Today in CS 42

HW9 (due Monday): Last Python

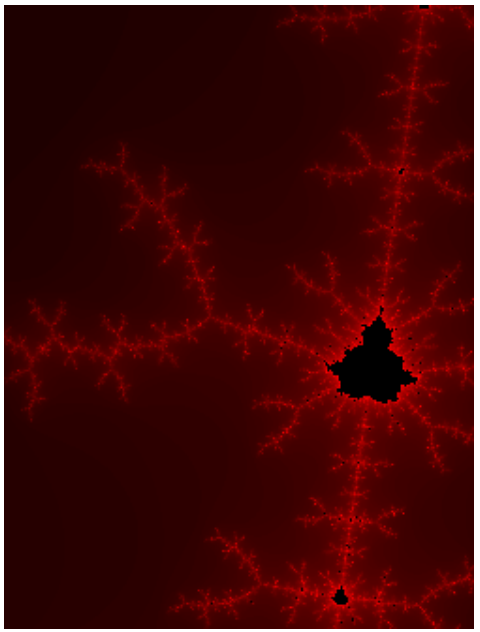
Python graphics with tkinter


```
class App:
  def __init__(self, master):
    frame = Frame(master)
    frame.pack()
    self.button = Button(frame, text="QUIT", fg="red", command=frame.quit)
    self.button.pack(side=LEFT)
    self.hi_there = Button(frame, text="Hello", command=self.say_hi)
    self.hi_there.pack(side=LEFT)

  def say_hi(self):
    print "hi there, everyone!"
```

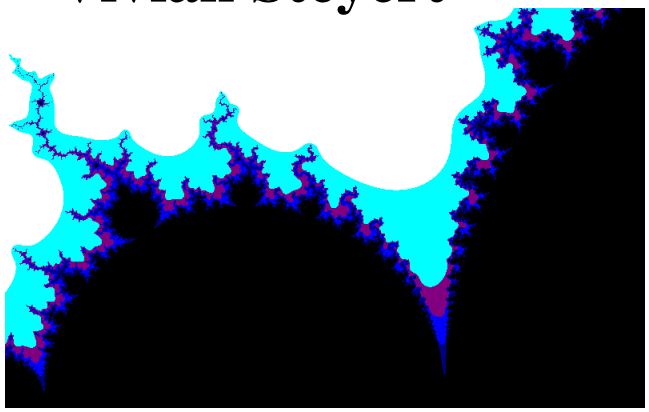
Event-driven programming and the return of functions as first-class citizens

