Bgui Documentation

Release 0.09

Mitchell Stokes (Moguri)

November 21, 2016

Contents

CHAPTER 1

Tutorials

1.1 Getting Started

This tutorial is intended to get you up and running with Bgui quickly and easily. This tutorial assumes you already know how to use Python with the BGE. If you do not, take a look here.

1.1.1 Getting Bgui

First, you'll need to either grab a "release" version of Bgui from here, or you can grab the latest development version from here. After you've downloaded Bgui copy the "bgui" folder to where you need to for your project to access it (the current working directory of the project works well). To test if your project can find Bgui simply try an `import bgui` in a script and try to run it from the BGE.

1.1.2 Setup a System

After getting Bgui setup in your project, the next step is to setup a System. In Bgui a System is the top level element in a GUI. It handles mouse and keyboard events and renders the widgets. Bgui's ultimate goal is to be independent of Blender and the BGE. However, there is a `bgui.bge_utils` module that contains classes for getting Bgui setup quickly for the BGE:

```
import bgui
import bgui.bge_utils
import bge


class SimpleLayout(bgui.bge_utils.Layout):
    """A layout showcasing various Bgui features"""

    def __init__(self, sys, data):
        super().__init__(sys, data)

        # Add widgets here

def main(cont):
    own = cont.owner
    mouse = bge.logic.mouse

    if 'sys' not in own:
        # Create our system and show the mouse
```

```
        own['sys'] = bgui.bge_utils.System('../../themes/default')
        own['sys'].load_layout(SimpleLayout, None)
        mouse.visible = True
    else:
        own['sys'].run()
```

1.1.3 BGE Logic

To get Bgui working in the BGE we'll need to setup a bit of logic. The good news is it's pretty simple: just add an always sensor (pulse mode on) and a Python module controller. Now, assuming you used the earlier example, just enter `name_of_py_file.main` into the module controller. You should now be able to press "P", but you wont see much because we haven't added any widgets yet. So, let's get to that next.

1.1.4 Adding Widgets

Widgets (also known as controls or components in other GUI libraries) are the actual elements that will be drawn to the screen. This includes things like text and buttons. At the time of this writing, Bgui currently has the following widgets available:

- *Frame*
- *FrameButton*
- *Image*
- *ImageButton*
- *Label*
- *ListBox*
- *ProgressBar*
- *TextBlock*
- *TextInput*
- *Video*

Let's go ahead and add a *FrameButton* and a *Label* to our example. After the `# Add widgets here` line add the following:

```
# Use a frame to store all of our widgets
self.frame = bgui.Frame(self, border=0)
self.frame.colors = [(0, 0, 0, 0) for i in range(4)]

# A Label widget
self.lbl = bgui.Label(self.frame, text='I sure wish someone would push that button...',
        pt_size=70, pos=[0, 0.7], options=bgui.BGUI_CENTERX)

# A FrameButton widget
self.btn = bgui.FrameButton(self.frame, text='Click Me!', size=[0.3, 0.1], pos=[0, 0.4],
        options=bgui.BGUI_CENTERX)
```

I won't go much into the constructors for these widgets. You can look up more on the constructors in the *API docs*.

Okay, so now we test the changes. You should have a label and a button, both just asking for the button to be pushed. However, when we push the button, nothing much happens other than pretty button effects. A GUI isn't much use if it can't actually do anything. To add actions to widgets, we use callbacks, which are described in the next section.

1.1.5 Callbacks

Alright, time to get our button to do something. The *FrameButton* widget has an `on_click` callback that we can make use of. Add the following after creating the button:

```
self.btn.on_click = self.button_click
```

And then add the following method to the SimpleLayout class:

```
def button_click(self, widget):
    self.lbl.text = 'Yippie! You clicked the button! ^_^'
```

Now if you test the new changes, you should get a very ecstatic message when clicking the button.

