CS61A Lab 11: User Interfaces

Week 11, 2012

In this lab, you'll be creating a Graphical User Interface (also known as a GUI) for the game of Pig. If you are able to finish the GUI in the allotted time you can get started on the project. At the end of the lab there is an introduction to turtle graphics using a partial implementation of the project.

To build the user interface, we'll be using Tkinter, which is part of Python's standard library. (Tkinter itself is a Python binding for Tk, which is an open source, cross-platform toolkit for creating graphical user interfaces)

An Introduction to User Interfaces in Tkinter

Create a new file (say, `my_ui.py`), and enter in the following program:

```
from tkinter import *

root = Frame()
root.mainloop()
```

In the first line, we're importing the `tkinter` library.
Next, we instantiate a `Frame` object - this object will serve as the main window of our UI.
Finally, to transfer control to the UI, we call the `mainloop` method on the `root`. Run it, and you'll see that a (blank) window comes up.

Not a very interesting window, I know. Patience, young grasshopper...

User Interfaces - An Event-Driven Paradigm

Unlike the Python programs we've been programming so far in 61A, UI programs aren't structured in terms of function/method calls. UI programs are instead structured around events that the user triggers through various means (mouse, keyboard, sound, etc). For instance, when you click a button in an application, that button object is 'woken up' by the UI, and the button then performs its relevant action (This is a bit reminiscent of Constraint Networks). The User Interface is constantly monitoring the window, so that whenever the user performs some action (a mouse click, or a keyboard stroke), the appropriate entities are notified.

As a concrete example, let's set up a simple example involving a Button. Modify your code in `my_ui.py` to be the following:

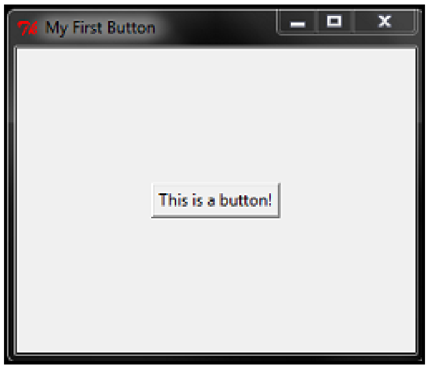
```
from tkinter import *

def handle_button_push():
    print("Button pushed.")
```

```
root = Frame()
root.grid()
root.rowconfigure(0, pad=200)
root.columnconfigure(0, pad=200)
root.master.title("My First Button")

button = Button(root, text="Press me!", command=handle_button_push)
button.grid(row=0, column=0)
root.mainloop()
```

Here, we see the interesting line: `button = Button(root, text="Press me!", command=handle_button_push)`. Here, we are creating a Button **widget**, whose **parent** is `root` (i.e. main window), whose text is `"Press me!"`, and whose **callback** function is the `handle_button_push` function.
Some terminology:

> **widget** -- A widget is a generic name for any basic element of a User Interface, such as a Button, a Frame, a Label, etc.
> **parent/child** -- The parent of a widget W is the widget that contains W. In our example above, the parent of `button` is `root` (a Frame widget).
> **callback** -- A callback is a function that is called when a certain event is triggered. For instance, if the user presses a button, then that button's callback will be invoked by the UI.

By specifying that the `button`'s parent is `root`, Tkinter knew to 'add' the `button` widget to the `root` Frame. Try running the program, and you'll see your first simple application. What happens when you click the button?



Our First Simple UI

Designing a Pig GUI

Now, let's get started on building a GUI for the game of Pig!

Grab the provided code

Copy the provided skeleton framework into your current directory by doing:

```
cp -r ~cs61a/lib/lab/lab11 .
```

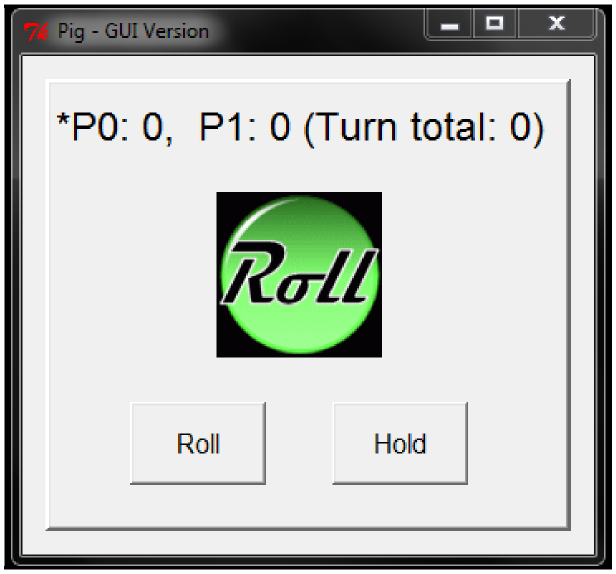
Note the `.` (period) as the third parameter to `cp`.