Mset hall of fame

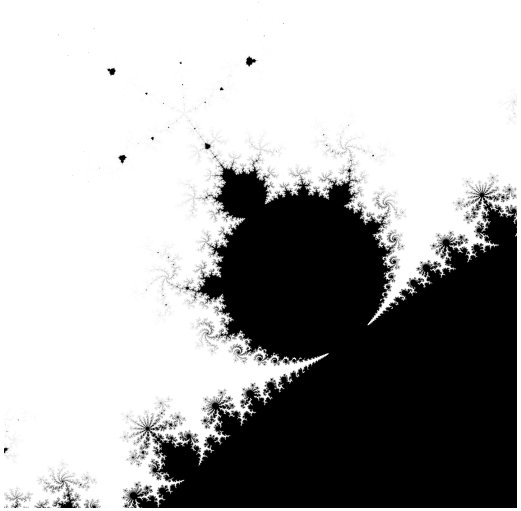
Ari Hausman-Cohen



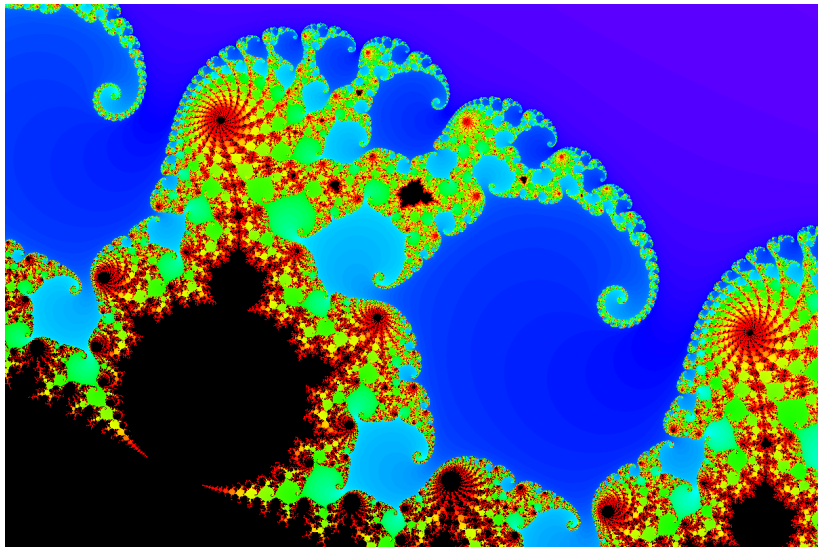
Vivian Steyert



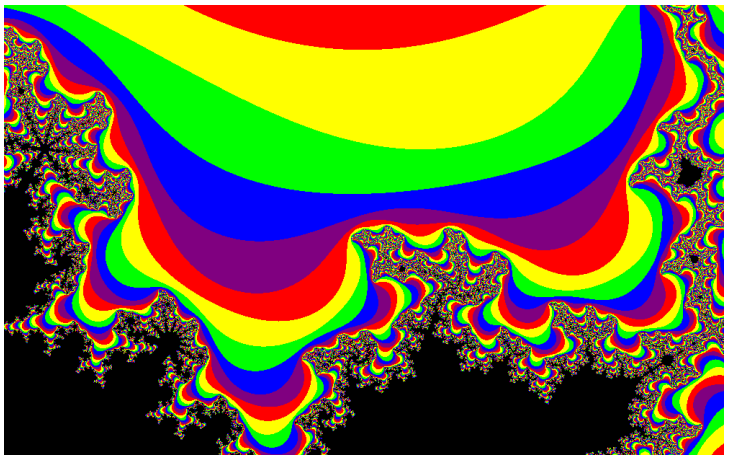
Gourav Khadge



Matt Lam



Colin Stanfill

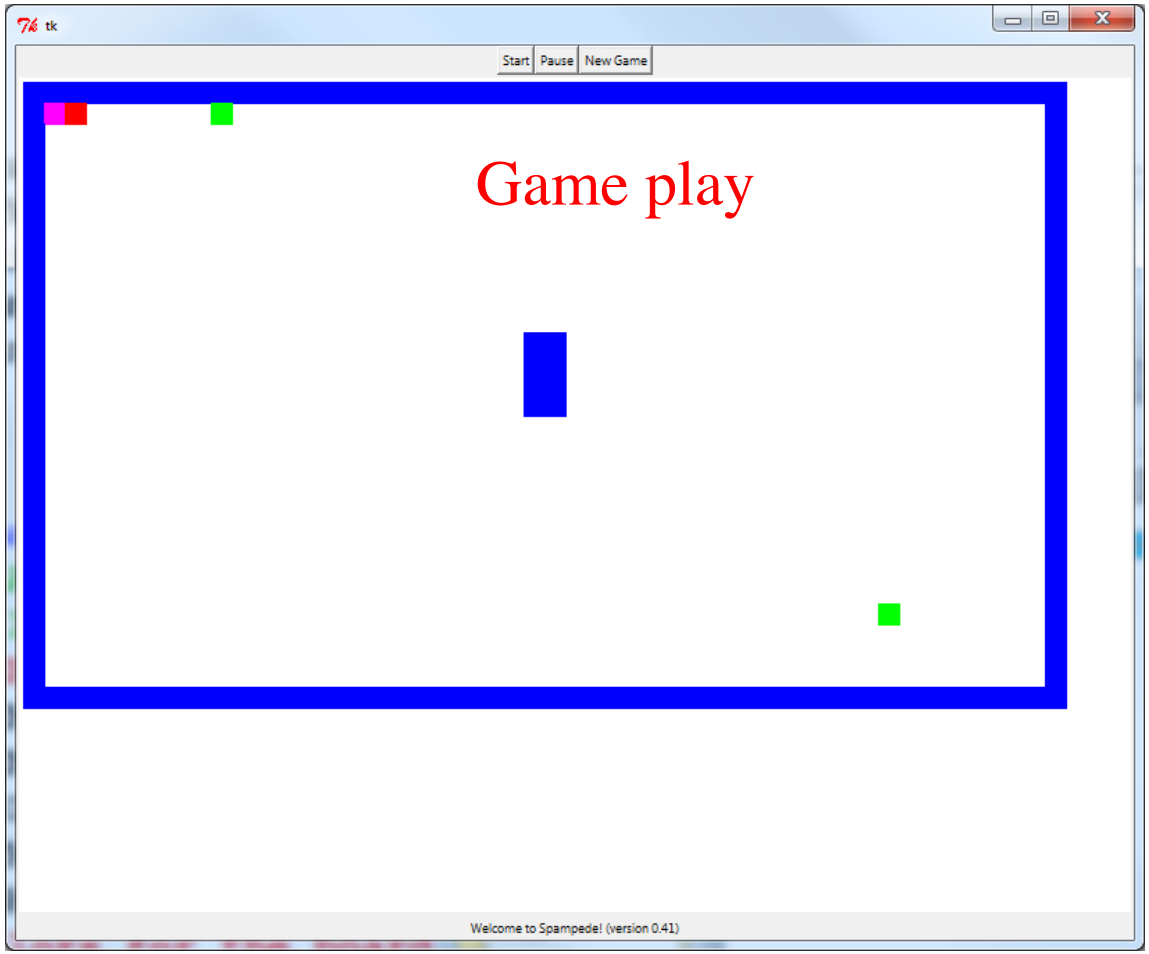


Designing a GUI in Python

```
import Tkinter
tk = Tkinter
```

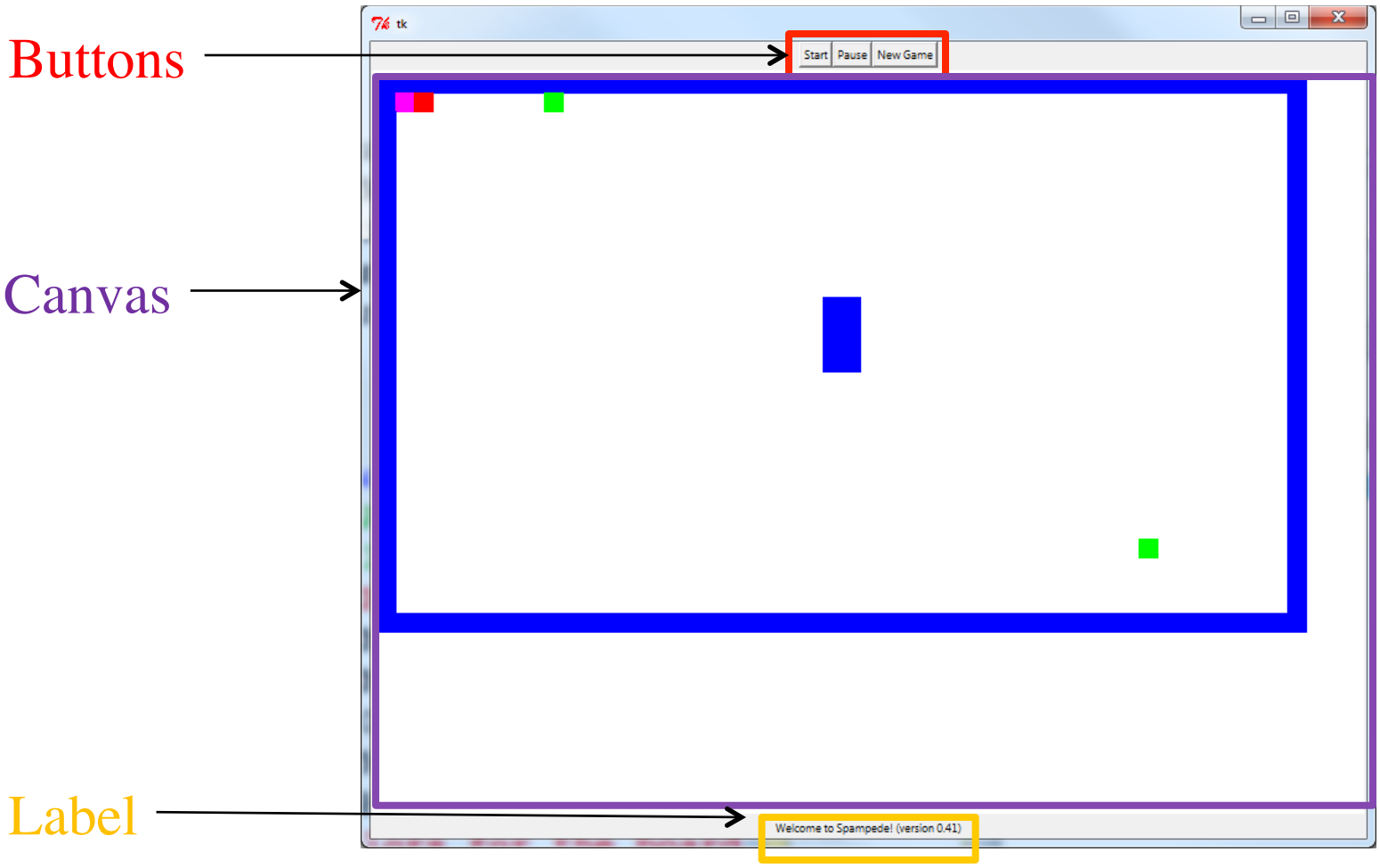
Responding to user input

GUI layout
+ drawing
the game board



Designing a GUI in Python

GUI layout



Overview

```
import Tkinter
tk = Tkinter

def main():
    ''' Start the game. '''
    root = tk.Tk()          # Create the main window

    app = Spampede(root) # Create a Spampede object


    root.mainloop()         # Give the main window control
```

Laying out the Window

```
class Spampede(object):
    ''' A class that implements the game of Spampede
        using the tkinter library.'''
    PADDING = 5    # The space between the edge of the
                   # canvas and the board
    WIDTH = 800    # The width of the gameboard
    HEIGHT = 600   # The height of the gameboard
    # Game colors
    BGCOLOR = "white"
```

Laying out the Window

```
def __init__(self, master):       # master is the main window
    ''' Initialize the Spampede application '''
    frame = tk.Frame(master)    # Create the main frame, inside master
    frame.pack()       # Pack it into the master window

    buttonFrame = tk.Frame(frame)  # Create a frame for buttons
    buttonFrame.pack(side=tk.TOP)  # Put that frame at the top of
                                   # the main frame

    startb = tk.Button(buttonFrame, text="Start", command=self._go)
    startb.pack(side=tk.LEFT)
    pauseb = tk.Button(buttonFrame, text="Pause", command=self._pause)
    pauseb.pack(side=tk.LEFT)
    resetb = tk.Button(buttonFrame, text="New Game", command=self._reset)
    resetb.pack(side=tk.LEFT)

    self.canvas = tk.Canvas(frame, width=self.WIDTH, height=self.HEIGHT,
                            bg=self.BGCOLOR)
    self.canvas.pack(fill=tk.BOTH)

    self.message = "Welcome to Spampede v0.42!"
    self.messageLabel = tk.Label(frame, text=self.message)
