Week 4 - Topic 6: GUI Control

GUI Control Flow

In this topic we look at the Ruby Gosu cycle.

1. Looping
2. What the methods do
3. Call backs

Looping

We have been using main as our starting point for programs.
To loop, we would write a while or repeat or similar loop within main. Eg:

Run RUBY

```ruby
def main
  finished = false
  begin
    puts 'Main Menu:'
    puts '1 To Enter or Update Album'
    puts '2 To Play existing Album'
    puts '3 Exit'
    print 'Please Enter Choice: '
    choice = gets.to_i
    case choice
    when 1
      maintain_albums
    when 3
      finished = true
```

Gosu cycle methods

In Gosu, we don’t need to write the main loop, it is implemented for us.
Instead of putting our code in main, we put it in any of:

- `draw()`
- `update()`
- or one of the callback methods (eg: `button_down(id)`)

Gosu Cycle I

Instead of putting our code in main, we put it in any of:

- `draw()`
- `update()`
- or one of the callback methods (eg: `button_down(id)`)

Most of your 'work' will go in `update()`.
`update()` and `draw()` are called in turn in a loop that continues until you stop the program.

Gosu Cycle II

Gosu Major Cycle

Modify a simple Ruby program to move a shape across the screen by modifying the tasks in the `Gosu` cycle `update()` and `draw()` methods.

Enhance the code provided as follows:

- Add a variable in the `initialize()` method called `shape_x` with the initial value of zero.
- Add code into the `update()` method that will add 10 to `shape_x`.
- Add code into the `draw()` method that will draw a shape (square or circle of any visible colour) at the *y* coordinate of 30 and *x* coordinate of `shape_x`.

Use the code provided to get started. To run it on your own machine also need to download the **media** folder with its contents.

Gosu Call Backs

- `button_down()` is a call-back method. i.e it is called by Gosu when it detects that you have pressed a button or key (as opposed to `update()` and `draw()` which are called every loop cycle

```
# the following method is called when you press a mouse
# button or key
def button_down(id)
end
```

- There is also the following, which you tend to leave unchanged (it shows the cursor):

button_down()

In `button_down()` you can evaluate the variable id to see what button or key was pressed. You also have access to two variables `mouse_x` and `mouse_y` that indicate at what position the mouse button was clicked:

These variables and the keys are defined in the Gosu documentation.

Gosu Shape Moving

Modify a simple Ruby program to move a shape across the screen by changing the provided code in the Gosu cycle `update()` method.

Use the code provided to get started.

You must enhance the code provided as follows:

1. The shape also can be moved up and down
2. The shape does not move out of the window area

Alternative to button_down()

You can also access which keys were pressed in `update()` (from the Shape Moving task):

```
def update
  if button_down?(Gosu::KbRight)
    if @shape_x != (WIDTH - SHAPE_DIM)
      @shape_x += 3
    end
  end
  if button_down?(Gosu::KbLeft) && (@shape_x != 0)
      @shape_x -= 3
  end
end
```

Gosu Task for this week

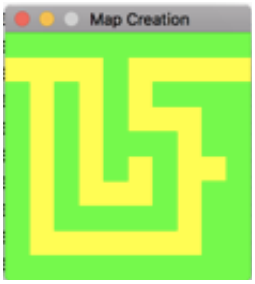
Run RUBY

```ruby
require 'gosu'

module ZOrder
  BACKGROUND, MIDDLE, TOP = *0..2
end

# Instructions:
# Add a shape_x variable in the following (similar to the cycle one)
# that is initialised to zero then incremented by 10 in update.
# Change the draw method to draw a shape (circle or square)
# (50 pixels in width and height) with a y coordinate of 30
# and an x coordinate of shape_x.
class GameWindow < Gosu::Window

```

Maze Creation

You must complete the code so that it works as follows:

1. The code in the `initialize()` procedure of the `Gosu` window should be completed to set up cells that are connected to each other with variables joining each cell to its neighbours (using references).
2. The user should be able to left click on cells on the screen to create mazes (and later in the Maze Search task we will use recursion to find a path through the maze).
3. Each cell clicked on should turn yellow.



4. Once this is working (or perhaps before) add code to print out each cell and indicate whether the reference to the neighbour on each side is nil or not, as per the following:

```
Cell x: 0, y: 0 north:0 south: 1 east: 1 west: 0
```

In this case there is no neighbour to the north or the west.

(NB: whether the east or the west neighbour is nil will depend on your perspective – i.e is your perspective looking into the screen, or out of the screen)

Your submitted screenshot should look something like the following:

Once you have the program working as required submit a screenshot to the workspace.

Sobkowicz, M 2015 *Learn game programming with Ruby : bring your ideas to life with Gosu*, The Pragmatic Bookshelf (See chapter 7 for help the grid aspect of the Maze Task)

Gosu Ruby Documentation

Gosu site

Gosu game video tutorial

Lights Example

Let's go through building a GUI program in Ruby. See the code for this lesson as attached.

 Lights.zip

Lights 1

This code slide does not have a description.

Lights 2

This code slide does not have a description.

Lights 3

This code slide does not have a description.

Lights 4

This code slide does not have a description.

Lights 5

This code slide does not have a description.

Lights 6

This code slide does not have a description.

GUI Supplement