1.1.6 What next?

Okay, so where to go from here, right? Well, unfortunately there isn't much in the way of docs, so I'd recommend taking a look at the examples in the example folder. They show you how to use some widgets.

CHAPTER 2

Auto-Generated API Docs

2.1 bge_utils

class bgui.bge_utils.**Layout**(*sys*, *data*)
Bases: *bgui.widget.Widget*

A base layout class to be used with the BGESystem

Parameters

- **sys** – The BGUI system
- **data** – User data

update()
A function that is called by the system to update the widget (subclasses should override this)

class bgui.bge_utils.**System**(*theme=None*)
Bases: *bgui.system.System*

A system that is intended to be used with BGE games

Parameters **theme** – the path to a theme directory

load_layout(*layout*, *data=None*)
Load a layout and replace any previously loaded layout

Parameters

- **layout** – The layout to load (None to have no layouts loaded)
- **data** – User data to send to the layout's constructor

add_overlay(*overlay*, *data=None*)
Add an overlay layout, which sits on top of the currently loaded layout

Parameters

- **overlay** – The layout to add as an overlay
- **data** – User data to send to the layout's constructor

remove_overlay(*overlay*)
Remove an overlay layout by name

Parameters **overlay** – the class name of the overlay to remove (this is the same name as the layout used to add the overlay)

toggle_overlay(*overlay*, *data=None*)
Toggle an overlay (if the overlay is active, remove it, otherwise add it)

Parameters

- **overlay** – The class name of the layout to toggle
- **data** – User data to send to the layout's constructor

run()
A high-level method to be run every frame

2.2 frame

class bgui.frame.**Frame**(*parent, name=None, border=None, aspect=None, size=[1, 1], pos=[0, 0], sub_theme='', options=0*)
Bases: *bgui.widget.Widget*

Frame for storing other widgets

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **border** – the size of the border around the frame (0 for no border)
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **options** – various other options

theme_section = 'Frame'

theme_options = {'Color4': (0, 0, 0, 0), 'BorderColor': (0, 0, 0, 1), 'BorderSize': 0, 'Color3': (0, 0, 0, 0), 'Color1': (0, 0

colors = None
The colors for the four corners of the frame.

border_color = None
The color of the border around the frame.

2.3 frame_button

class bgui.frame_button.**FrameButton**(*parent, name=None, base_color=None, text='', font=None, pt_size=None, aspect=None, size=[1, 1], pos=[0, 0], sub_theme='', options=0*)
Bases: *bgui.widget.Widget*

A clickable frame-based button.

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget

- **base_color** – the color of the button
- **text** – the text to display (this can be changed later via the text property)
- **font** – the font to use
- **pt_size** – the point size of the text to draw (defaults to 30 if None)
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **options** – various other options

theme_section = 'FrameButton'

theme_options = {'BorderSize': 1, 'Color': (0.4, 0.4, 0.4, 1), 'LabelSubTheme': '', 'BorderColor': (0, 0, 0, 1)}

text

color

2.4 gl_utils

bgui.gl_utils.**glGenTextures**(*n*, *textures=None*)

bgui.gl_utils.**glDeleteTextures**(*textures*)

bgui.gl_utils.**glGetIntegerv**(*pname*)

2.5 image

This module defines the following constants:

Texture interpolation modes

- BGUI_NEAREST
- BGUI_LINEAR

class bgui.image.**Image**(*parent, img, name=None, aspect=None, size=[1, 1], pos=[0, 0], texco=[(0, 0), (1, 0), (1, 1), (0, 1)], interp_mode=<class 'GL_LINEAR'>, sub_theme='', options=0*)

Bases: *bgui.widget.Widget*

Widget for displaying images

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **img** – the image to use for the widget
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position

- **texco** – the UV texture coordinates to use for the image
- **interp_mode** – texture interpolating mode for both maximizing and minifying the texture (defaults to BGUI_LINEAR)
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **options** – various other options

texco = None
The UV texture coordinates to use for the image.