    self.messageLabel.pack(side=tk.BOTTOM)
```

Frames, Buttons, Canvases, oh my!

```
import Tkinter
tk = Tkinter

# More code here not shown…
# then, in __init__:

    buttonFrame = tk.Frame(frame)  # Create a frame for buttons

    buttonFrame.pack(side=tk.TOP)  # Put that frame at the top of
                                   # the main frame

    startb = tk.Button(buttonFrame, text="Start", command=self._go)

    startb.pack(side=tk.LEFT)
```

using tkinter (which we've named tk)

Create a new Frame object to hold the buttons

And put this new frame in the main frame

and put it in the button frame

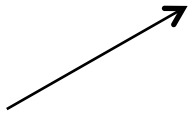
Create a new Button to be the start button

and set some of its attributes

pack?
attributes?

Attributes of graphical objects

```
import Tkinter
tk = Tkinter

# More code here not shown…
# then, in __init__:

    buttonFrame = tk.Frame(frame)  # Create a frame for buttons
    buttonFrame.pack(side=tk.TOP)  # Put that frame at the top of
                                   # the main frame
    startb = tk.Button(buttonFrame, text="Start", command=self._go)
```



A widget's parent is always the first argument

But how do you know what you can specify here?
Answer: Look it up.

http://effbot.org/tkinterbook/button.htm

```
    startb.pack(side=tk.LEFT)
```

you can also change attributes later using config, e.g.

```
self.messageLabel.config(text=myNewText)
```

The pack geometry manager

```
import Tkinter
tk = Tkinter

# More code here not shown…
# then, in __init__:

    buttonFrame = tk.Frame(frame)   # Create a frame for buttons
    buttonFrame.pack(side=tk.TOP)   # Put that frame at the top of
                                    # the main frame
```

pack tells tkinter to show the item in the frame.
Optional arguments: side (values: TOP, BOTTOM, LEFT, RIGHT, CENTER)
fill (values: X, Y, BOTH) – see canvas for example

```
    startb = tk.Button(buttonFrame, text="Start", command=self._go)
    startb.pack(side=tk.LEFT)
```

There are other geometry managers (e.g., grid), but pack is the simplest.