Take a second and read through the code - much of it should be understandable to you at this point (i.e. the Pig game logic).

Try running it right now!

```
    python3 pig_gui.py
```

You'll notice that a simple GUI pops up:



...but nothing works. Here's where you come in!

1.) Add the 'Roll' and 'Hold' Button callbacks

Right now, the Roll/Hold Buttons don't do anything when you click them. This is because they don't have any callbacks registered. Modify the `PigGUI.initialize_ui` method to register each button with `handle_roll_click` and `handle_hold_click` as callback functions. You should consider using the provided `register_callback` function.

Then, fill in the body of `handle_roll_click` and `handle_hold_click`. You should use the provided methods `PigGUI.roll`, `PigGUI.hold`.

You'll know you did this question correctly if the PigState is being printed out in the terminal at each Button click.

2.) Update the UI Game State

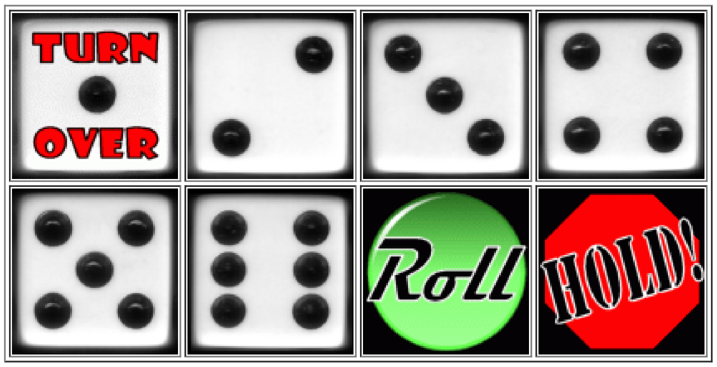
Rolls and holds are now happening behind the scenes. However, these changes aren't being reflected on the UI. Bummer!

Modify the `PigGUI.roll` and `PigGUI.hold` methods to also update the Label widget that displays the game state information: `self.state_label`.

You'll want to use the provided `set_widget_label` function.

3.) Add Images Functionality

You might notice that we have some nice `.gif` images living inside of `lab11/images/`:



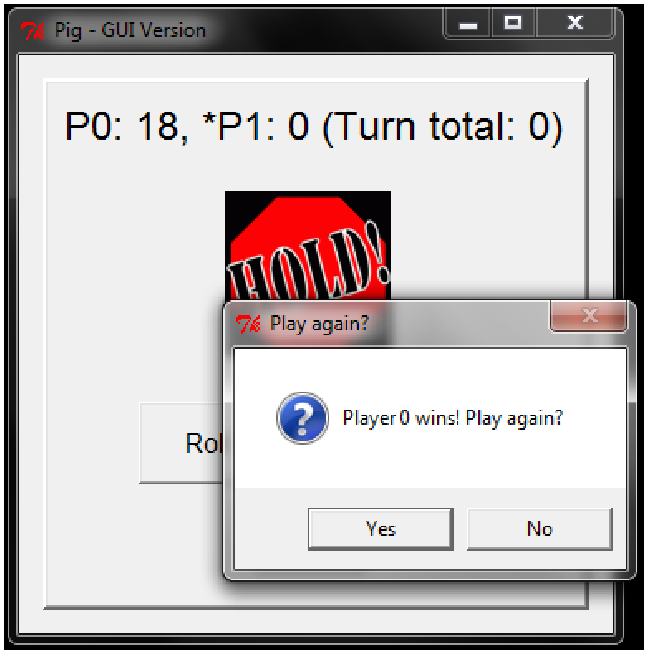
Well, we're going to use them!

Add the necessary modifications so that, on each dice roll, the die value is reflected on the screen (i.e. rolling a 4 should result in `images/die4.gif` being displayed). In addition, performing a 'hold' action should result in `images/hold.gif` being displayed.

