Week 7 - GUI and Game Programming

Graphical Tools for Ruby

Some common GUI libraries are:

- Gosu (only for Ruby and C)
- Tcl/Tk
- FOX (FXRuby)
- Open GL

GOSU - Examples of Some features

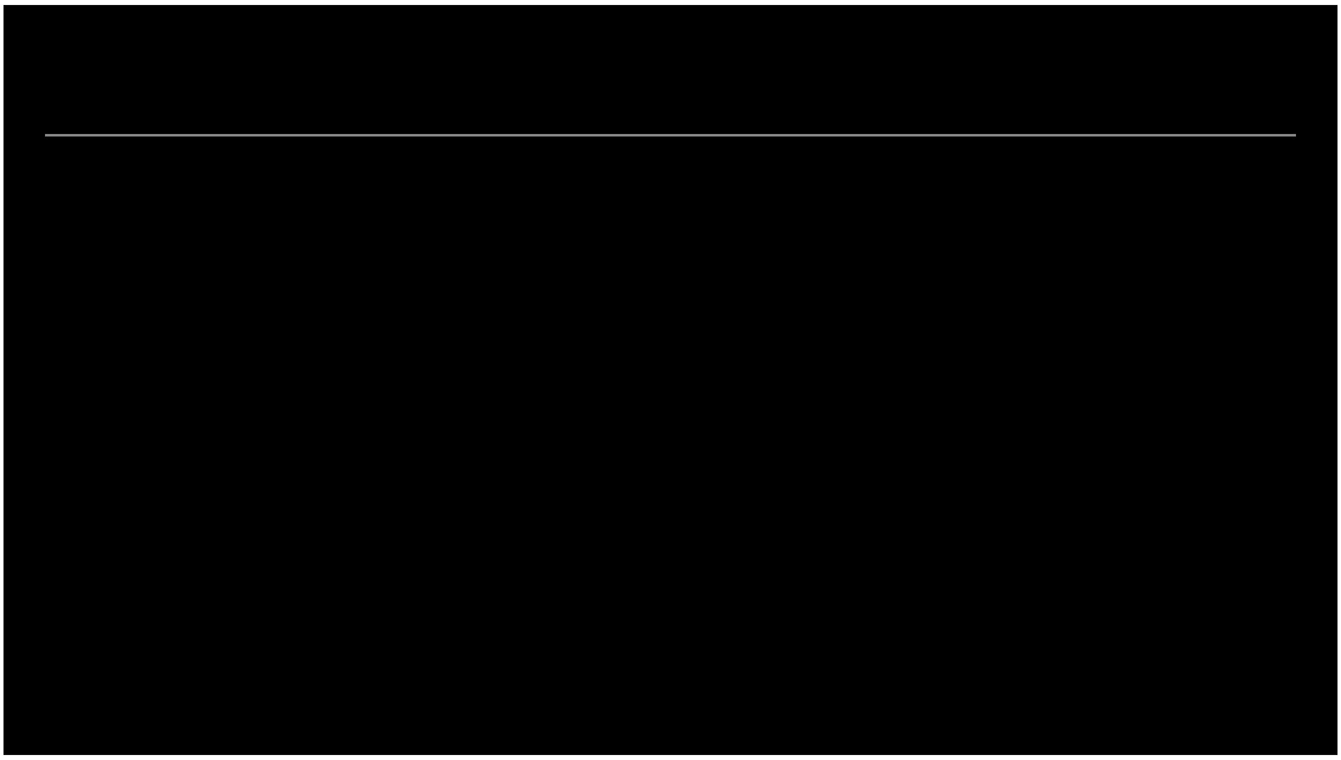
We are going to look at an example program that includes:

- Tileable images
- Cameras

Lets see the Captain Ruby example running (the code is attached below:)

CptnRuby.zip

NB: The version of Captain Ruby provided with the code for this lecture has been modified so as to be written in a more structured way (rather than Object Oriented). We look at this Structured version in the code snippets in the later slides.



Understanding the Example Code

We will look at the construction of the Captain Ruby example. The next slides will cover:

1. Splitting Arrays

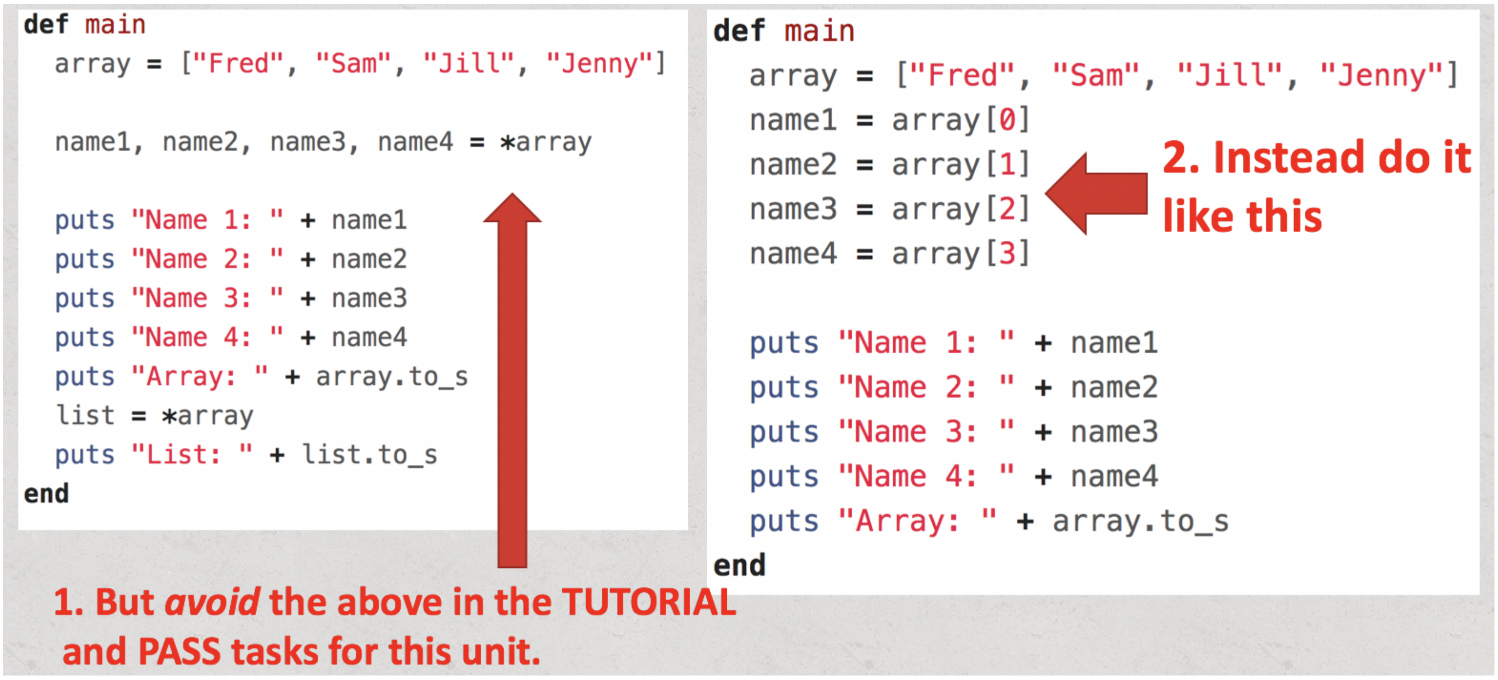
2. Some new operators

3. Tileable images

4. Creating the game terrain map

5. Camera movement

Splitting Arrays



```ruby
def main
  array = ["Fred", "Sam", "Jill", "Jenny"]

  name1, name2, name3, name4 = *array

  puts "Name 1: " + name1
  puts "Name 2: " + name2
  puts "Name 3: " + name3
  puts "Name 4: " + name4
  puts "Array: " + array.to_s
  list = *array
  puts "List: " + list.to_s
end
```