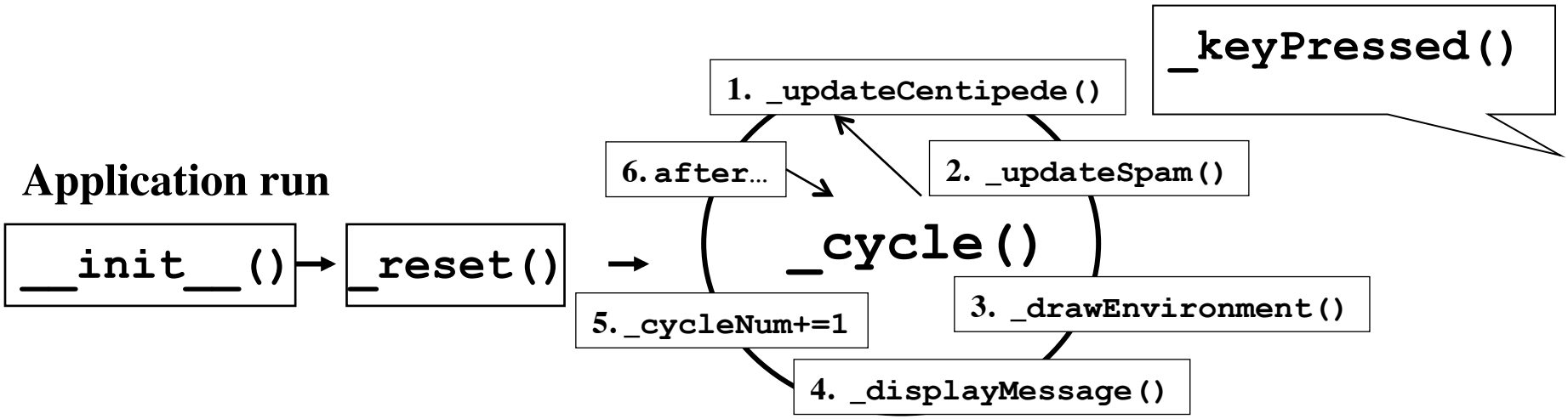
http://effbot.org/tkinterbook/pack.htm

The **Canvas** hosts the game

```
# Create the canvas object and save it.  This is where you will
# draw all of the components of the gameboard.
self.canvas = tk.Canvas(frame, width=self.WIDTH,
                        height=self.HEIGHT, bg=self.BGCOLOR)

self.canvas.pack(fill=tk.BOTH)
```

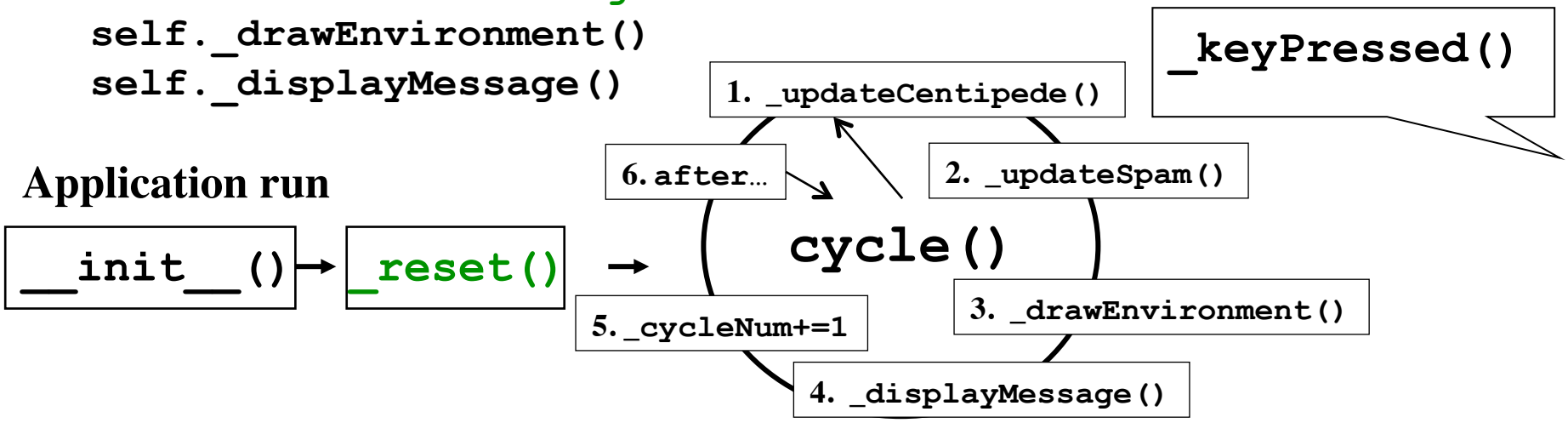
Try it! Download the Spampede starter file and play around with it. Change attributes of objects, add buttons, etc. Or create your own new GUI from scratch.