color = None
The color of the plane the texture is on.

interp_mode
The type of image filtering to be performed on the texture.

image_size
The size (in pixels) of the currently loaded image, or [0, 0] if an image is not loaded

update_image(*img*)
Changes the image texture

Parameters **img** – the path to the new image

Return type None

2.6 image_button

class bgui.image_button.**ImageButton**(*parent, name=None, default_image=None, default2_image=None, hover_image=None, click_image=None, aspect=None, size=[1, 1], pos=[0, 0], sub_theme='', options=0*)

Bases: *bgui.widget.Widget*

A clickable image-based button.

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **default_image** – list containing image data for the default state ('image', xcoord, ycoord, xsize, ysize)
- **default2_image** – list containing image data for a second default state, which is used for toggling ('image', xcoord, ycoord, xsize, ysize)
- **hover_image** – list containing image data for the hover state ('image', xcoord, ycoord, xsize, ysize)
- **click_image** – list containing image data for the click state ('image', xcoord, ycoord, xsize, ysize)
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)

- **options** – various other options

theme_section = 'ImageButton'

theme_options = {'Default2Image': (None, 0, 0, 1, 1), 'DefaultImage': (None, 0, 0, 1, 1), 'ClickImage': (None, 0, 0, 1, 1

2.7 key_defs

2.8 label

class bgui.label.**Label**(*parent, name=None, text='', font=None, pt_size=None, color=None, outline_color=None, outline_size=None, outline_smoothing=None, pos=[0, 0], sub_theme='', options=0*)

Bases: *bgui.widget.Widget*

Widget for displaying text

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **text** – the text to display (this can be changed later via the text property)
- **font** – the font to use
- **pt_size** – the point size of the text to draw (defaults to 30 if None)
- **color** – the color to use when rendering the font
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **options** – various other options

theme_section = 'Label'

theme_options = {'OutlineSmoothing': False, 'Size': 30, 'OutlineSize': 0, 'Color': (1, 1, 1, 1), 'OutlineColor': (0, 0, 0,

text
The text to display

pt_size
The point size of the label's font

2.9 list_box

ListBoxes make use of a ListBoxRenderer. The default ListBoxRenderer simply displays an item's string representation. To make your own ListBoxRenderer create a class that has a render_item() method that accepts the item to be rendered and returns a widget to render.

Here is an simple example of using the ListBox widget:

```
class MySys(bgui.System):
        def lb_click(self, lb):
                print(lb.selected)
```

```
        def __init__(self):
                bgui.System.__init__(self)

                items = ["One", "Two", 4, 4.6]
                self.frame = bgui.Frame(self, 'window', border=2, size=[0.5, 0.5],
                        options=bgui.BGUI_DEFAULT|bgui.BGUI_CENTERED)
                self.lb = bgui.ListBox(self.frame, "lb", items=items, padding=0.05, size=[0.9, 0.9],
                self.lb.on_click = self.lb_click

                # ... rest of __init__
```

class bgui.list_box.**ListBoxRenderer**(*listbox*)

Bases: object

Base class for rendering an item in a ListBox

Parameters **listbox** – the listbox the renderer will be used with (used for parenting)

render_item(*item*)

Creates and returns a *bgui.label.Label* representation of the supplied item

Parameters **item** – the item to be rendered

Return type *bgui.label.Label*

class bgui.list_box.**ListBox**(*parent, name=None, items=[], padding=0, aspect=None, size=[1, 1], pos=[0, 0], sub_theme='', options=0*)