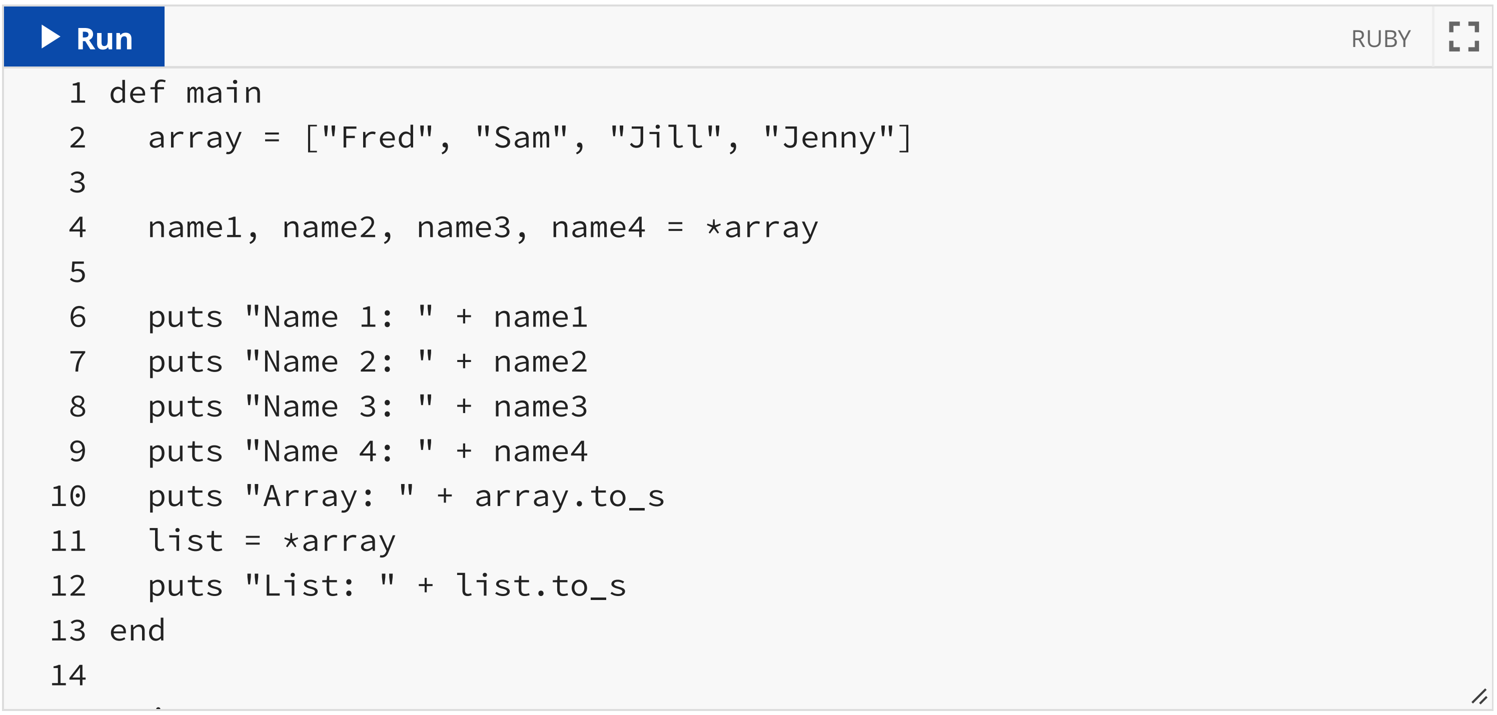
1. But *avoid* the above in the TUTORIAL and PASS tasks for this unit.

```ruby
def main
  array = ["Fred", "Sam", "Jill", "Jenny"]
  name1 = array[0]
  name2 = array[1]
  name3 = array[2]
  name4 = array[3]

  puts "Name 1: " + name1
  puts "Name 2: " + name2
  puts "Name 3: " + name3
  puts "Name 4: " + name4
  puts "Array: " + array.to_s
end
```

2. Instead do it like this

Run   RUBY

```ruby
def main
  array = ["Fred", "Sam", "Jill", "Jenny"]

  name1, name2, name3, name4 = *array

  puts "Name 1: " + name1
  puts "Name 2: " + name2
  puts "Name 3: " + name3
  puts "Name 4: " + name4
  puts "Array: " + array.to_s
  list = *array
  puts "List: " + list.to_s
end

```



```ruby
def main
  array = ["Fred", "Sam", "Jill", "Jenny"]

  name1, name2, name3, name4 = *array

  puts "Name 1: " + name1
  puts "Name 2: " + name2
  puts "Name 3: " + name3
  puts "Name 4: " + name4
  puts "Array: " + array.to_s
  list = *array
  puts "List: " + list.to_s
end

```

Digression - some other operators

%r() is a way to write a regular expression.

%w[foo, bar] is a shortcut for ["foo", "bar"].

%q() is a way to write a single-quoted string (and can be multi-line, which is useful)

%Q() gives a double-quoted string

%x() is a shell command

%i() gives an array of symbols (Ruby >= 2.0.0)

%s() turns foo into a symbol (:foo)

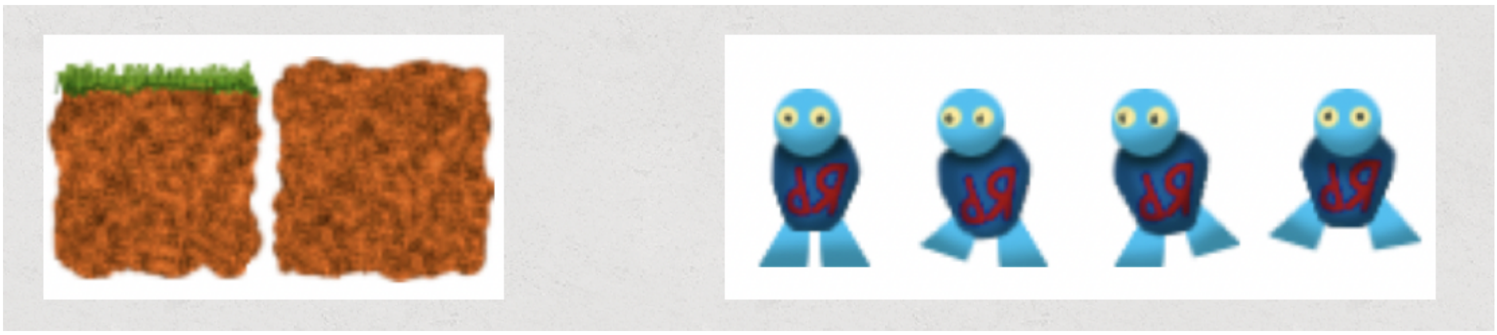
%i(a b c) # => [:a, :b, :c]

```ruby
if "fred@mydomain.com".match(/\w{1,10}\@\w{1,10}\.\w{1,10}/)
    puts("Email address")
else
    puts("Not Email address")
end

if "fred@mydomain.com".match(%r{\w{1,10}\@\w{1,10}\.\w{1,10}})
    puts("Email address")
else
    puts("Not Email address")
end

puts %w{one, two, three}

```

Source: https://stackoverflow.com/questions/1274675/what-does-warray-mean

Tileable Images (Sprite Sheets)

Two sets used in Gosu “Captain Ruby” example:



These are split up using code like the following:

```
game_map.tile_set =
  Gosu::Image.load_tiles("media/tileset.png", 60, 60, :tileable => true)

player.standing, player.walk1, player.walk2, player.jump =
         Gosu::Image.load_tiles("media/cptn_ruby.png", 50, 50)
```