Game control overview



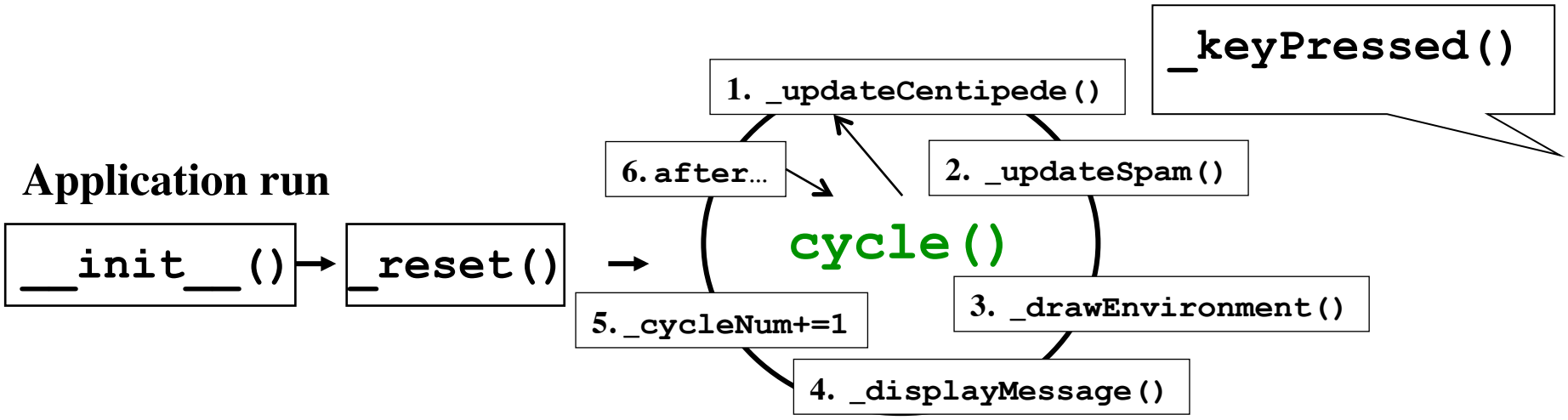
Game control details

```
def _reset(self):
    ''' This is the function that will run when the user
      presses the New Game button.  Resets the game.'''
    #set up the maze here
    self.theMaze = SpamMaze()

    #You may also want to initialize the pede's direction here
    #and to other initializations that need to happen each time
    # a new game is started.

    self.currentColor = "green"
    self._drawEnvironment()
    self._displayMessage()
```



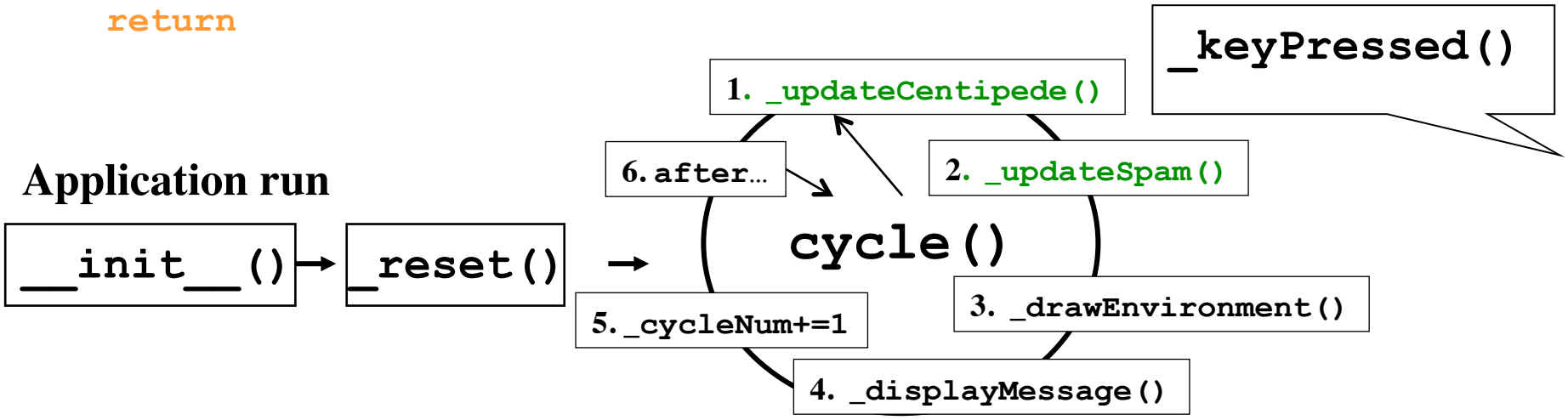
Game control details

```
def _cycle(self):
    ''' This is the main "loop" that controls the gameplay. '''
    if self.running:
        self._updateCentipede()    # update the Spampede deque
        self._updateSpam()         # update the Spam deque
        self._drawEnvironment()    # draw things to buffer
        self._displayMessage()     # display messages
        self.cycleNum += 1         # add a cycle
        # Wait a little while and then cycle again.
        self.canvas.after(self.sleepTime, self._cycle)
```



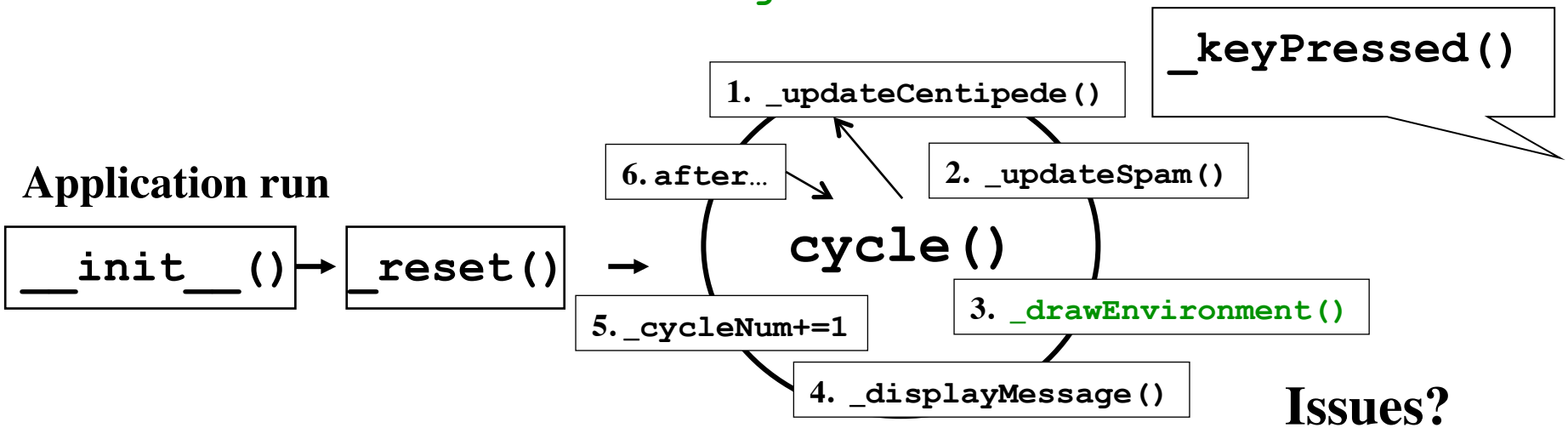
Game control details

```
# You might use this method to move the centipede one square
# Also, this method can check if the centipede runs
# into a can of spam, a wall, itself, etc. and act appropriately.
def _updateCentipede(self):
    ''' Update the centipede.  You don't have to do this
        every cycle if you want a slower game. '''
    return

# You might use this method to add/delete spam cans periodically
def _updateSpam(self):
    ''' Possibly add or remove spam.  You won't want to do this
        every cycle. '''
    return
```