Bases: *bgui.widget.Widget*

Widget for displaying a list of data

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **items** – the items to fill the list with (can also be changed via ListBox.items)
- **padding** – the amount of extra spacing to put between items (can also be changed via ListBox.padding)
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **options** – various other options

theme_section = 'ListBox'

theme_options = {'HighlightColor2': (0, 0, 1, 1), 'Border': 1, 'Padding': 0, 'HighlightColor1': (1, 1, 1, 1), 'HighlightC

padding = None

The amount of extra spacing to put between items

renderer = None

The ListBoxRenderer to use to display items

items

The list of items to display in the ListBox

2.10 progress_bar

class bgui.progress_bar.**ProgressBar**(*parent, name=None, percent=1.0, sub_theme=''', aspect=None, size=[1, 1], pos=[0, 0], options=0*)

Bases: *bgui.widget.Widget*

A solid progress bar. Controlled via the 'percent' property which assumes percent as a 0-1 floating point number.

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **percent** – the initial percent
- **sub_theme** – sub type of theme to use
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **options** – various other options

theme_section = 'ProgressBar'

theme_options = {'FillColor2': (0.0, 0.42, 0.02, 1.0), 'FillColor3': (0.0, 0.42, 0.02, 1.0), 'FillColor4': (0.0, 0.42, 0.02, 1.0)

percent

2.11 system

class bgui.system.**System**(*textlib, theme=None*)

Bases: *bgui.widget.Widget*

The main gui system. Add widgets to this and then call the render() method draw the gui.

Parameters **theme** – the path to a theme directory

normalize_text = True

focused_widget

The widget which currently has "focus"

update_mouse(*pos, click_state=0*)

Updates the system's mouse data

Parameters

- **pos** – the mouse position
- **click_state** – the current state of the mouse

Return type None

update_keyboard(*key, is_shifted*)

Updates the system's keyboard data

Parameters

- **key** – the key being input
- **is_shifted** – is the shift key held down?

Return type None

render()
Renders the GUI system

Return type None

2.12 text

class bgui.text.**TextLibrary**
Bases: object

Class for handling text drawing.

load(*filename*)

draw(*fontid*, *text*)

dimensions(*fontid*, *text*)

position(*fontid*, *x*, *y*, *z*)

size(*fontid*, *size*, *dpi*)

2.13 text_block

class bgui.text_block.**TextBlock**(*parent, name=None, text='', font=None, pt_size=None, color=None, aspect=None, size=[1, 1], pos=[0, 0], sub_theme='', overflow=1, options=0*)
Bases: *bgui.widget.Widget*

Widget for displaying blocks of text

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **text** – the text to display (this can be changed later via the text property)
- **font** – the font to use
- **pt_size** – the point size of the text to draw
- **color** – the color to use when rendering the font
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **overflow** – how to handle excess text
- **options** – various other options

theme_section = 'TextBlock'

theme_options = {'LabelSubTheme': ''}

text
The text to display

2.14 text_input

This module defines the following constants:

InputText options

- BGUI_INPUT_NONE = 0
- BGUI_INPUT_SELECT_ALL = 1
- BGUI_INPUT_DEFAULT = BGUI_INPUT_NONE

class bgui.text_input.**TextInput**(*parent, name=None, text='', prefix='', font=None, pt_size=None, color=None, aspect=None, size=[1, 1], pos=[0, 0], sub_theme='', input_options=0, options=0*)

Bases: *bgui.widget.Widget*

Widget for getting text input

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **text** – the text to display (this can be changed later via the text property)
- **prefix** – prefix text displayed before user input, cannot be edited by user (this can be changed later via the prefix property)
- **font** – the font to use
- **pt_size** – the point size of the text to draw
- **color** – color of the font for this widget
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **options** – various other options

theme_section = 'TextInput'

theme_options = {'LabelSubTheme': '', 'HighlightColor': (0.6, 0.6, 0.6, 0.5), 'InactiveBorderColor': (0, 0, 0, 0), 'Inact

text

prefix

on_enter_key
A callback for when the enter key is pressed while the TextInput has focus

select_all()
Change the selection to include all of the text

select_none()
Change the selection to include none of the text

activate()

deactivate()

swapcolors(*state=0*)

update_selection()

find_mouse_slice(*pos*)

calc_mouse_cursor(*pos*)