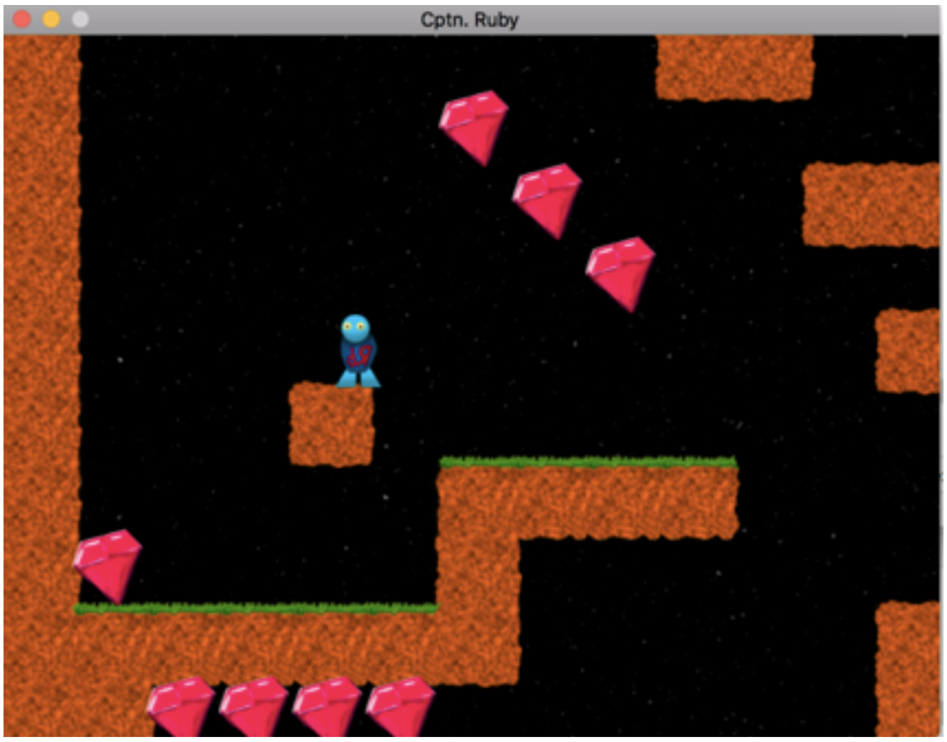
•Each call to load_tiles returns an array of tiled images. In the second case each tile is 50 x 50 pixels.

•Each element of the array contains a drawable Image.

See 'Learn Game Programming with Ruby', Chapt 5.

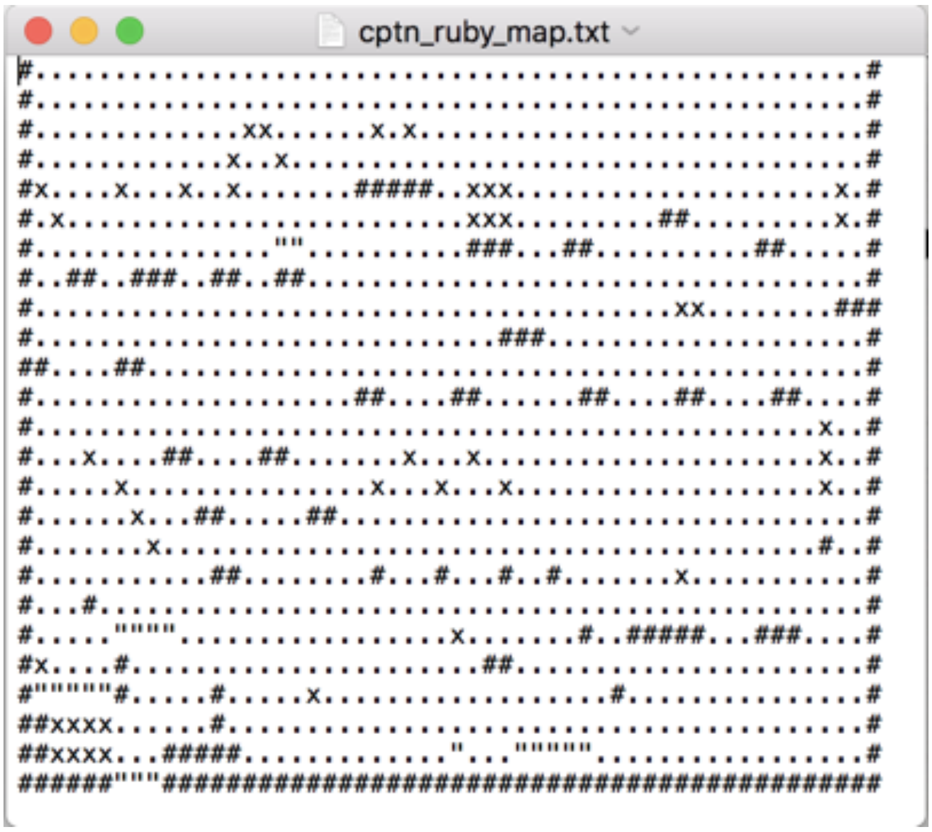
Creating the game terrain I

The terrain looks as follows, with black squares, gem squares and blocks (with or without grass):



Creating the game terrain II

The terrain is drawn based on the following text file:



```
#....................................................#
#....................................................#
#.............xx......x.x............................#
#............x..x....................................#
#x....x...x..x.......#####..xxx....................x.#
#.x.........................xxx.........##.........x.#
#...............""..........###...##..........##.....#
#..##..###..##..##...................................#
#........................................xx........###
#.............................###....................#
##....##.............................................#
#....................##....##......##....##....##....#
#.................................................x..#
#...x....##....##.......x...x.....................x..#
#.....x...............x...x...x...................x..#
#......x...##.....##.................................#
#.......x.........................................#..#
#...........##........#...#...#..#.......x...........#
#...#................................................#
#....."""".................x.......#..#####...###....#
#x....#......................##......................#
#"""""#.....#.....x..................#...............#
##xxxx......#........................................#
##xxxx...#####.............."..."""""................#
######"""#############################################
```