Game control details

```
def _drawEnvironment(self):
    ''' Draw the state of the game on the board.
        Note that (0,0) is in the top left corner of the board
        and +y is down.'''
    self._clear()
    r3 = self.canvas.create_rectangle(40,50,100,100)
    self.canvas.itemconfig(r3, fill="red")
    r2 = self.canvas.create_rectangle(250, 100, 350, 200,
                                      fill=self.currentColor)
    if (self.cycleNum+1) % 42 == 0:
        self.currentColor = "magenta"
```



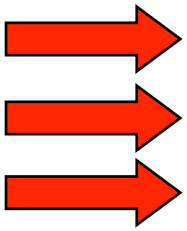
Issues?

Double Buffering

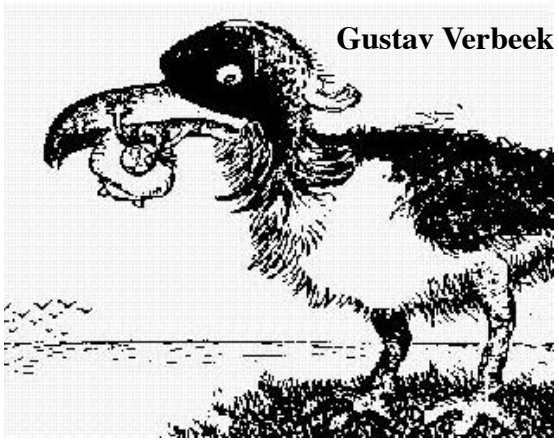
avoiding flicker…

individual drawing commands

fillRect ...



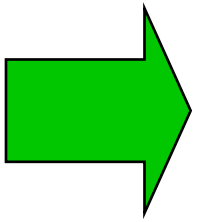
Off screen buffer



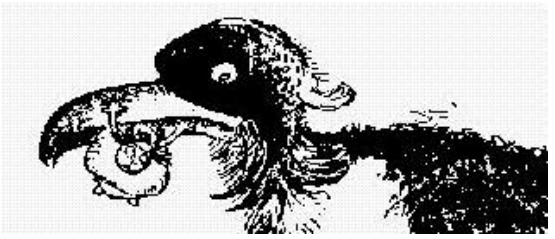
image

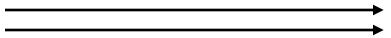
possibly slow to draw…

raster copy



Screen





screen

but fast to copy

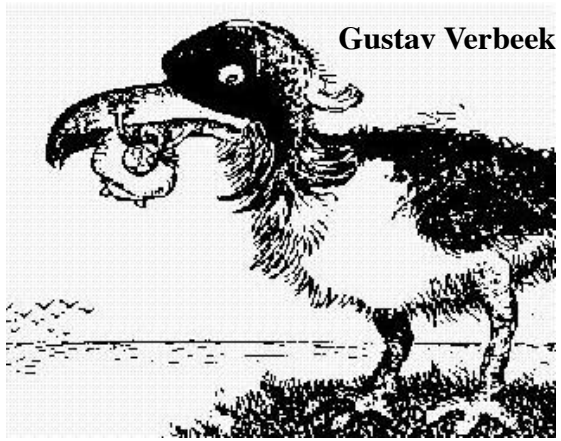
Double Buffering

avoiding flicker…

individual drawing commands

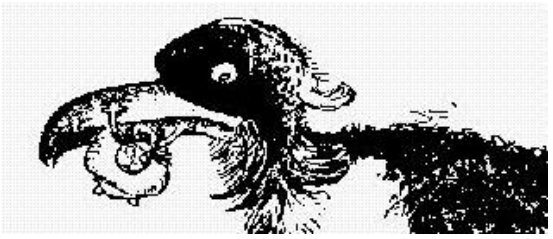
fillRect ...