2.15 texture

class bgui.texture.**Texture**(*path*, *interp_mode*)
Bases: object

interp_mode

bind()

class bgui.texture.**ImageTexture**(*image*, *interp_mode*, *caching*)
Bases: *bgui.texture.Texture*

reload(*image*)

class bgui.texture.**VideoTexture**(*video*, *interp_mode*, *repeat*, *play_audio*)
Bases: *bgui.texture.Texture*

reload(*video*)

update()

play(*start*, *end*, *use_frames=True*, *fps=None*)

2.16 theme

class bgui.theme.**NewSectionProxy**(*parser*, *name*)
Bases: configparser.SectionProxy

Creates a view on a section of the specified *name* in *parser*.

class bgui.theme.**Theme**(*file*)
Bases: configparser.ConfigParser

path = ''

supports(*widget*)
Checks to see if the theme supports a given widget.

Parameters **widget** – the widget to check for support

warn_legacy(*section*)

warn_support(*section*)

2.17 video

class bgui.video.**Video**(*parent, vid, name=None, play_audio=False, repeat=0, aspect=None, size=[1, 1], pos=[0, 0], sub_theme='', options=0*)

Bases: *bgui.image.Image*

Widget for displaying video

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **vid** – the video to use for the widget
- **play_audio** – play the audio track of the video
- **repeat** – how many times to repeat the video (-1 = infinite)
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **options** – various other options

play(*start*, *end*, *use_frames=True*, *fps=None*)

on_finish

The widget's on_finish callback

2.18 widget

This module defines the following constants:

Widget options

- BGUI_DEFAULT = 0
- BGUI_CENTERX = 1
- BGUI_CENTERY = 2
- BGUI_NO_NORMALIZE = 4
- BGUI_NO_THEME = 8
- BGUI_NO_FOCUS = 16
- BGUI_CACHE = 32
- BGUI_CENTERED = BGUI_CENTERX | BGUI_CENTERY

Widget overflow

- BGUI_OVERFLOW_NONE = 0
- BGUI_OVERFLOW_HIDDEN = 1
- BGUI_OVERFLOW_REPLACE = 2
- BGUI_OVERFLOW_CALLBACK = 3

Mouse event states

- BGUI_MOUSE_NONE = 0
- BGUI_MOUSE_CLICK = 1
- BGUI_MOUSE_RELEASE = 2
- BGUI_MOUSE_ACTIVE = 4

Note: The Widget class should not be used directly in a gui, but should instead be subclassed to create other widgets.

---

class bgui.widget.**WeakMethod**(*f*)
Bases: object

class bgui.widget.**Animation**(*widget*, *attrib*, *value*, *time_*, *callback*)
Bases: object

update()

class bgui.widget.**ArrayAnimation**(*widget*, *attrib*, *value*, *time_*, *callback*)
Bases: *bgui.widget.Animation*

update()

class bgui.widget.**Widget**(*parent, name=None, aspect=None, size=[1, 1], pos=[0, 0], sub_theme='', options=0*)
Bases: object

The base widget class

Parameters

- **parent** – the widget's parent
- **name** – the name of the widget
- **aspect** – constrain the widget size to a specified aspect ratio
- **size** – a tuple containing the width and height
- **pos** – a tuple containing the x and y position
- **sub_theme** – name of a sub_theme defined in the theme file (similar to CSS classes)
- **options** – various other options

theme_section = 'Widget'

theme_options = {}

name = None
The widget's name

frozen = None
Whether or not the widget should accept events

visible = None
Whether or not the widget is visible

z_index = None
The widget's z-index. Widget's with a higher z-index are drawn over those that have a lower z-index

on_click
The widget's on_click callback

on_release
The widget's on_release callback

on_hover
The widget's on_hover callback

on_mouse_enter
The widget's on_mouse_enter callback

on_mouse_exit
The widget's on_mouse_exit callback

on_active
The widget's on_active callback

parent
The widget's parent

system
A reference to the system object

children
The widget's children

position
The widget's position

size
The widget's size

move(*position*, *time*, *callback=None*)
Move a widget to a new position over a number of frames

