RubyTutorial

Tutorial for a small game using Ruby/Gosu.

Updated Oct 27, 2010 by julianraschke

Please post feedback and additions as comments to this page and visit the boards for questions outside of the scope of a single wiki page. Thank you!
http://www.libgosu.org/cgi-bin/mwf/forum.pl

Translations

For links to translations of this tutorial (Traditional Chinese, Spanish, French) see the DocsOverview page.

Source code

The code for the complete game, together with the required media files, can be found in the Gosu distribution of your choice ('examples/Tutorial.rb').

If you have installed Gosu via RubyGems, the examples are in your 'gems' folder together with the rest of the library. For example, on OS X 10.5, the examples can be found in `/Library/Ruby/Gems/1.8/gems/gosu-<version>/examples`.

If you have not installed Gosu via RubyGems, you have to copy gosu.so (or gosu.bundle, respectively) into the examples directory, then run Tutorial.rb.

If you don't have an editor that supports direct execution (TextMate, SciTE…), `cd` into the directory and run it via `ruby Tutorial.rb`.

1. Overriding Window's callbacks

The easiest way to create a complete Gosu application is to write a new class that derives from Gosu::Window (see the reference for a complete description of its interface). Here's how a minimal GameWindow class might look like:

```ruby
require 'gosu'

class GameWindow < Gosu::Window
  def initialize
    super(640, 480, false)
    self.caption = "Gosu Tutorial Game"
  end

  def update
  end

  def draw
  end
end

window = GameWindow.new
window.show
```

The constructor initializes the Gosu::Window base class. The parameters shown here create a 640x480 pixels large, non-fullscreen—that's what the "false" stands for—window. Then it changes the window's caption, which is empty until then.

update() and draw() are overrides of Gosu::Window's member functions. update() is called 60 times per second (by default) and should contain the main game logic: move objects, handle collisions, etc.

draw() is called afterwards and whenever the window needs redrawing for other reasons, and may also be skipped every other time if the FPS go too low. It should contain the code to redraw the whole screen, and no logic whatsoever.

Then follows the main program. A window is created and its show() member function is called, which does not return until the window has been closed by the user or its own code. Tada - now you have a small black window with a title of your choice!

A diagram of the main loop is shown on the WindowMainLoop page.

2. Using Images

```
class GameWindow < Gosu::Window
  def initialize
    super(640, 480, false)
    self.caption = "Gosu Tutorial Game"

    @background_image = Gosu::Image.new(self, "media/Space.png", true)
  end

  def update
  end

  def draw
    @background_image.draw(0, 0, 0);
  end
end
```

Gosu::Image#initialize takes three arguments. First, like all media resources, it is tied to a window (self). All of Gosu's resources need a Window for initialization and will hold an internal reference to that window. Second, the file name of the image file is given. The third argument specifies whether the image is to be created with hard borders. See BasicConcepts for an explanation.

As mentioned in the last lesson, the window's draw() member function is the place to draw everything, so this is the place for us to draw our background image.

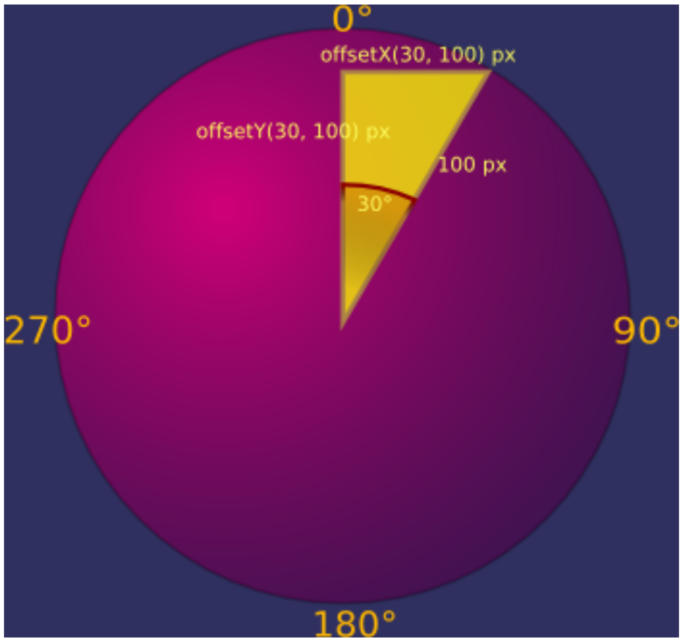
The arguments are almost obvious. The image is drawn at (0;0) - the third argument is the Z position; again, see BasicConcepts.