Off screen buffer



raster copy

Screen



`image`

possibly slow to draw…

screen

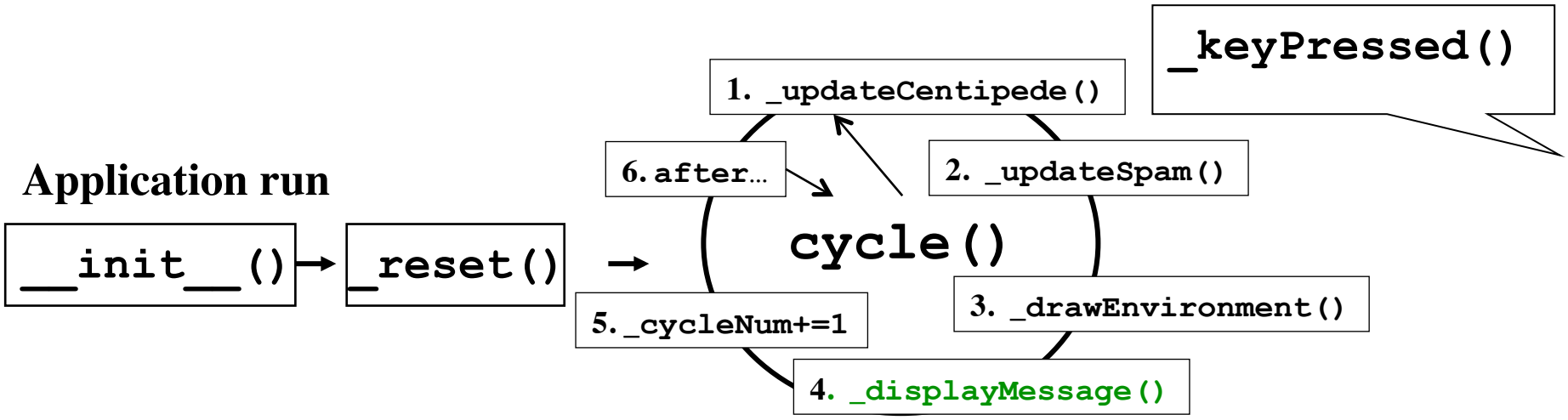
but fast to copy

```
_drawEnvironment()
_displayMessage()
```

Python automatically updates all at once

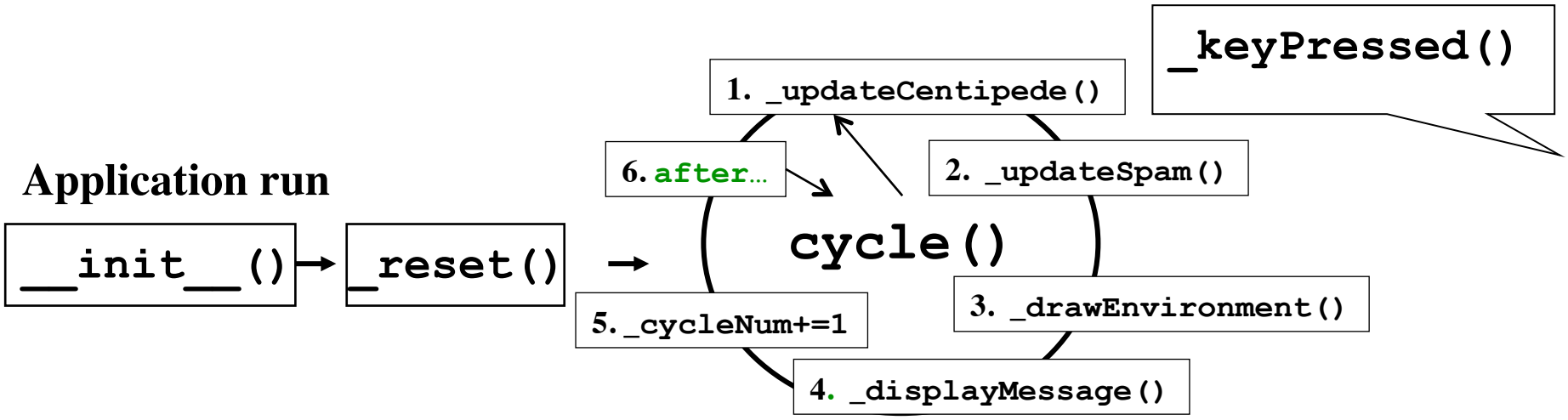
Game control details

```
def _displayMessage(self):
    ''' Display a text message to the user.'''
    self.messageLabel.config(text=self.message)
```



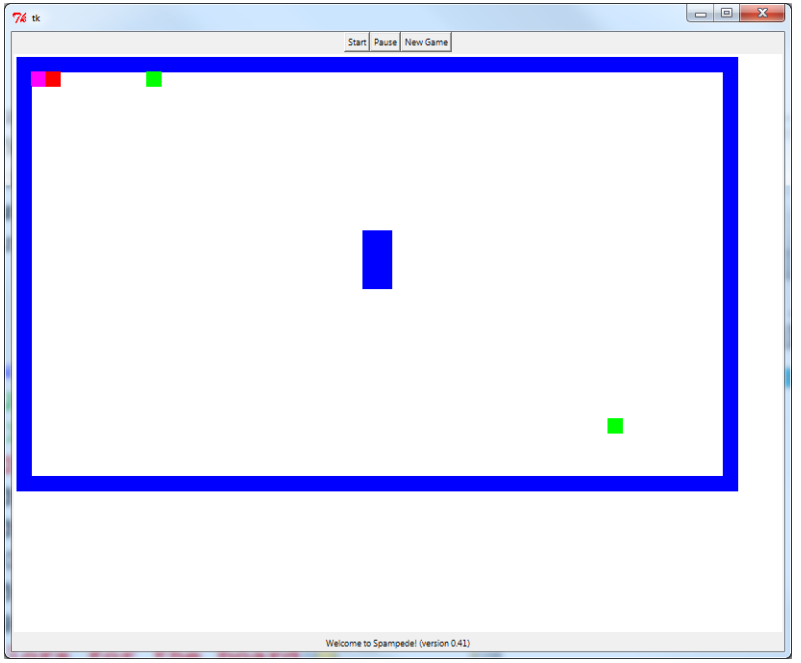
Event-driven programming

```
import Tkinter
tk = Tkinter

def main():
    ''' Start the game. '''
    root = tk.Tk()          # Create the main window

    app = Spampede(root) # Create a Spampede object


    root.mainloop()         # Give the main window control
```



