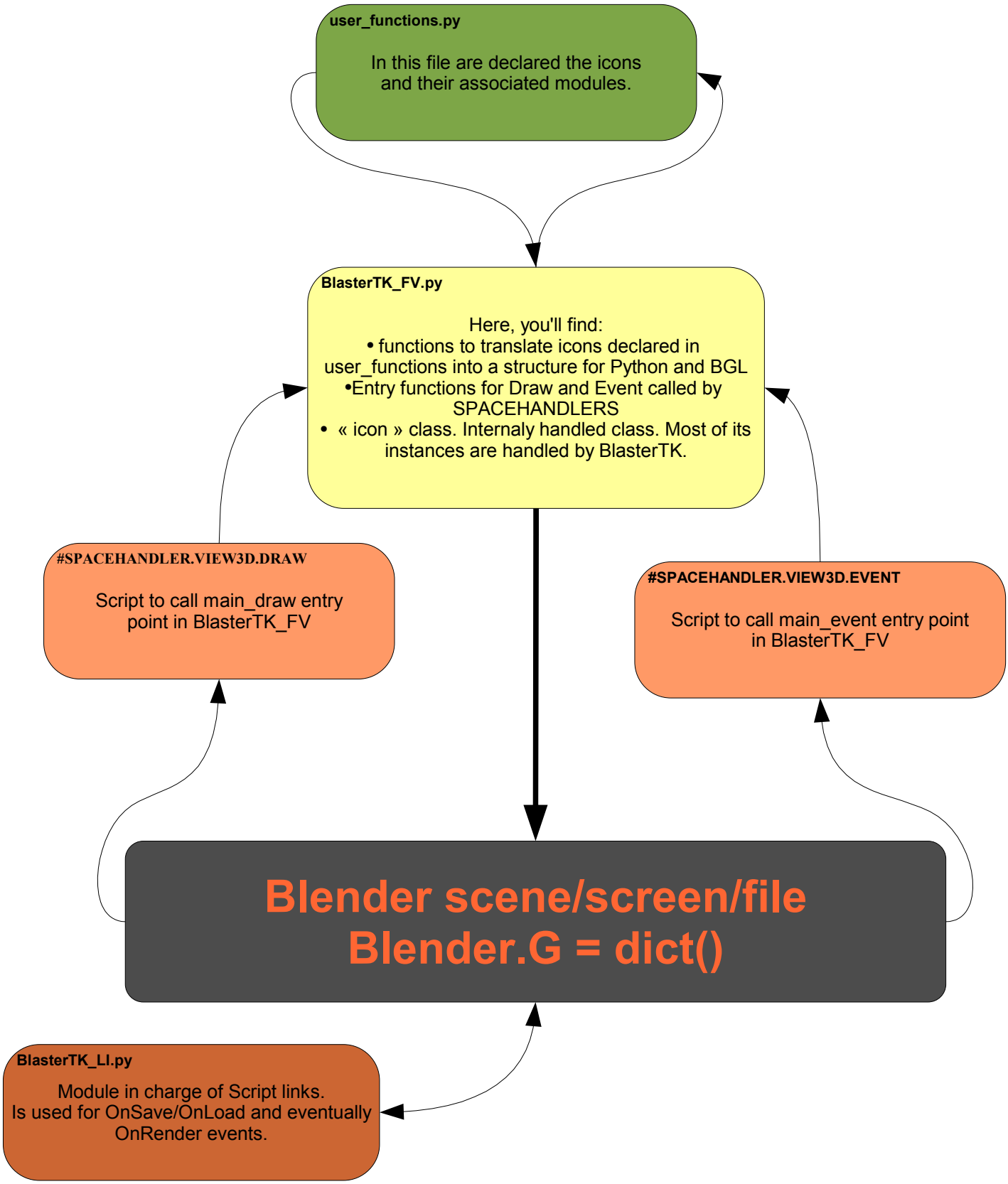
1.2.5 Icône liéée à l'objet sélectionné

Les icones liéées à l'objet sélèctionné ne sont actives que lorsqu'un seul objet est actif.
Les icons non-liéées sont dessinnées tout le temps.

BlasterTK : Fonctionnement

BlasterTK is a set of Python modules for Blender. They allow the developper to include interactive icones in the 3D view. These modules are:

- BlasterTK_FV.py
- BlasterTK_LI.py
- user_functions.py



1. Creating a new icon.

For this, you need:

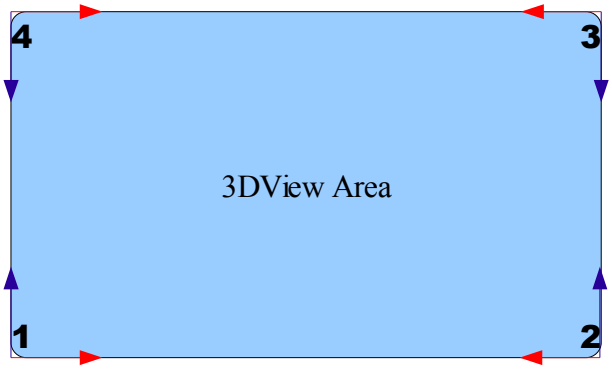
- a: Number of states for the icon (unique, binary, ternary ...),
- b: Create as many .png pics as the number of states of the icon. (except for « unique » which needs two anyway)
- c: Place the icon in the 3D view.
- d: Is the icon's data dependant on the selected object?
- e: Does the icon have a right-click action?
- F: If the icon is hidden, should its data persist?
- g: Write the functions called by this icon.
- h: All this should take place in user_functions.py.