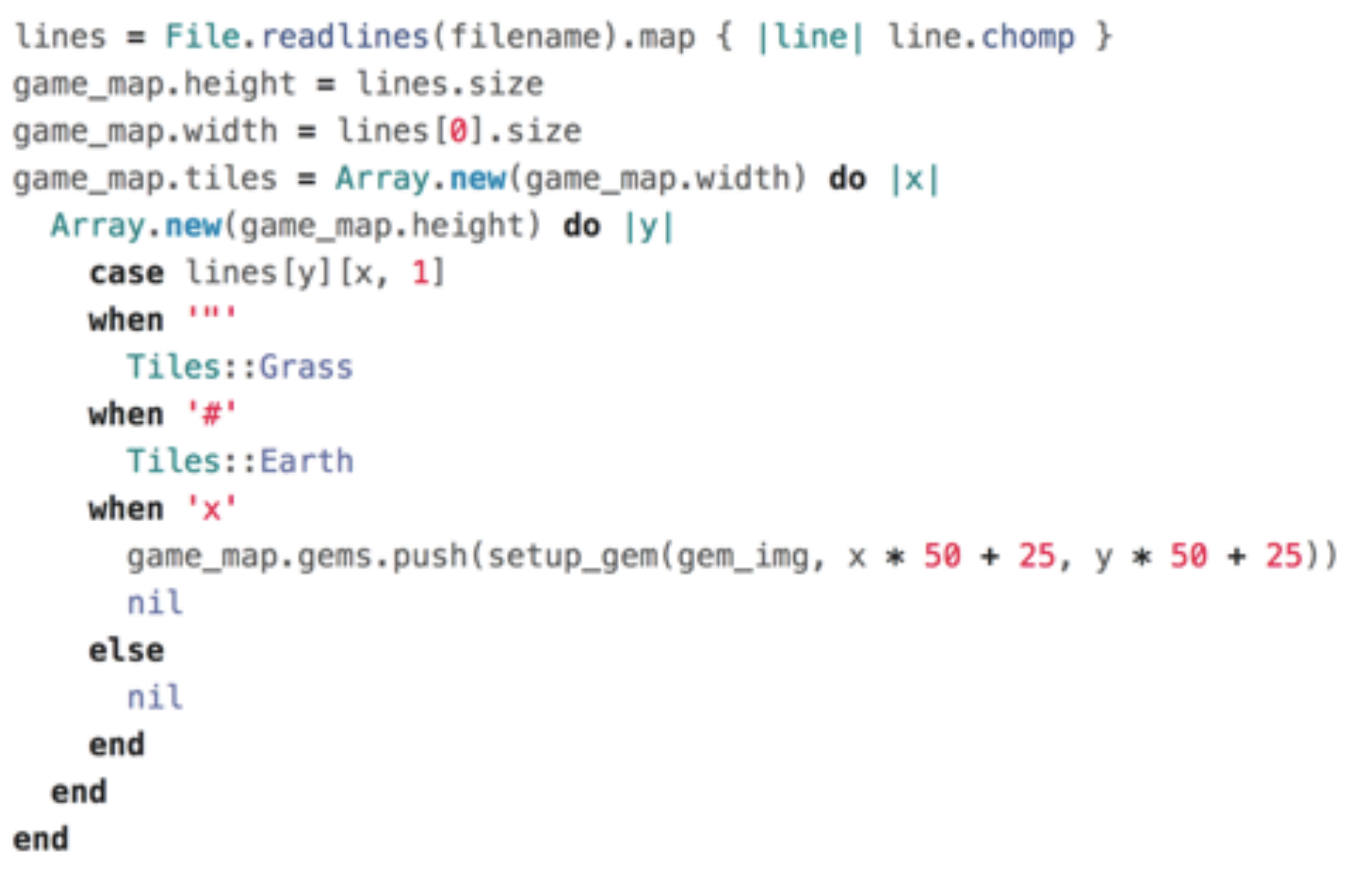
Creating the game terrain III

cptn_ruby_structured.rb

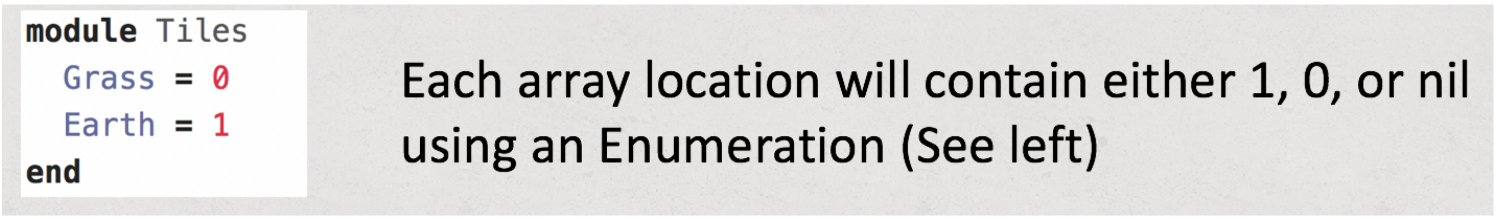
```ruby

# create an array to process:

a = [1, 2, 3, 1, 2, 3]

# create a new array of the same size and fill based on contents of 1st

final = Array.new(a.length) do |x|
  case a[x]
    when 1
      'a'
    when 2
      'b'
    when 3
```

```ruby
lines = ["one", "two", "three"]

puts lines[0][0, 1]
puts lines[0][0, 2]
puts lines[0][0, 3]

```

The following code maps the text into a 2D array:



```ruby
lines = File.readlines(filename).map { |line| line.chomp }
game_map.height = lines.size
game_map.width = lines[0].size
game_map.tiles = Array.new(game_map.width) do |x|
  Array.new(game_map.height) do |y|
    case lines[y][x, 1]
    when '"'
      Tiles::Grass
    when '#'
      Tiles::Earth
    when 'x'
      game_map.gems.push(setup_gem(gem_img, x * 50 + 25, y * 50 + 25))
      nil
    else
      nil
    end
  end
end
```



```ruby
module Tiles
  Grass = 0
  Earth = 1
end
```

Each array location will contain either 1, 0, or nil using an Enumeration (See left)

Here is that section of code:

```ruby
# game_map functions and procedures
# converted from OOP to Structured
# Note: I change the name to GameMap as the Map here is NOT the same
# one as in the standard Ruby API, which could be confusing.

def setup_game_map(filename)
  game_map = GameMap.new

  # Load 60x60 tiles, 5px overlap in all four directions.

  game_map.tile_set = Gosu::Image.load_tiles("media/tileset.png", 60, 60

  gem_img = Gosu::Image.new("media/gem.png")
  game_map.gems = []
```

Creating the game terrain IV

The one or zero in the tile array is used as an index into the tileset to determine which terrain image () is drawn:

```ruby
def draw_game_map(game_map)
  # Very primitive drawing function:
  # Draws all the tiles, some off-screen, some on-screen.
  game_map.height.times do |y|
    game_map.width.times do |x|
      tile = game_map.tiles[x][y]
      if tile
        # Draw the tile with an offset (tile images have some overlap)
        # Scrolling is implemented here just as in the game objects.
        game_map.tile_set[tile].draw(x * 50 - 5, y * 50 - 5, 0)
      end
    end
  end
  game_map.gems.each { |c| draw_gem(c) }
end
```

The gems are drawn rotating based on the current time in a wave cycle:

RUBY

```ruby
def draw_gem(gem)
  # Draw, slowly rotating
  gem.image.draw_rot(gem.x, gem.y, 0, 25 * Math.sin(Gosu.milliseconds /
end

```

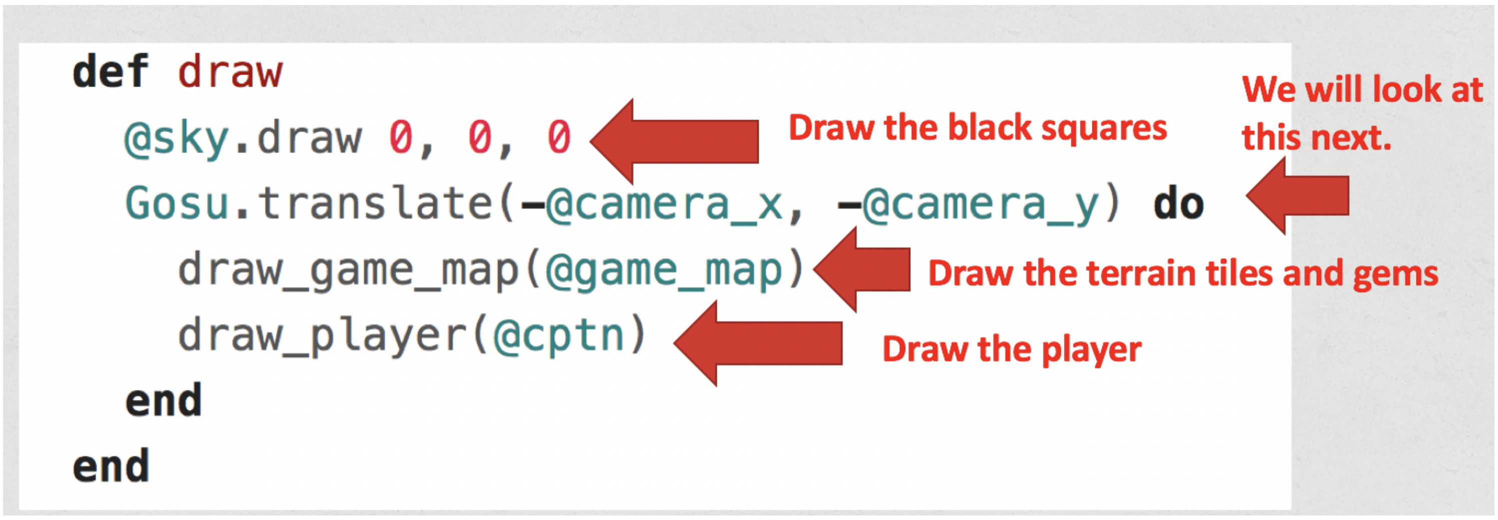
Gems are removed once a collision is detected between the gem and the player:

