# GUI Library Functions

The GUI library allows the creation of some basic GUI elements such as buttons, checkboxes, radio buttons, dialog boxes, spinners, dropdown lists, popup menus, and frames. It also has an option to create a keypad for data entry. By default most GUI widgets are drawn at depth 5 except for dialog boxes and popup menus which are drawn at depth zero.
Generally, the functions here return 1 to indicate success and 0 to indicate failure or 1 to indicate TRUE and 0 to indicate FALSE. Where other values are returned, the descriptions give details. Parameters end with # to indicate a real (floating point) value and $ to indicate a string. Integer values have no special end character. When a widget is disabled, its opacity is set to 128 (50%). All widgets ( except traffic lights and frame) are fixed-to-screen and are not affected by zoom or viewoffset settings.

## GUIButton

A button displays a three-vertical-frame image sprite (or creates a simple default one) and overlayed text (may be blank). Frame one displays by default, frame two when the pointer is over the button, frame three when the mouse button is pressed. Only frames one and three will be seen on a touch device. Buttons can be made invisible

| | | |
|---|---|---|
| int CreateGUIButton(x#,y#,w#,h#, g$, t$) | Creates button (dim *w#* x *h#*) at (*x#*,*y#*) ,img *g$*, txt *t$*. Returns ID assigned to button. | |
| int DeleteGUIButton(id) | Deletes button *id*. Returns 1: ok, 0:fail | |
| int GetGUIButtonEnabled(id) | Returns 1: enabled, 0:disabled, -1:fail | |
| int GetGUIButtonExists(id) | Returns 1: button *id* exists, 0: not exists | |
| int GetGUIButtonVisible(id) | Returns 1:button *id* visible, 0:invisible -1: not exists | |
| flt GetGUIButtonX(id) | Returns x-coord of button *id*. -1: not exists | |
| flt GetGUIButtonY(id) | Returns y-coord of button *id*. -1: not exists | |
| int HandleGUIButton(id) | Reacts to user. Returns 1: pressed, 0: not pressed/not exists/disabled | |
| int SetGUIButtonDepth(id, ly) | Button to depth *ly*. Returns 1: ok, 0: fail | * |
| int SetGUIButtonEnabled(id,fl) | Button dis(*fl*=0)/en(*fl*=1)abled. Returns 1:ok,0:fail | * |
| int SetGUIButtonPosition(id, x#, y#) | Button position to (*x#*,*y#*). Returns 1:ok, 0:fail | * |
| int SetGUIButtonSize(id, w#, h#) | Button size to *w#* by *h#*. Returns 1:ok, 0:fail | * |
| int SetGUIButtonVisible(id,f) | Button visible (*f*=1)/invisible (*f*=0). Returns 1:ok, 0:fail | * |

* Can modify attributes of disabled buttons

## GUICheckbox

A checkbox consists of a two-vertical frame image sprite and associated text. Clicking on the image or text will cause the checkbox to flip to its alternate setting (checked/unchecked). Default image used if no image file specified.

| | | |
|---|---|---|
| int CreateGUICheckbox(x#, y#, w#, g$, t$) | Positions checkbox at (*x#*,*y#*). Shows image *g$* (*w#* wide) and text *t$*.Returns ID assigned. | |
| int DeleteGUICheckbox(id) | Deletes checkbox *id*. Returns 1:ok, 0:fail | |
| int GetGUICheckboxExists(id) | Returns 1:exists, 0:not exists | |
| int GetGUICheckboxState(id) | Returns current frame (1/2), 0:disabled, -1:not exists | |
| int HandleGUICheckbox(id) | Makes checkbox reacts to user clicks Returns frame shown by checkbox *id* (1/2).. 0:disabled/deleted/not hit | |
| int SetGUICheckboxDepth(id, ly) | Places checkbox on depth *ly*. Returns 1:ok, 0:fail | * |
| int SetGUICheckboxEnabled(id,fl) | Checkbox dis(*fl*=0)/en(*fl*=1)abled. Returns 1:ok, 0:fail | |
| int SetGUICheckboxPosition(id, x#, y#) | Places checkbox at (*x#*,*y#*). Returns 1:ok, 0:fail | * |
| int SetGUICheckboxState(id, s) | Changes frame showing to *s* (1/2). Returns 1:ok, 0:fail | |
| int SetGUICheckboxTextAlignment(id, al) | Changes text alignment to *al*. Returns 1:ok, 0: fail | * |
| int SetGUICheckboxTextColor(id, col) | Changes text colour to *col*. Returns 1:ok, 0: fail | * |
| int SetGUICheckboxTextSize(id, sz#) | Changes text size to *sz#*. Returns 1:ok, 0:fail | * |

*Can modify attributes of disabled checkboxes.