Tip: A good amount of the work has been done inside of `PigGUI.set_dice_image`. Your job is to figure out how to use this nice function to achieve your goal.

4.) Improve the 'Game Over' experience

At the moment, the end of a game is a rather lack-luster experience for the winner - all that happens is an AssertionError is raised on the terminal. Let's do better!
Add functionality such that, as soon as one of the players wins, a MessageBox pops up with a message saying which Player won, and two options: one to restart the game, and another to exit:



P0: 18, *P1: 0 (Turn total: 0)

Player 0 wins! Play again?

To get the MessageBox to pop up, we'll be using the Tkinter `askyesno` provided function:

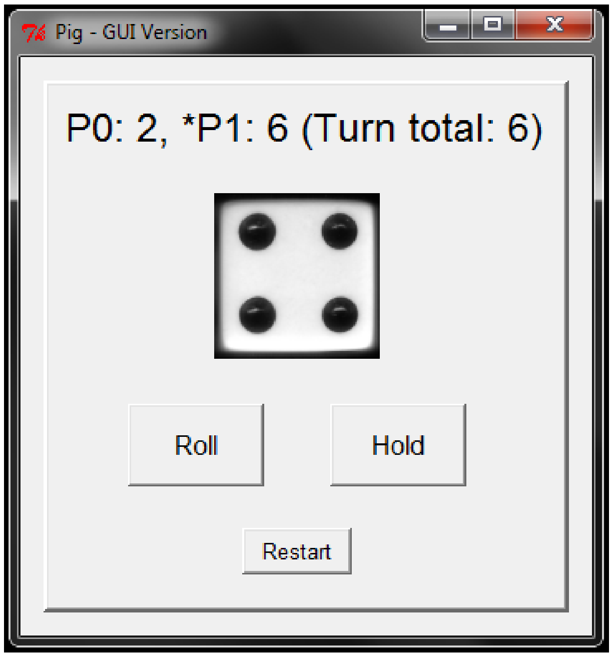
```
from tkinter import *
from tkinter.messagebox import askyesno

response = askyesno("Important Question", "Do you like CS 61A?")
if response:
    print("Hooray, you like 61A!")
else:
    print("Hm, we'll convince you yet!")
```

Calling the `askyesno` function creates a Yes/No MessageBox, and returns `True` if the user clicked 'Yes', and `False` if the user clicked 'No'. Try it out!

Tip: To exit the application, call `self.master.destroy()`, or simply do `exit()`. The former is the preferred, cleaner way of exiting a Tkinter application, and the latter is a general way to terminate a Python program.

5.) Add a 'Restart' Functionality

At this point, we have a pretty complete UI for the Pig game - awesome! However, as always, we can do a little better. It would be nice to be able to restart the game at whim. For instance, you might want to start over if the current game is taking too long, or you want to play against someone else, or you're losing too badly, etc... Add the ability for a player to restart the current game - this will have you add a new Button to the `PigGUI`, and a new callback.



Here's what your completed UI might look like

Congratulations! You've created your first GUI Application!

Project 4 - A Scheme Interpreter In Python

If you haven't already copied the project files over to your account, use *one* of the following command (sequences) to do so, modified to your taste.

```
mkdir proj4
cp -r ~cs61a/public_html/sp12/projects/scheme/* proj4
```

```
unzip ~cs61a/public_html/sp12/projects/scheme/scheme.zip
mv -f scheme proj4
rm scheme.zip
```

```
wget http://inst.eecs.berkeley.edu/~cs61a/sp12/projects/scheme/scheme.zip
unzip scheme.zip
mv -f scheme proj4
rm scheme.zip
```

Great! Now that you're all ready to start the project, let's get a taste of what you can do once you've implemented part of it. There is a completed version of the interpreter you can toy around with by typing `staff-proj4` into the shell. Enter the following statements into the resulting scheme prompt and be (very slowly) amazed:

```
(define (spiral deg) (if (= deg 0) (penup) (begin (forward (- 100 deg)) (left 20) (spiral (- deg 1)))))
(pendown)
(color 'red)
(spiral 100)
```

Neat, huh? And that's just the tip of the iceberg... so don't waste too much time playing with this, it's much more fulfilling (and the drawing is much faster) when you can create recursive images using your own project.