Run RUBY

```ruby
def collect_gems(player, gems)
  # Same as in the tutorial game.
  gems.reject! do |c|
    (c.x - player.x).abs < 50 and (c.y - player.y).abs < 50
  end
end
```

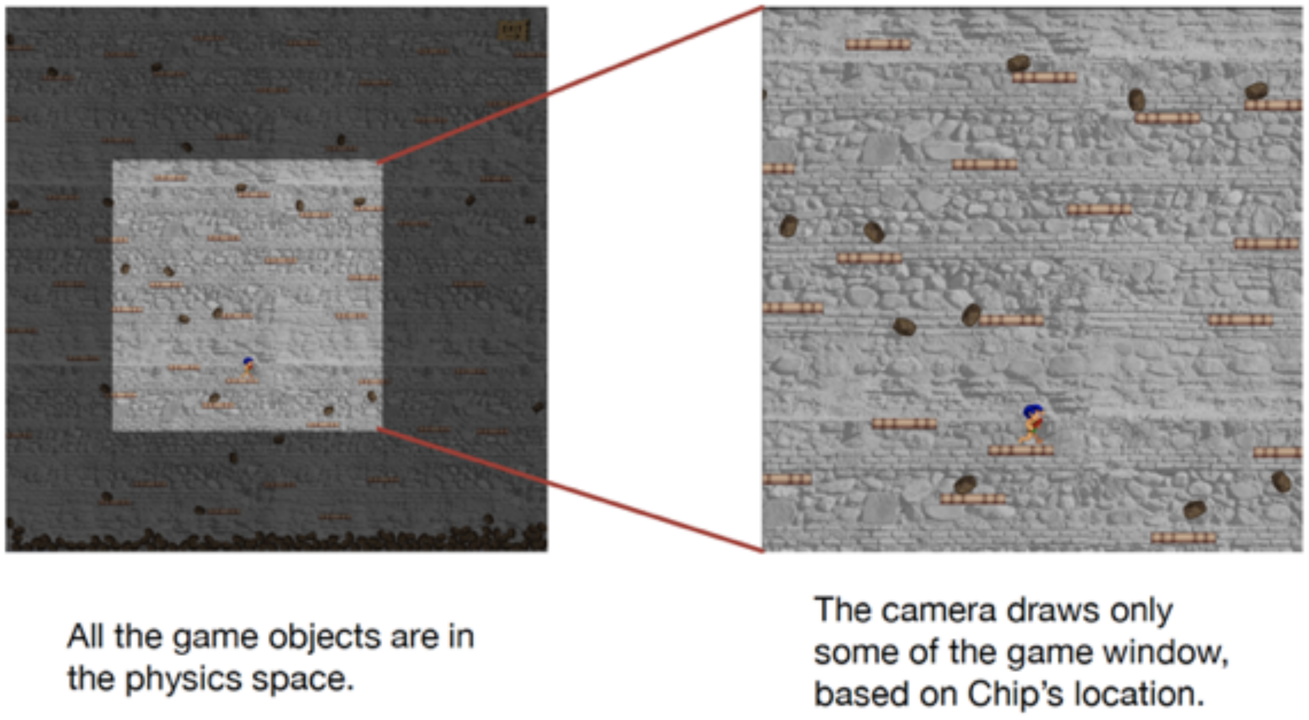
Creating the game terrain V

Thus at the top level we have:



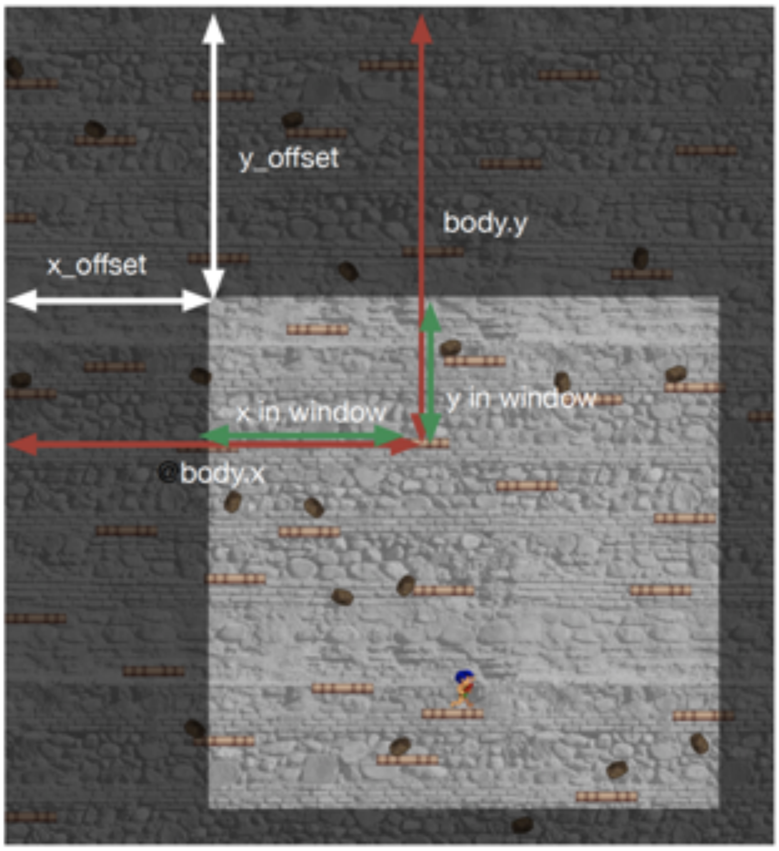
```
def draw
  @sky.draw 0, 0, 0
  Gosu.translate(-@camera_x, -@camera_y) do
    draw_game_map(@game_map)
    draw_player(@cptn)
  end
end
```

Gosu – Cameras I


All the game objects are in the physics space.
The camera draws only some of the game window, based on Chip's location.

Source: 'Learn Game Programming with Ruby', pg 158

Gosu – Cameras II



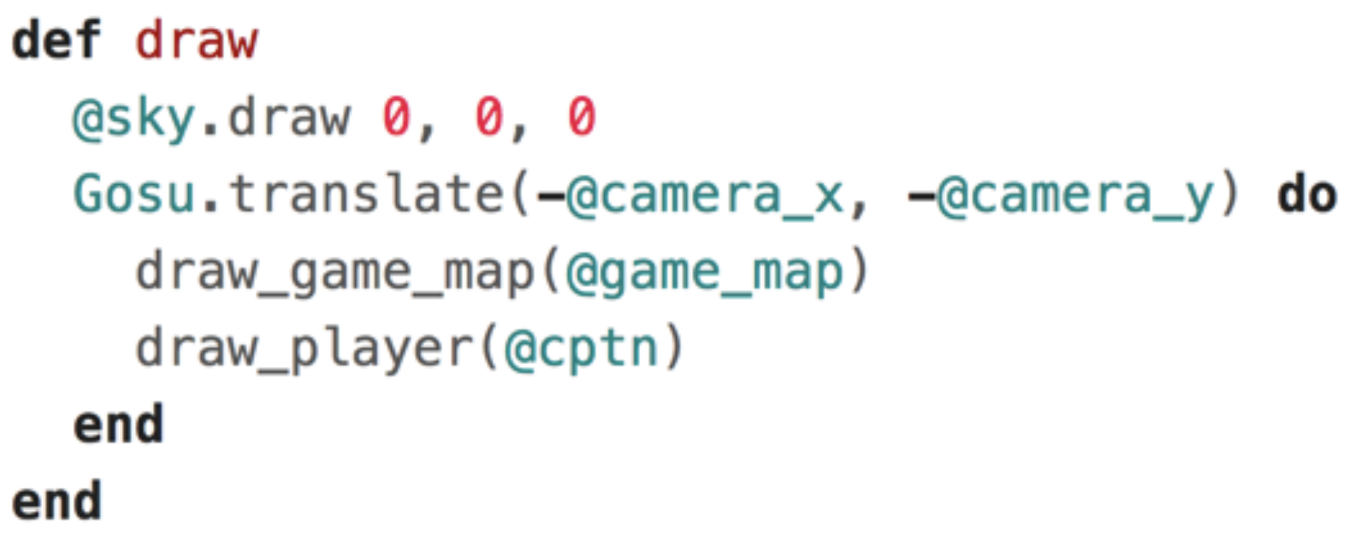
Calculating the position of an object in the window.