## GUIColorPicker

The ColorPicker widget allows the user to select any colour shown on the ColorPicker sprite. Only a single ColorPicker widget can exist at any one time. The last colour picked is saved and its value can be accessed.

| | |
|---|---|
| CreateGUIColorPicker(x#,y#,w#,h#,fg$) | Creates sprite of image *fg$* at (*x#,y#*) with size *w#* by *h#*. |
| int DeleteGUIColorPicker() | Deletes ColorPicker. Returns 1: ok, 0: fail |
| int GetGUIColorPickerBlue() | Returns blue value of last colour picked, -1:fail |
| int GetGUIColorPickerExists() | Returns 1: exists 0: not exists |
| int GetGUIColorPickerGreen() | Returns green value of last colour picked, -1:fail |
| int GetGUIColorPickerRed() | Returns red value of last colour picked, -1:fail |
| int HandleGUIColorPicker() | Returns colour selected as integer (0xFFBBGGRR). -1:none |
| int SetGUIColorPickerDepth(d) | Draws the ColorPickers sprite on layer *d*. Returns 1:ok, 0:fail |
| int SetGUIColorPickerPosition(x#, y#) | ColorPicker position to (*x#*,*y#*). Returns 1:ok, 0:fail |

## GUIDialogBox

A dialog box consists of an image sprite and a single button by default (more can be added). Pressing a dialog box button causes the dialog box to be deleted and the pressed button's number is returned (1,2,3, etc.). Only a single Dialog widget can exist at any one time.

| | |
|---|---|
| int CreateGUIDialogBox(x#, y#, w#, h#, g$, t$, bg$, bt$) | Creates a dialog box *w#* x *h#*, at (*x#*,*y#*),box framed by image *g$* and title *t$*. Button images *bg$* (\| separated) showing *bt$* (\| separated). |
| int GetGUIDialogBoxExists() | Returns 1: exists, 0: not exists |
| int HandleGUIDialogBox() | Deletes dialog box. Returns no. of button pressed,0:fail |
| int SetGUIDialogBoxButtonPosition(n, x#, y#) | Repositions button *n* to (*x#*,*y#*). Returns 1:okay, 0:fail |
| int SetGUIDialBoxButtonSize(n, w#,h#) | Resizes button *n* to *w#* by *h#*. Returns 1:okay, 0:fail. |

## GUIDropdown

A dropdown list is related to the edit box. The data placed in the field must be selected from a drop-down list which is itself activated by pressing the integrated DOWN button.

| | |
|---|---|
| int CreateGUIDropdown(x#,y#,op$,df$,lf$) | Creates a dropdown list at (*x#,y#*). *op$* lists the contents of the dropdown list options (separated by a pipe (\|)). *df$* is the filename for the down button image (3 frames). *lf$* is the filename for the option list image (3 frames). Returns the ID assigned to the newly created widget |
| int DeleteGUIDropdown(id) | Deletes dropdown. Returns 1:ok, 0:fail |
| int GetGUIDropdownEnabled(id) | Returns 1: enabled, 0: disabled; -1: fail |
| int GetGUIDropdownExists(id) | Returns 1: exists, 0: not exists |
| str GetGUIDropdownValue(id) | Returns value showing in dropdown, "":fail. |
| int HandleGUIDropdown(id) | Reacts to the user clicking. Returns 1:click 0:no click/fail |
| int SetGUIDropdownBorderColor(id,col) | Changes border colour to *col*. Returns 1:ok, 0:fail |
| int SetGUIDropdownDepth(id, ly) | Sets depth to *ly*. Returns 1:ok, 0:fail. |
| int SetGUIDropdownEnabled(id fl) | Dropdown dis(*fl*=0)/en(*fl*=1) abled. Returns 1:ok, 0:fail |
| int SetGUIDropdownFieldColor(id,col) | Changes field colour of *id* to *col*. Returns 1:ok, 0:fail |
| int SetGUIDropdownPosition(id, x#, y#) | Sets *id*'s position to (*x#,y#*). Returns 1:ok, 0:fail |