Parameters

- **positon** – The new position
- **time** – The time in milliseconds to take doing the move
- **callback** – An optional callback that is called when he animation is complete

add_animation(*animation*)
Add the animation to the list of currently running animations

Parameters **animation** – The animation

CHAPTER 3

Change Log

3.1 0.09

3.1.1 New Features

- Adding a BGE System subclass (bgui.bge_utils.System) to make it easier to get Bgui up and running in the BGE, which is the only environment it currently supports anyways. The layout code might eventually get moved into bgui.System. The simple example has been updated to make use of the new BGE system, but the other examples have not yet been updated.
- Image widgets now have an image_size attribute that returns the pixel dimensions of their loaded images. (thanks to reach.me.et)

3.1.2 Bugs Fixed

- The outline_color for Labels was grabbing the color value instead of outline_color if outline_color was set via the constructor. (reported by SolarLune)
- position and size attributes and properties should now accept tuple values without crashing. (reported by SolarLune)

3.1.3 Other Stuff

- Widget names are now optional, which reduces the amount of typing for creating new widgets. However, to make name an optional argument required switching the arguments for the Image and Video widget constructors. All other widgets should work without changes to users' code.
- The default ListBoxRenderer now tries to display the item as a string instead of using __repr__ directly. This makes the most basic case (a list of strings) display correctly without extra fuss.
- Switching some logic so BGUI_DEFAULT is 0. This simplifies passing options to a widget constructor. For example, no more BGUI_DEFAULT | BGUI_CENTERED mess. However, this can break existing setups. In order to make this work, BGUI_NORMALIZED is now BGUI_NO_NORMALIZE and BGUI_THEMED is now BGUI_NO_THEME. If you're relying on these flags, you will have to update your scripts.
- Getting rid of Widget._cleanup() since we should have working destructors with all the WeakRefs.
- Image, TextInput, and Widget have had their default size parameter changed from [0, 0] to [1, 1]. This may break some scripts.

3.2 0.08

3.2.1 New Features

- FrameButton, TextBlock and TextInput now have a LabelSubTheme theme option which can be used to control the subtheme used for the underlying labels in these widgets.
- Labels can now have outlines.
- Widgets now have a z_index value for additional control over drawing order. (thanks to andrew-101)
- New animation system that allows for any property of a widget to be animated using the Animation and ArrayAnimation classes along with Widget.add_animation().
- The minification and magnification filters to use for Image widgets is now user settable between two possible values: BGUI_NEAREST (crisper) and BGUI_LINEAR (smoother). (thanks to SolarLune)

3.2.2 Bugs Fixed

- Issue #25: "Image from image widget is repeated when using animated tiled textures"

3.2.3 Other Stuff

- Labels now have a fixed height based on the font instead of the individual characters in the string. This makes the default ListBoxRenderer have consistent spacing between elements.
- The default color for all four corners of a Frame are now (0, 0, 0, 0) as opposed to the blue/white gradient.

3.3 0.07

3.3.1 New Features

- An ImageButton widget has been added.
- A simple animation system has been added that allows widgets to move over time using linear interpolation.
- The Video widget can now also play the audio from a video file if play_audio=True.
- FontSize on labels is now themeable.
- Two new utility functions for TextInput: select_all() and select_none(). (thanks to jplur)
- TextInput now supports the delete key.
- Widgets now have on_mouse_enter and on_mouse_exit callbacks.