x = body.x - x_offset

y = body.y - y_offset

Source: 'Learn Game Programming with Ruby', pg 159

Gosu – Cameras III



```
def draw
  @sky.draw 0, 0, 0
  Gosu.translate(-@camera_x, -@camera_y) do
    draw_game_map(@game_map)
    draw_player(@cptn)
  end
end
```

`Gosu::translate()` will move the camera based on the offsets you provide.

Source: 'Learn Game Programming with Ruby', pg 159

Gosu – Sounds I

Two options:

1. Sample – a short sound that is played – perhaps as part of a game
2. Song – a longer sound file that is played – eg: for the music player.

Lets see two examples: Food Hunter and Music Player

Gosu – Sounds II: Samples

Playing sounds: Two steps

1. In the food hunter task we use the following:

```
@yuk = Gosu::Sample.new("media/Yuk.wav")
@yum = Gosu::Sample.new("media/Yum.wav")
```

2. To play the sound we simply use the following code:

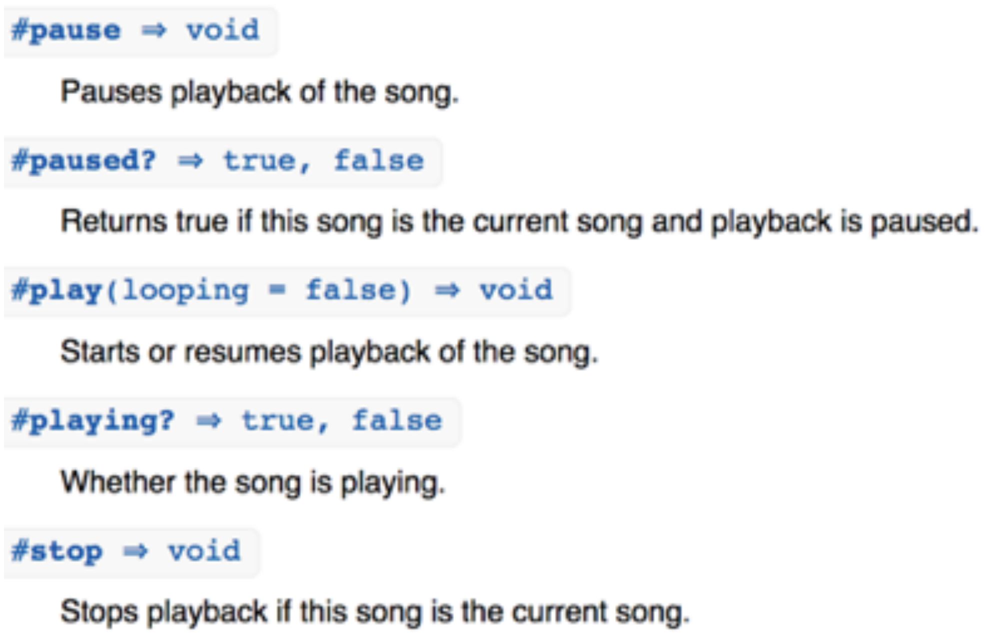
```
@yum.play
```

Gosu – Sounds III: Songs

From the Music Player Task:

```ruby
@song = Gosu::Song.new(album.tracks[track].location)
@song.play(false)
```

But you also may want to use the following:



#pause ⇒ void

Pauses playback of the song.

#paused? ⇒ true, false

Returns true if this song is the current song and playback is paused.

#play(looping = false) ⇒ void

Starts or resumes playback of the song.

#playing? ⇒ true, false

Whether the song is playing.

#stop ⇒ void

Stops playback if this song is the current song.

The TK Library

A GUI library for drawing widgets like text boxes, check boxes etc.

To install: gem install tk

A tutorial:

–https://www.tutorialspoint.com/ruby/ruby_tk_guide.htm

–For message boxes see: https://tkdocs.com/tutorial/windows.html

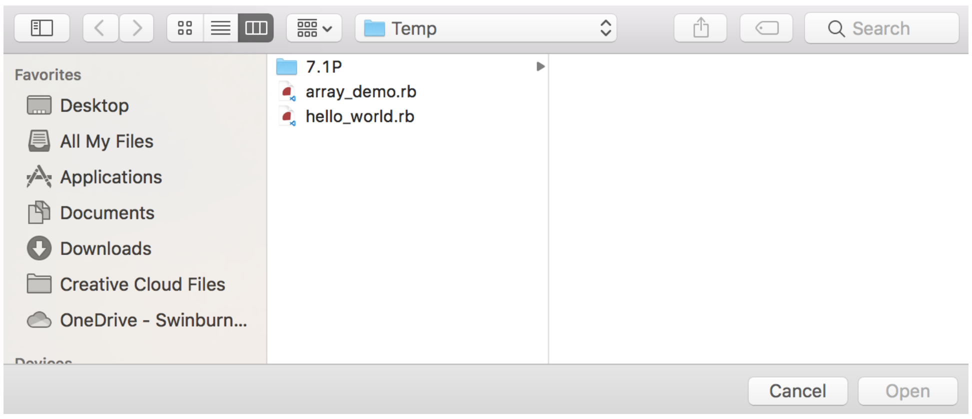
–See also: Pragmatic Programmers Guide

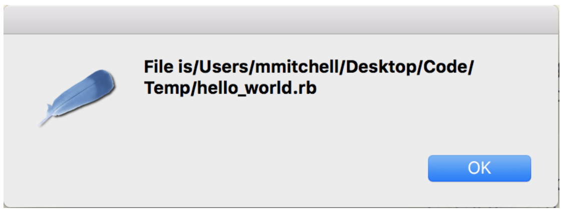
TK – Message Boxes

Lets see an example (tk_test1.rb):

tk_test1.rb

```
require 'tk'

# https://tkdocs.com/tutorial/windows.html

root = TkRoot.new
root.title = "Window"

filename = Tk::getOpenFile
Tk::messageBox :message => "File is" + filename

Tk.mainloop
```




File is/Users/mmitchell/Desktop/Code/Temp/hello_world.rb

This opens a Finder window and returns the filename selected. Others include:

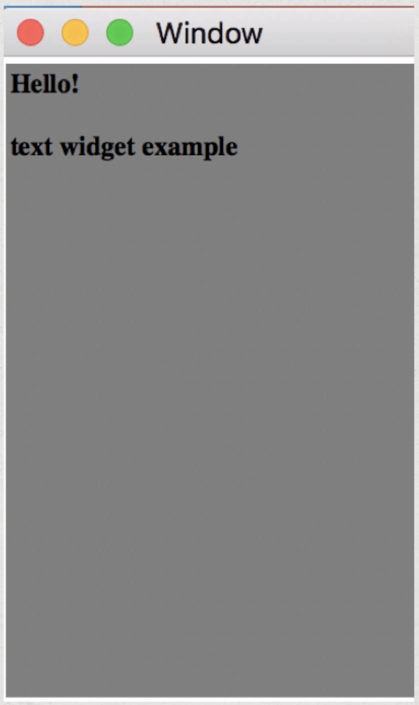
```
filename = Tk::getOpenFile
filename = Tk::getSaveFile
dirname = Tk::chooseDirectory
```