## GUIFrame

A frame is an area to which other elements can be added. The frame may be filled with a background image and have a title. Elements positioned within a frame have positions relative to the top-left corner of the frame. Added elements are given an index number starting at 1. Moving a frame automatically moves the elements within the frame. Deleting a frame also deletes all the elements it contains.

| | |
|---|---|
| int CreateGUIFrame(x#,y#,w#,h#,g$) | Creates frame (size *w#* x *h#*) at (*x#*,*y#*) filled with image *g$*. Returns frame ID. |
| int AddButtonToGUIFrame(frm,x#,y#,w#,h#,g$,t$) | Creates button in frame at *(x#,y#)*. size:*(w#xh#);* image(3 frames):*g$;* text:*t$*. Returns button's frame index,0:fail |
| int AddCheckboxToGUIFrame(frm,x#,y#,w#,g$,t$) | Creates a checkbox in frame at *(x#,y#)*. image width: *w#*,image(2 frame):*g$;* text:*t$*. Returns checkbox's frame index, 0:fail |
| int AddColorPickerToGUIFrame(frm, x#, y#, w#, h#,g$) | Creates colorpicker in frame at (*x#*,*y#*). Size: *w#* by *h#* Image : *g$*. Returns button's frame index,0:fail |
| int AddDropdownToGUIFrame(frm,x#,y#,op$,g1$,g2$) | Creates dropdown in frame at (*x#*,*y#*). Dropdown list contains entries from *op$* (\| separated). Uses *g1$* as image for drop button; *g2$* for image in each option. Returns checkbox's frame index, 0:fail |
| int AddEditboxToGUIFrame(frm, x#, y#, w#, h#) | Creates edit box in frame at (*x#*,*y#*). Size: *w#* x *h#*. Returns edit box's frame index; 0:fail |
| int AddKeyPadToGUIFrame(frm,x#,y#,k$,kw#,kr,g$) | Creates a keypad in frame at (x#,y#). *k$* holds keys (\| sep arated). *kw#*: key width. *kr*: keys in one row. *g$*:key image(3 frames). Returns edit box's frame index; 0:fail |
| int AddRadioButtonToGUIFrame(frm,x#,y#,g$,t$,gp) | Creates radio button in frame at *(x#,y#)*. image(2 frame):*g$;* text:*t$*. In group *gp*. Returns RB's frame index, 0:fail |
| int AddSpinnerToGUIFrame(frm,x#y#,w#,h#,g$) | Creates spinner (size: *w#* x *h#*) in frame at (*x#*,*y#*). Buttons loaded from "up"+*g$* and *"down"*+*g$*. Returns RB's frame index., 0:fail |
| int AddSpriteToGUIFrame(frm,x#,y#,w#,h#,g$) | Creates a sprite with image *g$* in frame. Size: *w#* x *h#* at (*x#*,*y#*). Returns sprite's frame index, 0:fail |
| int AddTextToGUIFrame(frm,x#,y#,sz#,t$,col,al) | Creates a text in frame at *(x#,y#)*. size:*sz#;* colour:*col* text:*t$;* alignment:*al*. Returns text's frame index, 0:fail |
| int DeleteGUIFrame(frm) | Deletes frame *frm*. Returns 1:okay, 0:fail. |
| int GetGUIFrameElementDetails(frm,idx) | Returns details of the *idx*th added widget. (element type *100000 + true id), 0: fail |
| int GetGUIFrameExists(frm) | Returns 1: exists, 0: not exists |
| flt GetGUIFrameHeight(frm) | Returns height of frame, -1: fail |
| flt GetGUIFrameWidth(frm) | Returns width of frame, -1: fail |
| flt GetGUIFrameX(frm) | Returns x-coord (top-left) of frame , -1: fail |
| flt GetGUIFrameY(frm) | Returns x-coord (top-left) of frame , -1: fail |
| int HandleGUIFrame(frm) | Returns frame index of any frame element clicked by user. |
| int SetGUIFrameDepth(frm,ly) | Places frame on depth *ly*. Returns 1: okay, 0: fail |
| int SetGUIFramePosition(frm,x#,y#) | Positions frame at (*x#*,*y#*). Returns 1: okay, 0: fail |

## GUIKeyPad

A keypad widget is a set of buttons each representing a single keyboard key. The keys to be shown, the size of a single key, the number of keys per row and the key image used can be specified. The keyboard widget maintains a keyboard buffer containing the keys entered since the previous *Enter* key press. Once *Enter* has been pressed the buffer contents is moved to a *last-entry* space and the buffer emptied.