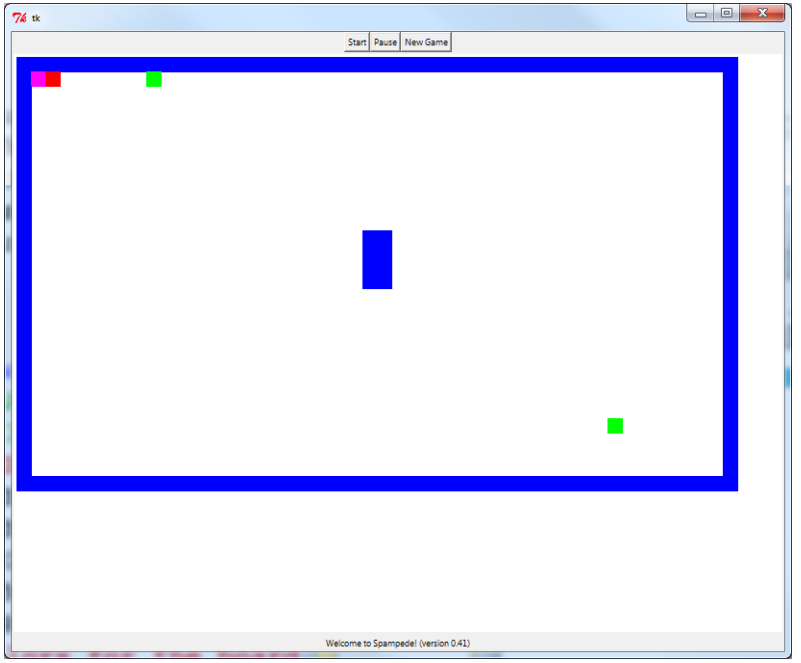
Event-driven programming

```
import Tkinter
tk = Tkinter

def main():
    ''' Start the game. '''
    root = tk.Tk()
    app = Spampede(root)
    root.mainloop()
```



Event-driven programming

```
import Tkinter
tk = Tkinter

def main():
    ''' Start the game. '''
    root = tk.Tk()
    app = Spampede(root)
    root.mainloop()
```



The program is watching a queue waiting for an "event" to show up.

Events may be processed internally, but you can also ask the system to call a specific function (a "callback" function)

Event-driven programming

```
buttonFrame.pack(side=tk.TOP)  # Put that frame at the top of
                               # the main frame

startb = tk.Button(buttonFrame, text="Start", command=self._go)
```

Passing the function in as an argument will register it to be called whenever a click event happens on this button

Event-driven programming

```
startb = tk.Button(buttonFrame, text="Start", command=self._go)
startb.pack(side=tk.LEFT)
pauseb = tk.Button(buttonFrame, text="Pause", command=self._pause)
pauseb.pack(side=tk.LEFT)
resetb = tk.Button(buttonFrame, text="New Game", command=self._reset)
resetb.pack(side=tk.LEFT)

self.canvas = tk.Canvas(frame, width=self.WIDTH, height=self.HEIGHT,
                                bg=self.BGCOLOR)
self.canvas.pack(fill=tk.BOTH)

self.message = "Welcome to Spampede v0.42!"
self.messageLabel = tk.Label(frame, text=self.message)
self.messageLabel.pack(side=tk.BOTTOM)

# Capture all keystroke events.  Handle them with the self._keyPressed
# function, which you will need to add to.
master.bind_all("<Key>", self._keyPressed)
```

Event-handlers

```
def _go(self):
    ''' This is the function that will run when the user presses the
        Start button.  (Re)Starts the game.'''
    self.running = True
    self._cycle()

def _pause(self):
    ''' This is the function that will run when the user presses the
            Pause button.  Pauses the game.'''
    self.running = False
```

Event-handlers and **`after`**

```
def _cycle(self):
    ''' This is the main "loop" that controls the gameplay. '''
    while self.running:
        self._updateCentipede()    # update the Spampede deque
        self._updateSpam()         # update the Spam deque
        self._drawEnvironment()    # draw things to buffer
        self._displayMessage()     # display messages
        self.cycleNum += 1         # add a cycle
```

Assuming all helpers were written correctly,
what would this code do?

A. Correctly play the game
B. Move the pede but not allow the user to control it
C. Move the pede and allow the user to control it, but not allow the user to stop the game
D. Cause the main window to "hang"

Event-handlers and **`after`**

```
def _cycle(self):
    ''' This is the main "loop" that controls the gameplay. '''
    if self.running:
        self._updateCentipede()    # update the Spampede deque
        self._updateSpam()         # update the Spam deque
        self._drawEnvironment()    # draw things to buffer
        self._displayMessage()     # display messages
        self.cycleNum += 1         # add a cycle
        # Wait a little while and then cycle again.
        self.canvas.after(self.sleepTime, self._cycle)
```

`after` schedules an event to occur at least `self.sleepTime` milliseconds in the future.

IDLE and TkInter

```
import Tkinter
tk = Tkinter

def main():
    ''' Start the game. '''
    root = tk.Tk()          # Create the main window

    app = Spampede(root) # Create a Spampede object


    root.mainloop()         # Give the main window control
```

IDLE is written in TkInter! So why does this cause a problem…?

Name:

"Quiz"

Code Treasure Hunt!

1) Where is the code that is drawing the small red square?

2) What do the first four arguments to `create_rectangle(int,int,int,int)` mean?

3) What is the point of the `PADDING` member variable?

4) What keypress(es) lead to drawing the large square dark blue?
   Which keypress is incorrectly reported on the screen?

5) What code draws the text of the latest keypress onto the screen?

6) What's your favorite color?

7) How long does it take for the large square to return to magenta color?

8) What happens in the code when "Pause" and "Start" are pressed?
   Where/how does **`sleepTime`** determine the speed of the application's updates?

Hw 9 ~ overall planning

1) Do the **`Maze`** and **`SpamMaze`** parts of the assignment first…

2) Make sure you can change, and test **`Spampede.py`**

3) Get a bird's-eye view of the code…

`self.theMaze`

`self.dir` the current direction the snake is heading

(doesn't exist yet)

4) Write **`_drawEnvironment`** (and test…)

5) Write **`_updatePede`** and **`_keyPressed`** (and test…) one direction at a time…

6) Build up to ***reversing*** and ***AI*** (note that AI is almost completely implemented by **`multiBFS`**)

EXTRA FEATURES!!