Player & movement

Here comes a simple player class:

```
class Player
  def initialize(window)
    @image = Gosu::Image.new(window, "media/Starfighter.bmp", false)
    @x = @y = @vel_x = @vel_y = @angle = 0.0
  end

  def warp(x, y)
    @x, @y = x, y
  end

  def turn_left
    @angle -= 4.5
  end

  def turn_right
    @angle += 4.5
  end
  def accelerate
    @vel_x += Gosu::offset_x(@angle, 0.5)
    @vel_y += Gosu::offset_y(@angle, 0.5)
  end

  def move
    @x += @vel_x
    @y += @vel_y
    @x %= 640
    @y %= 480

    @vel_x *= 0.95
    @vel_y *= 0.95
  end

  def draw
    @image.draw_rot(@x, @y, 1, @angle)
  end
end
```

There are a couple of things to say about this:



- Player#accelerate makes use of the offset_x/offset_y functions. They are similar to what some people use sin/cos for: For example, if something moved 100 pixels at an angle of 30°, it would pass offset_x(30, 100) pixels horizontally and offset_y(30, 100) pixels vertically.
- When loading BMP files, Gosu replaces 0xff00ff (fuchsia; that really ugly pink) with transparent pixels.
- Note that draw_rot puts the **center** of the image at (x; y) - **not** the upper left corner as draw does!
- The player is drawn at z=1, i.e. over the background (obviously). We'll replace these magic numbers with something better later.
- Also, see the RubyReference for all drawing methods and arguments.

Integrating Player with the Window

```
class GameWindow < Gosu::Window
  def initialize
    super(640, 480, false)
    self.caption = "Gosu Tutorial Game"

    @background_image = Gosu::Image.new(self, "media/Space.png", true)
    @player = Player.new(self)
    @player.warp(320, 240)
  end

  def update
    if button_down? Gosu::Button::KbLeft or button_down? Gosu::Button::GpLeft then
      @player.turn_left
    end
    if button_down? Gosu::Button::KbRight or button_down? Gosu::Button::GpRight then
      @player.turn_right
    end
    if button_down? Gosu::Button::KbUp or button_down? Gosu::Button::GpButton0 then
      @player.accelerate
    end
    @player.move
  end

  def draw
    @player.draw
    @background_image.draw(0, 0, 0);
  end

  def button_down(id)
    if id == Gosu::Button::KbEscape
      close
    end
  end
end
```

As you can see, we have introduced keyboard and gamepad input! Similar to update() and draw(), Gosu::Window provides two member functions button_down(id) and button_up(id) which can be overriden, and do nothing by default. We do this here to close the window when the user presses ESC. (For a list of predefined button constants, see RubyReference). While getting feedback on pushed buttons is suitable for one-time events such as UI interaction, jumping or typing, it is rather useless for actions that span several frames - for example, moving by holding buttons down. This is where the update() member function comes into play, which only calls the player's movement methods. If you run this lesson's code, you should be able to fly around!

3, Simple animations

First, we are going to get rid of the magic numbers for Z positions from now on by replacing them with the following constants:

```
module ZOrder
  Background, Stars, Player, UI = *0..3
end
```

What is an animation? A sequence of images - so we'll use Ruby's built in Arrays to store them. (For a real game, there is no way around writing some classes that fit the game's individual needs, but we'll get away with this simple solution for now.)

Let's introduce the stars which are the central object of this lesson. Stars appear out of nowhere at a random place on the screen and live their animated lives until the player collects them. The definition of the Star class is rather simple:

```
class Star
  attr_reader :x, :y

  def initialize(animation)
    @animation = animation
    @color = Gosu::Color.new(0xff000000)
    @color.red = rand(255 - 40) + 40
    @color.green = rand(255 - 40) + 40
    @color.blue = rand(255 - 40) + 40
    @x = rand * 640
    @y = rand * 480
  end
  def draw
    img = @animation[Gosu::milliseconds / 100 % @animation.size];
    img.draw(@x - img.width / 2.0, @y - img.height / 2.0,
        ZOrder::Stars, 1, 1, @color, :additive)
  end
end
```

Since we don't want each and every star to load the animation again, we can't do that in its constructor, but rather pass it in from somewhere else. (The Window will load the animation in about three paragraphs.)

To show a different frame of the stars' animation every 100 milliseconds, the time returned by Gosu::milliseconds is divided by 100 and then modulo-ed down to the number of frames. This very image is then additively drawn, centered at the star's position and modulated by a random colour we generated in the constructor.

Now let's add easy code to the player to collect away stars from an array:

```
class Player
…
  def collect_stars(stars)
    stars.reject! do |star|
      Gosu::distance(@x, @y, star.x, star.y) < 35
    end
  end
end
```

Now let's extend Window to load the animation, spawn new stars, have the player collect them and draw the remaining ones:

```
class Window < Gosu::Window
  def initialize
    super(640, 480, false)
    self.caption = "Gosu Tutorial Game"

    @background_image = Gosu::Image.new(self, "media/Space.png", true)

    @player = Player.new(self)
    @player.warp(320, 240)

    @star_anim = Gosu::Image::load_tiles(self, "media/Star.png", 25, 25, false)
    @stars = Array.new
  end

  def update
    ...
    @player.move
    @player.collect_stars(@stars)

    if rand(100) < 4 and @stars.size < 25 then
      @stars.push(Star.new(@star_anim))
    end
  end

  def draw
    @background_image.draw(0, 0, ZOrder::Background)
    @player.draw
    @stars.each { |star| star.draw }
  end
…
```

Done! You can now collect stars.