| | |
|---|---|
| int CreateGUIKeyPad(x#,y#,ky$,kw#,kir,g$) | Creates a keypad widget at (x#,y#). Keys created determined by *ky$ (\| separated)*. Keys are *kw* units square. Each row consists of *kir* keys. *g$*: filename of key image. Returns 1:ok, 0: fail. |
| int DeleteGUIKeyPad() | Deletes the keypad. Returns 1:ok, 0:fail |
| str GetGUIKeyPadBuffer() | Returns contents of keypad buffer, "": fail |
| int GetGUIKeyPadEnabled() | Returns 1: enabled, 0: disabled, -1: fail |
| str GetGUIKeyPadEntry() | Returns the last complete entry, "": fail |
| int GetGUIKeyPadExists() | Returns 1: exists, 0: not exists |
| int HandleGUIKeyPad() | Reacts to user. Returns 0: no press, 1: press, 2: *Enter* pressed |
| int SetGUIKeyPadDepth(ly) | Places keypad on depth *ly*. Returns 1: ok, 0: fail |
| int SetGUIKeyPadEnabled(fl) | Keypad dis(*fl*=0)/ en (*fl*=1) abled. Returns 1:ok, 0: fail |
| int SetGUIKeyPadKeysEnabled(ky$, fl) | Keys in *ky$ dis*(*fl*=0)/en(*fl*=1)abled. Returns 1:ok, 0: fail |
| int SetGUIKeyPadPosition(x#, y#) | Moves keypad to (*x#*,*y#*). Returns 1: ok, 0: fail |

## GUIPopUpMenu

The popup menu is created from a combination of vertically-aligned buttons. The dimensions of the popup menu are determined automatically. A disabled menu is invisible and unresponsive. The best approach is to create all necessary popup menu and only enable them when required. This minimises the creation and deletion of the buttons used. When creating a menu all options are specified in a single string with the pipe (|) character separating options, Each option consists of two parts text (which will appear in the menu) and an integer value (the two elements are comma-separated). The integer value is returned when the user selects a menu option.

| | |
|---|---|
| int CreateGUIPopUpMenu(op$,g$) | Creates a popup menu off-screen. *op$* contains \| separated menu options. Each option has format \|*string,number*\|. *g$* is the image used by each option when the menu is displayed. The menu is initially disabled. Returns the ID of the menu. |
| int DeleteGUIPopUpMenu(id) | Deletes the menu. Returns 1: ok, 0: fail |
| int GetGUIPopUpEnabled(id) | Returns 1: enabled, 0: disabled, -1: fail |
| int GetGUIPopUpExists(id) | Returns 1: exists, 0: not exists |
| int HandleGUIPopUpMenu(id) | Reacts to user. Returns value of option selected, 0:fail |
| int SetGUIPopUpEnabled(id, fl) | Popup dis(*fl*=0)/en (*fl*=1)abled. Returns 1:ok, 0: fail |
| int SetGUIPopUpPosition(id,x#,y#) | Moves popup to (*x#*,*y#*). Returns 1: ok, 0: fail |

## GUIRadioButton

A radio button consists of a two-vertical frame image sprite and associated text. Radio buttons are associated with a group number. Clicking on the image or text will cause an unselected radio button to become selected and all other radio buttons in that group to be unselected. In the parameters below *id* refers to an integer value of the *ggxxx* where *gg* represents the radio button group and *xxx* represents the index of the radio button within that group.