Tk – Text Boxes

Example (tk_test2.rb):

tk_test2.rb

```
require 'tk'

root = TkRoot.new
root.title = "Window"

text = TkText.new(root) do
   width 30
   height 20
   borderwidth 1
   background "gray"
   font TkFont.new('times 12 bold')
   grid('row'=>0, 'column'=>0)
end
text.insert 'end', "Hello!\n\ntext widget example"

Tk.mainloop
```



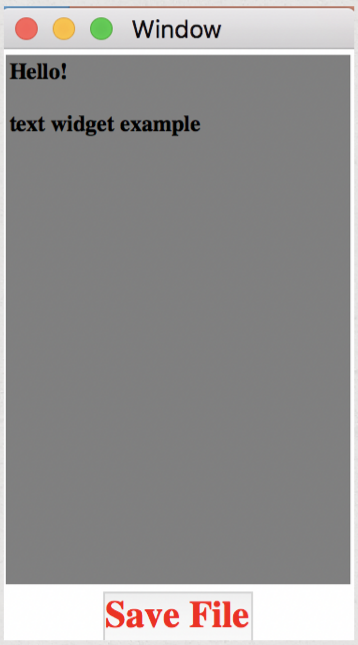
```
Hello!

text widget example
```

Tk – Button

Example (tk_test3.rb):

tk_test3.rb

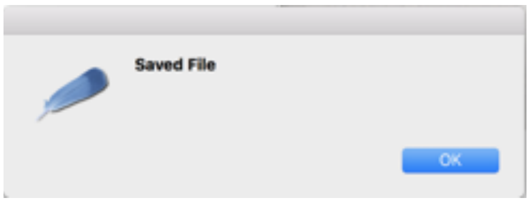
```
btn_OK = TkButton.new(root) do
   text "Save File"
   borderwidth 5
   state "normal"
   cursor "watch"
   font TkFont.new('times 20 bold')
   foreground  "red"
   activebackground "blue"
   relief      "groove"
   command (proc {Tk::messageBox :message => 'Saved File'})
   grid('row'=>2, 'column'=>0)
 end
```


Window

Hello!

text widget example

Includes a message box:


Saved File

FXRuby

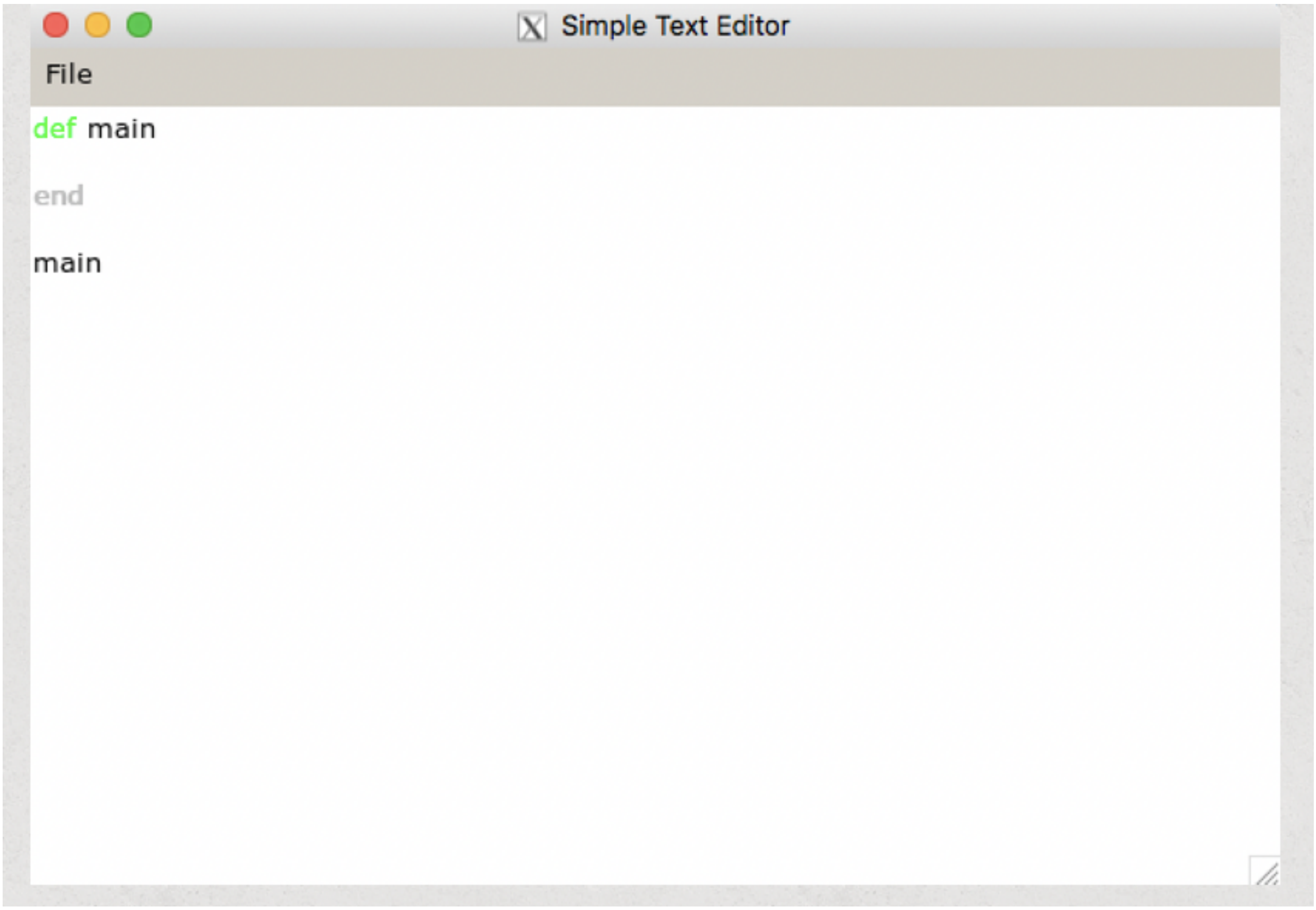
Example (texteditor.rb):

texteditor.rb

```
require 'fox16'
include Fox
```

```
app = FXApp.new
editor = TextEditor.new(app, "Simple Text Editor", 600, 400)
editor.add_menu_bar
app.create
app.run
```

This one highlights some keywords:



```
def main

end

main
```

FXRuby II

```ruby
def add_menu_bar
  menu_bar = FXMenuBar.new(self, LAYOUT_SIDE_TOP | LAYOUT_FILL_X)
  file_menu = FXMenuPane.new(self)
  FXMenuTitle.new(menu_bar, "File", :popupMenu => file_menu)
  new_cmd = FXMenuCommand.new(file_menu, "New")
  add_text_area
  load_cmd = FXMenuCommand.new(file_menu, "Load")
  load_cmd.connect(SEL_COMMAND) do
    dialog = FXFileDialog.new(self, "Load a File")
    dialog.selectMode = SELECTFILE_EXISTING
    dialog.patternList = ["All Files (*)"]
    if dialog.execute != 0
      load_file(dialog.filename)
    end
  end
  save_cmd = FXMenuCommand.new(file_menu, "Save")
```

FXRuby III

```ruby
def load_file(filename)
  contents = ""
  File.open(filename, 'r') do |f1|
    while line = f1.gets
      contents += line
    end
  end
  @txt.text = contents
  # find text only finds the first instance (seems to be a bug)
  styliseText(@txt, "def", 1)
  styliseText(@txt, "end", 2)
end
```

FXRuby IV

This is taken from the book: "FXRuby: Create Lean and Mean GUIs with Ruby”.