3.3.2 Bugs Fixed

- The system's size now updates if the viewport size changes.
- Issue 11: Externally resetting text input contents breaks selection. (thanks to jplur)

3.3.3 Other Stuff

- Widgets can now define various _handle_*() methods that match the callbacks. This allows subclasses to use callbacks without interfering with the user-defined ones.
- The themeing interface has been updated to simplify access to theme options.
- System is now a subclass of Widget to simplify code and reduce code duplication.
- Bgui now uses weakrefs to break dependency cycles and allow Python's GC to clean up widgets. This should solve most memory leak problems with Bgui.

3.4 0.06

3.4.1 New Features

- Multiple Image widgets can reuse the same image file for efficiency (thanks to andrew-101)
- ListBox widget
- Image.texco is now exposed allowing for UV coordinates to be changed (thanks to jplur)

3.4.2 Bugs Fixed

- Images would loose their "on_hover" when they were clicked on
- When removing a widget, that widget's cleanup method is now also run
- Issue 3: Positioning of TextBlock is off when not passing BGUI_NORMALIZED

3.4.3 Other Stuff

- Various TextInput improvements (thanks to jplur and Gomer)
- Updated demo (thanks to jplur)
- Moving or resizing a widget now affects it's children

3.5 0.05

3.5.1 New Features

- ProgressBar widget (thanks to andrew-101)
- Widgets now support sub-themes (similar to CSS classes)
- Widgets now have an aspect option to lock the aspect ratio of the widget
- Widgets can now be "frozen" with the frozen property (thanks to Kupoman)
- Themeing supoprt and color property added to FrameButton (thanks to Kupoman)
- Newline (n) support added to TextBlock widgets
- Overflow options added to TextBlock widgets (thanks to Gomer)

- Support for a "prefix" added to TextInput widgets via a prefix property (thanks to Gomer)
- on_enter callback added to TextInput widgets

3.5.2 Bugs Fixed

- BGUI now plays nice with "Show Physics Visualizations"
- Various OpenGL state bug fixes
- VRAM leaks from Image and Video widgets

3.5.3 Other Stuff

- Mouse focus is now more "intuitive" (thanks to Gomer)
- Available usable keys for TextInput expanded (thanks to Gomer)

3.6 0.04

3.6.1 New Features

- Font point sizes for Labels now scale with the screen height (1000px is the baseline). This isn't "correct" but it makes things a lot easier. This can be disabled by setting System.normalize_text = False
- TextBlock widget added for displaying multi-line text
- Image widgets now have an aspect option

3.6.2 Bugs Fixed

- ENTERKEY added to keydefs to better match Blender
- TextInput now works a little better (no negative cursor and you can input text when you have an empty string)

3.7 0.03

3.7.1 New Features

- BGUI now has themeing support (for mow info go here: http://stokes.dyndns.org/redmine/projects/bgui/wiki/Theming)

3.7.2 Bugs Fixed

- BGUI widgets could sometimes clip with scene elements

3.7.3 Other Stuff

- Widgets are now stored in OrderedDicts to allow for control over z sorting
- BGUI now uses relative imports so there are less restrictions on where the module is placed

3.8 0.02

3.8.1 New Features

- Video widget to display videos using VideoTexture (no sound support at the moment)
- TextInput widget to get text input from the user
- Frame widget to place widgets on (can also be used as a "window")
- BGUI can now handle keyboard input
- BGUI can now handle mouse states (NONE, CLICKED, RELEASE, ACTIVE)
- Widgets now support on_hover and on_release callbacks
- Widgets now have a visible attribute
- Color support added for Labels
- Alpha blending enabled for Images

3.8.2 Bugs Fixed

- Drawing labels would disable textures for images
- BGUI_DEFAULT was misspelled (was BGUI_DEFUALT)

3.8.3 Other Stuff

- BGUI now uses the bottom left as (0, 0) to match OpenGL

3.9 0.01

Initial release

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

b

O

P

R

S

T

U

V

W

Z