| | |
|---|---|
| int CreateGUIRadioButton(x#,y#,w#,g$,t$,gp) | Positions RB at (*x#, y#*). Shows image *g$* (*w#* wide) & text *t$* Belongs to group *gp*. Returns ID assigned |
| int DeleteGUIRadioButtonGroup(gp) | Deletes all RBs in group *gp*. Returns 1:ok, 0: fail |
| int GetGUIRadioButtonExists(id) | Returns 1: exists, 0: not exists |
| int GetGUIRadioButtonSelectedInGroup(gp) | Returns no. of selected button in group, -2: fail,-1:no RBs in group |
| int HandleGUIRadioButtonGroup(gp) | Reacts to user. Returns 1:RB clicked, 0:no click/fail |
| int SelectGUIRadioButton(id) | Selects RB *id* (calcs *gp* & idx from *id*). Returns 1: ok, 0:fail |
| int SetGUIRadioButtonDepth(gp, ly) | Sets depth of all RBs in group *gp* to *ly*. Returns 1: ok, 0: fail |
| int SetGUIRadioButtonEnabled(id,fl) | RB dis(*fl*=0)/en(*fl*=1)abled. Returns 1:ok, 0:fail |
| int SetGUIRadioButtonGroupEnabled(gp,fl) | Group *gp* dis(*fl*=0)/en(*fl*=1)abled. Returns 1: ok, 0: fail |
| int SetGUIRadioButtonPosition(id, x#, y#) | Places RB at (*x#,y#*). Returns 1: ok, 0: fail |
| int SetGUIRadioButtonTextAlignment(id,al) | Text alignment to *al (0:L,1:C,2:R)*. Returns 1: ok,0:fail |
| int SetGUIRadioButtonTextColor(id,col) | Sets text colour of RB *id* to *col*. Returns 1: ok, 0:fail |
| int SetGUIRadioButtonTextSize(id,sz#) | Sets text size of RB *id* to *sz#*. Returns 1: ok, 0: fail |

## GUISpinner

A spinner is a limited type of editbox where a displayed integer value can be incremented or decremented using associated UP and DOWN buttons. The widget border and field containing the integer value can be recoloured. The images used to create the UP and DOWN buttons can also be specified.

| | |
|---|---|
| int CreateGUISpinner(x#,y#,w#,h#,min, max,bf$) | Creates a spinner at (*x#,y#*) size: *w#* by *h#*. Separates *bf$* into *path + filename*. Uses file *path*+"up"+*filename* and *path* + "down"+*filename* as images for buttons. Range of acceptable values *min* to *max*. Returns spinner ID |
| int DeleteGUISpinner(id) | Deletes spinner *id*. Returns 1: ok, 0:fail |
| int GetGUISpinnerEnabled(id) | Returns 1: enabled; 0: disabled, -1:fail |
| int GetGUISpinnerExists(id) | Returns 1: exists, 0: not exists |
| int GetGUISpinnerValue(id) | Returns the value displayed in spinner, -1: fail |
| int HandleGUISpinner(id) | Reacts to user; updates display. Returns 1:clicked, 0: no click |
| int SetGUISpinerBorderColor(id, col) | Changes border colour *col*. Returns 1:ok, 0: fail |
| int SetGUISpinnerDepth(id,ly) | Redraws spinner at depth *ly*. Returns 1: ok, 0: fail |
| int SetGUISpinnerEnabled(id, fl) | Spinner dis(*fl*=0)/en(*fl*=1)abled. Returns 1:ok, 0: fail |
| int SetGUISpinnerFieldColor(id, col) | Changes field colour of spinner to *col*. Returns 1:ok, 0: fail |
| int SetGUISpinnerLimits(id,min,max) | Sets spinner range to *min* to *max*. Returns 1: ok, 0: fail, -2: current value to new max, -1: current value to new min. |
| int SetGUISpinnerPosition(id,x#,y#) | Moves spinner to (*x#,y#*). Returns 1: ok, 0: fail |
| int SetGUISpinnerSize(id,w#,h#) | Resizes spinner to *w#* by *h#*. Returns 1: ok, 0: fail |
| int SetGUISpinnerValue(id,v) | Sets displayed value of spinner to *v*. Returns 1:ok, 0: fail |

## GUIStopwatch

The Stopwatch widget (see supplement) allows the elapsed time to be shown in minutes and seconds (internally recorded in milliseconds). The visuals are created by four images and a text label. Although in theory the programmer can create their own images this may prove difficult because of the hard-wired code for positioning and sizing the various elements when the watch is created. It is best to use the default images supplied. The user can control the starting and stopping of the stopwatch as well as resetting its time to zero by pressing the displayed watch buttons. This ability can be disabled.