Note on a, b, c: the developper shouldn't create icons too large. Consider 20x20px as a reasonable size. Pictures must be in PNG. Alpha is supported.

Specifically concerning « c »: The icon's position in the 3DView is determinated by:

- the corner you choose for it.
- A margin to the border of the 3Dview.

The corners are indexed and this modifies it's coordinates:



Note on « b » and « d »: There is a crucial link between the icon's name and the identifier of the called functions. These identifiers depend on the number of states of the icon and if it allows right clicks.

Adding an icon is all about adding it to G[" **LOI** "], and this happens in user_functions.py:

```
G[ " LOI "].append(tuple_parameters)
```

This list is internally converted by BlasterTK into dictionnaries :

- G[" **SObj** "][*object_name*]["**icons** "][*icon_name*] = *icon_instance*
- G[" **Icons** "][*object_name*] = *icon_instance*

These two dictionnaries are then analysed after Draw and Event events...

The **icon** class takes seven paremeters that are compulsory. They should be passed through `tuple_parameters`

tuple_parameters = (*str(TYPE), str(NOM), int(index_COIN), tuple(int(MargeX), int(MargeY)), bool(Object_relat), bool(RghtClick), bool(Save)****)*

Exemple:

```
G["LOI"].append( (" binary ", " Blastme ", 3, (5,5), True, False, False) )
```

1.1 Construction of the Space Handlers.

Two space handlers are needed in order to have BlasterTK correctly running: one to catch events, the other to draw the icons.

1.1.1 #SPACEHANDLER.VIEW3D.DRAW

This script is vital and MUST CONTAIN EXACTLY the next lines:

```
#SPACEHANDLER.VIEW3D.DRAW
####The following block is compulsary for any BlasterTK screen
####             IMPORT OF BLASTER MODULES AND FRIENDS     ####
import Blender, sys
def __init__(): #expands python path to Blaster dir
        script = Blender.Get("scriptsdir") + "/blaster"
        if sys.path.count(script) == 0:
                script = Blender.Get("scriptsdir") + "/blaster"
                sys.path.append(script)
        else:
                pass
        return
__init__()
import BlasterTK_FV
from Blender import G
####Entering BlasterTK drawing
BlasterTK_FV.main_draw()
####      END OF IMPORT OF BLASTER MODULES AND FRIENDS     ####
```

__init__ () extends Python path down to Blaster's dir. Nothing will work without this!

BlasterTK_FV.main_draw() is a minimal function which is enough to draw correctly the icons on screen. If it's capacities aren't sufficient it can be replaced. But careful! Later we will discuss how to do this.

Nevertheless, this function also initialised OGL for the 3D view. Without this, the icons owuld be draw in the wrong place.

It calls `BlasterTK_FV.modif_detect() to spot changes in the sceen..`

1.1.2 #SPACEHANDLER.VIEW3D.EVENT

Much simpler but must be run after the .DRAW handler.
It can run with just the following lines:

```
#SPACEHANDLER.VIEW3D.EVENT
import BlasterTK_FV
BlasterTK_FV.main_event()
```

As for main_draw(), main_event() is a simple function that should be enough to catch events and distribute them correctly though BlasterTK or back to Blender.

1.1 Declaring the new icon « `icon_name` »

1.1.1 Unary (?) icon: « unique »

Only calls one function (excepted if it allows a right click), but needs two pics.
The function is called `icon_name()`, and the two associated pics are `icon_name.png` and **un**`icon_name.png`.

1.1.2 Binary icon: « binary »

Calls two functions and needs two pics.
Function identifiers are: `icon_name()` and **un**`icon_name()`
The pics are `icon_name.png` and **un**`icon_name.png`

1.1.3 Ternary icon: « ternary »

Calls three functions and needs three pics..
Functions are: `icon_name()`, **bis_**`icon_name()` and **un**`icon_name()`
Pictures are called: `icon_name.png`, **bis_**`icon_name.png` and **un**`icon_name.png`

1.1.4 Accepts right-click:

No need of an extra imagen just one more function which's indentifier is built as follows: `icon_name` prefixed by **rclk:**
rclk`icon_name()`

1.1.5 Selected object dependant icon:

Such icons are only active when an object is selected.
The others are draw all the time.