4. Text and sound

Finally, we want to draw the current score using a bitmap font and play a 'beep' sound every time the player collects a star. The Window will handle the text part, loading a font 20 pixels high:

```
class Window < Gosu::Window
  def initialize
    …
    @font = Gosu::Font.new(self, Gosu::default_font_name, 20)
  end

  …

  def draw
    @background_image.draw(0, 0, ZOrder::Background)
    @player.draw
    @stars.each { |star| star.draw }
    @font.draw("Score: #{@player.score}", 10, 10, ZOrder::UI, 1.0, 1.0, 0xffffff00)
  end
end
```

What's left for the player? Right: A counter for the score, loading the sound and playing it.

```
class Player
  attr_reader :score

  def initialize(window)
    @image = Gosu::Image.new(window, "media/Starfighter.bmp", false)
    @beep = Gosu::Sample.new(window, "media/Beep.wav")
    @x = @y = @vel_x = @vel_y = @angle = 0.0
    @score = 0
  end

  …

  def collect_stars(stars)
    stars.reject! do |star|
      if Gosu::distance(@x, @y, star.x, star.y) < 35 then
        @score += 10
        @beep.play
        true
      else
        false
      end
    end
  end
end
```

As you can see, loading and playing sound effects couldn't be easier! See the RubyReference for more powerful ways of playing back sounds - fiddle around with volume, position and pitch.

That's it! Everything else is up to your imagination. If you can't imagine how this is enough to create games, take a look at the examples on the Gosu Showcase board.

Comment by RenanTom...@gmail.com, May 08, 2009

with sudo gem install gosu, compilation fail GosuImpl?/WindowX.cpp need #include <stdio.h>

Comment by project member julianraschke, May 08, 2009

Preview gem where this is hopefully fixed: http://www.raschke.de/julian/temp/gosu-0.7.13.3.gem

Couldn't test the gem locally because 'gem install filename' didn't work on my fresh Ubuntu VM, hope there's no typos :) Out of curiosity, which function from stdio did I use?

Comment by ssscripting, Sep 09, 2009

Hi guys!

If I create a Window (under Windows XP), the mouse cursor gets hidden when it enters the window area. Is there a way to make it visible?

Comment by project member julianraschke, Sep 09, 2009

Hi ssscripting. I think a flag to show or hide the cursor is already on the To Do list. Just takes time to get there :)

Comment by ssscripting, Sep 10, 2009

Ok Julian! Thanks for a great framework.

Comment by vanakenm, Feb 14, 2010

Hi ! Thanks a lot for the framework and the comprehensive example. I was able to run it in no time.

I think advanced tutorials could be a real addition to the documentation (actually, one has to step from this one to the showcases, which can be pretty steep).

If it can help, I did made a bit of improvement on the tutorial code, allowing for a computer player with (very basic) AI. Should you be interested, let me know, I can post the code somewhere and, who knows, some explanation.

Again, thanks a lot !

Martin

Comment by project member julianraschke, Feb 15, 2010

vanakenm, I agree. I think almost too many people start out by modifying the examples, whose structure is almost TOO basic to turn into a full game …

Feel invited to open a thread of simple examples for Gosu with you making the start, I will happily stickify it :)

Comment by vanakenm, Feb 16, 2010

Ok, I'll do that. I was just wondering if the wiki was'nt a better place for this sort of example (code snippet are ok in a forum post, but example such as this one are better on a wiki page).

Comment by project member julianraschke, Feb 16, 2010

It's easier to attach files to forum postings (beware: you do it AFTER posting) :) Also, I wonder if I shouldn't move the remaining information here to the rdoc/doxygen and close the wiki, to move all the information to less places, and all on libgosu.org.

Comment by zachkluver, Aug 02, 2010

I'm having trouble with this tutorial; I can't seem to figure out why gosu won't work! I installed gosu via rubygems and it said it was installed, but it won't work. I'm not sure if it's a problem with my code, but here it is: (Oh yeah, my compiler is NetBeans? IDE 6.9)

require 'rubygems' require 'gosu'

class GameWindow? < Gosu::Window

    def initialize

        super(640, 480, false) self.caption = 'Gosu Tutorial Game' end

def update end def draw end

end

window = GameWindow?.new window.show

Thanks, Zach

p.s. Should I post this on the forums?

Enter a comment:

Wiki markup help show

Submit