| | |
|---|---|
| int CreateGUIStopwatch(x#,y#,w#,h#,f$) | Positions stopwatch at (*x#,y#*); dimensions (w# by h#) Constructed from image file f$ and related named files. Returns ID assigned to watch |
| int DeleteGUIStopwatch(id) | Deletes watch *id*. Returns 1: ok, 0: fail |
| int GetGUIStopwatchExists(id) | Returns 1 if watch with ID id exists, else 0 returned |
| int GetGUIStopwatchResetEnabled(id) | Returns 1:reset button enabled, 0: disabled, -1: fail |
| int GetGUIStopwatchStartEnabled(id) | Returns 1:start/stop button enabled, 0: disabled, -1: fail |
| int GetGUIStopwatchState(id) | Returns 0: watch stopped, 1: watch ticking, -1: fail |
| int GetGUIStopwatchTime(id) | Returns the time recorded in watch (msecs), -1: fail |
| int HandleGUIStopwatch | Reacts to user Returns -1: stop, 1: start, 2: reset, 0: none |
| int ResetGUIStopwatch(id) | Sets time to 0; stops watch. Returns 1: ok, 0: fail |
| int SetGUIStopwatchControls(id,fl) | fl=1:enable s/s disable reset. fl= 2 disable s/s enable reset fl = 3 enable both, fl=0: disable both. Returns 1: ok, 0: fail |
| int SetGUIStopwatchDepth(id, ly) | Sets depth to *ly* (second hand: *ly*-1). Returns 1: ok, 0: fail |
| int SetGUIStopwatchPosition(id,x#,y#) | Positions top-left at (*x#,y#*). Returns 1: ok, 0: fail |
| int SetGUIStopwatchSize(id,w#,h#) | Resizes to *w#* by *h#* (best w#/h# as -1). Returns 1: ok, 0: fail |
| int StartGUIStopwatch(id) | Starts watch running. Returns 1: ok, 0: fail |
| int StopGUIStopwatch(id) | Stops watch(disp'd time unchanged). Returns 1: ok, 0: fail |
| int UpdateGUIStopwatch(id) | Updates display. Returns 1: ok, 0: fail |

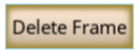
## GUITrafficLights

The trafficlight widget (see supplement) cycles through the colours red, amber, green, changing to the next colour when a mouse click occurs over the widget. Unlike other widgets, this one may be used as part of a game screen layout and therefore will relocate when the view-offset is changed and will change size in response to zooming unless *FixGUITrafficLightsToScreen()* called.

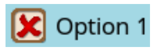
| | |
|---|---|
| int CreateGUITrafficLights(x#,y#,w#,g$) | Creates a trafficlights widget at (*x#,y#*); width : *w#*, height: dependant on image *g$* (four vertical frames). Returns ID. |
| int DeleteGUITrafficLights(id) | Deletes lights *id*. Returns 1: ok, 0:fail |
| int FixGUITrafficLightsToScreen(id, fl) | Fixes size & position of lights on screen. Returns 1:ok, 0: fail |
| int GetGUITrafficLightsEnabled(id) | Returns 1: enabled; 0: disabled, -1:fail |
| int GetGUITrafficLightsExists(id) | Returns 1: exists, 0: not exists |
| int GetGUITrafficLightsState(id) | Returns 1,2,3: current frame, -1: not exists, 0: disabled |
| int HandleGUITrafficLights(id) | Click moves to next frame. Returns 1:clicked, 0: no click |
| int SetGUITrafficLightsColor(id, cf) | Sets frame displayed to *cf*. Returns 1: ok, 0: fail |
| int SetGUITrafficLightsDepth(id,ly) | Redraws lights at depth *ly*. Returns 1: ok, 0: fail |
| int SetGUITrafficLightsEnabled(id, fl) | Lights dis(*fl*=0)/en(*fl*=1)abled. Returns 1:ok, 0: fail |
| int SetGUITrafficLightsPosition(id,x#,y#) | Moves lights to (*x#,y#*). Returns 1: ok, 0: fail |

## Widget Examples

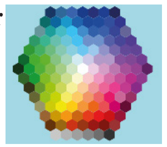
**Button**



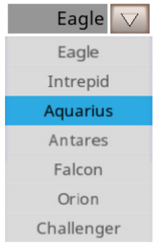
**Checkbox**



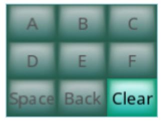
**ColorPicker**



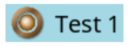
**Dropdown List**



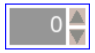
**Keypad**



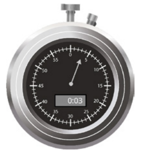
**Radiobutton**



**Spinner**



**Stopwatch**



**TrafficLights**

for

# Hands On AppGameKit Studio

Volume 2

## User-Defined GUI Library Functions

The functions listed here are created by various book activities and become a library of user-defined routines for use in other projects.

November 2020

www.digital-skills.co.uk