You could perhaps use FOX to write a Music Player:

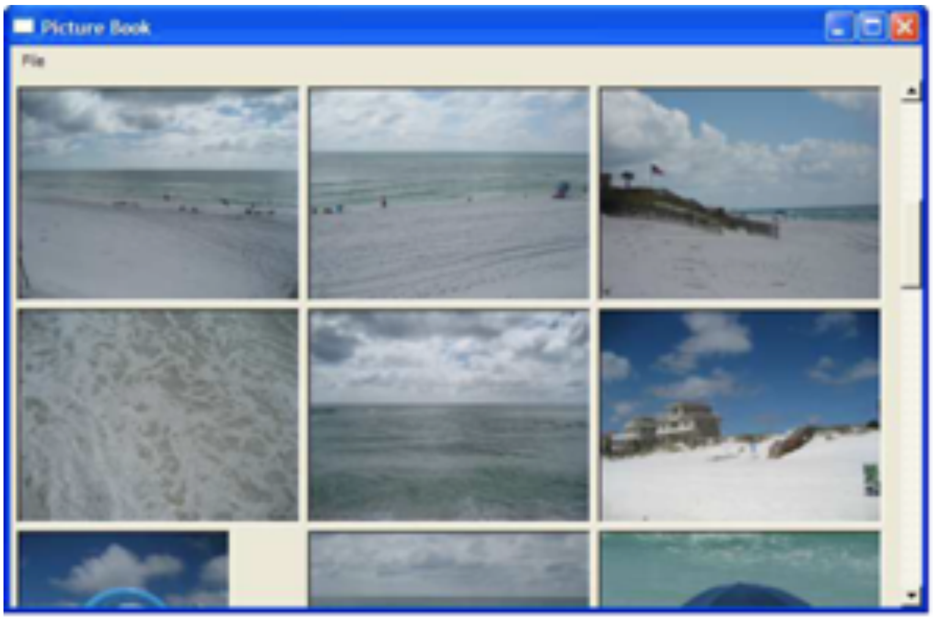


Figure 5.5: Making the album scrollable

See other FOX projects here:

http://fox-toolkit.org/projects.html

Open GL

Let us see the OpenGL site:

–https://www.opengl.org/about/

To install: gem install opengl

A (relatively simple) Tutorial:

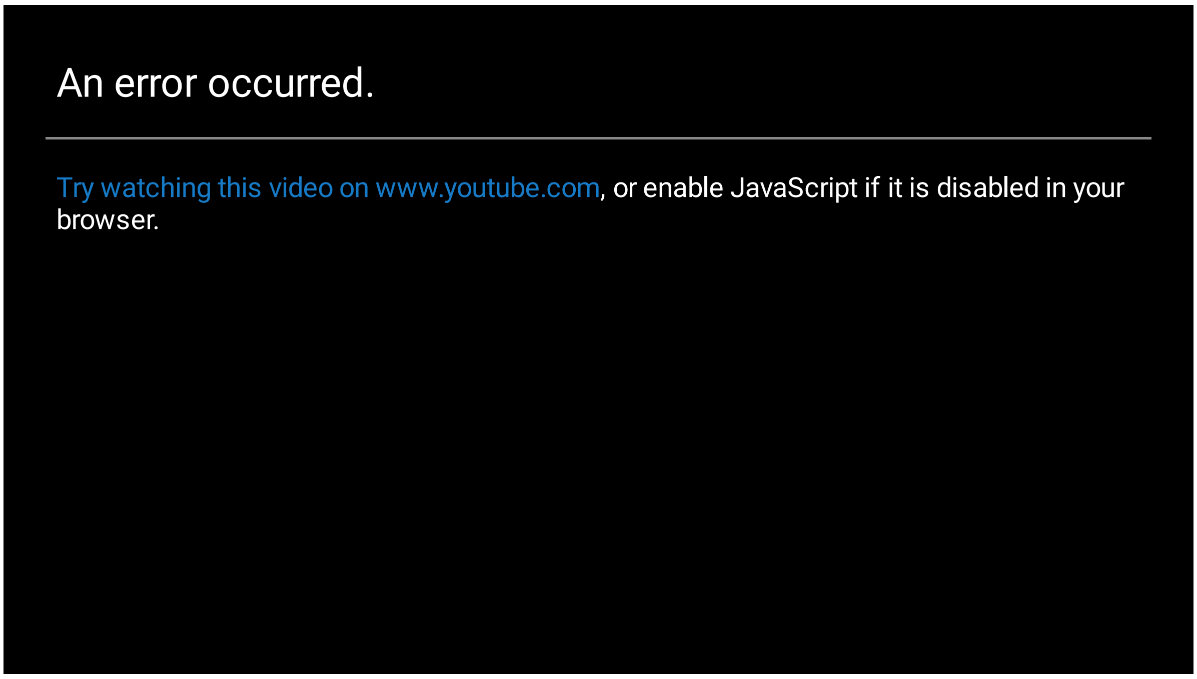
–https://www.diatomenterprises.com/different-sides-of-ruby-development-opengl/

ENTIRELY OPTIONAL FOR THIS COURSE!

Open GL

OpenGL.zip

Example (opengl_integration.rb):



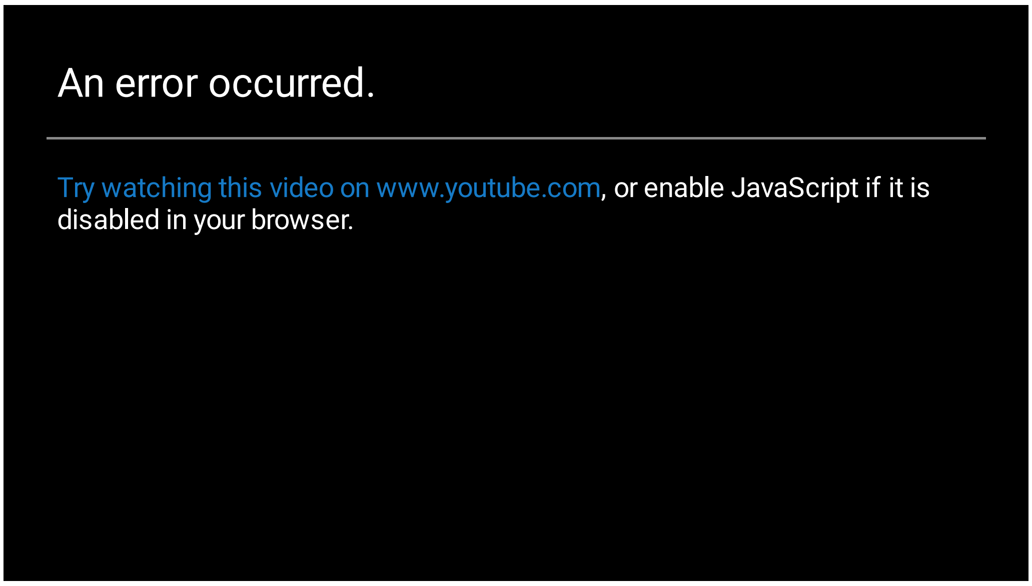
An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Food Hunter

Task 8.3D is a modification to a provided Food Hunter task which is explained in the video:

Resources.zip


An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

RUBY

```ruby
# Encoding: UTF-8

require 'rubygems'
require 'gosu'

# Create some constants for the screen width and height

# The following determines which layers things are placed on on the scr
# background is the lowest layer (drawn over by other layers), user int

module ZOrder
  BACKGROUND, FOOD, PLAYER, UI = *0..3
end

```

Summary

We looked at what can use in the Gosu library for your custom program:

- Tiles or Sprite Sheets
- Cameras
- Sounds

We looked at other libraries you might also be interested in (optional):

- Tk
- OpenGL

The Distinction Food Hunter task was also looked at.