Wiki

wiggle

[edit]

Wiggle / "Drunkard's Walk" Examples

by Terry Hancock

This is my attempt to get something on the screen as fast as possible with PyGame, with the least amount of Python programming concepts being used, so that it will be understandable to people new to programming.

The result is not super-well-designed object-oriented programming. It's just a series of edits to a script.

When I first learned to program, it was with BASIC on a TRS-80 Color Computer, and the ability to "make pretty pictures" was a major motivating factor. For my kids, I wanted to get them started with PyGame as quickly as possible, because they already know that there are a number of their favorite games that are written using it, and they really like making sprites and animating them already.

But, they still don't understand classes, objects, functions, or even all of the basic "procedural programming" constructs like "flow control". This program is meant to teach a few of those, but first of all, to be the minimum program that does something cool enough to get their attention.

[edit]

Drunkard's Walk

The "drunkard's walk" is a very old statistics problem which basically asks if a person (the drunkard) walks in a random direction for a random time (then repeats), when will he get home?". There are lots of variations, but the essence is that it's a statistics problem, and there actually is an answer -- with various degrees of probability, you can predict that he will intersect "home" within certain time intervals. Because of this, the general case of "object moves random distance for random time, then repeats" is often called a "drunkard's walk".

Which is a little beside the point. The point is that "gee, that would be cool and simple to demonstrate". All I need is an image, then I pop it up on the screen and have it wander around aimlessly. Eventually, it walks off screen, and it's time to hit "Ctrl-C" and start over. Cheesy, but very, very simple.

[edit]

pygame.org is powered by SiteSwing dev

Step 1

You need the graphic. I used the "snake eating controller" PyGame banner in my original. I got mine from the Debian python-pygame package, you can get one from this page. You'll need to use ImageMagick or some similar program to make it into a .bmp file, and name it "pygame_tiny.bmp". (Alternatively, you could use any picture you want, and change the scripts to point to that picture).

Step 2: wiggle0.py

Just about the simplest possible Drunkard's Walk:

- Use the "random" module and the "pygame" module. Python does this with "import" statements. PyGame also provides some constants and stuff in pygame.local, which we import into the script's namespace.
- Now we just start up a screen window. Yes, of course, there are smarter ways, but I just hard-coded the dimensions in this script.
- We also open the .bmp file to create our "sprite". This is not a member of the PyGame "sprite" class, by the way. It's just that sprite is a generic name for little pictures you manipulate on the screen in games.
- Note that both the screen and the sprite are "surfaces" in PyGame terminology. You can think of a "surface" as being similar to a "cel" in conventional cel animation -- a transparent layer with (potentially) a picture on it, that can be moved relative to other layers, or modified seperately.
- We figure out the middle of the screen and the middle of the sprite, then find the point where the upper left corner of the sprite should be placed to put the sprite in the middle of the screen. We call that point (x0, y0). We also start out the sprite's location (x,y) at that point.
- We define the directions as ordered pairs -- (change in x, change in y), and give these the four compass point names: N, S, E, W.
- We then start an infinite loop ("while 1:").
- In the loop, we pick a random direction (dx, dy) and a random distance in the interval (0,100), by using the random.choice() function.
- If we just change (x,y) by this much, the image will "hop" rather than "travel", so, instead, we loop and adjust its position by one-pixel each time.
- We repaint the screen by using the screen.fill method. This is sort of wasteful -- there are more efficient ways of doing this. But they would complicate the code, so I don't attempt to use those.
- We paint the sprite in the right place with screen.blit.
- Then we update the display and loop. This goes on until something interrupts the program.

The result is that the image slowly wanders around the screen, randomly.

Step 3: wiggle1.py

The first problem that shows up is that the image eventually wanders off of the screen. That's no fun. So, we use if-then statements to catch when this happens and "reflect" the image's motion back into the screen. So, it appears to "bounce" off of the edges of the screen instead of just wandering off.

Also, just for the fun of it, we define and use the diagonal directions, NE, SE, SW, NW.

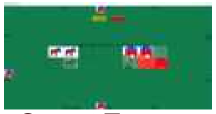
Step 4: wiggle2.py

To make things a little more interesting, we use random.choice again, and this time allow the "bounce" to be more random, instead of following simple reflection rules.

Step 5: wiggle3.py

Write text in pygame! 73910+"*&/

Pygame Text Input - 0.1

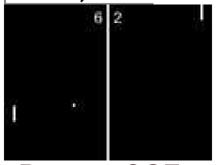


Secret Trump - v0.2.0b

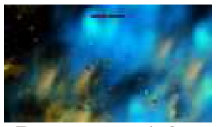
Nov 6, 2016

koxinga - v0.2.7b

Nov 5, 2016



Pygame SGE - 1.4.1.1



Pacewar - 1.6.5

Oct 22, 2016

Nashes to Ashes - 0.1.1

Oct 11, 2016

One Bullet Frog - 1.0

no screenshot available

mygame - 1

... more!

Okay, random is fun, but I want to *play* with the program. So we add keyboard control. This attaches the numeric keypad buttons to each of our 8 compass directions, and the sprite now keeps moving in whatever direction we've sent it, instead of randomly picking directions.

This introduces PyGame's even queue, and how you pull events off of it to act on them.

We decided to make the '5' key do a "random teleport".

[edit]

Voila, you've used Python and PyGame

There is, of course, a lot more to programming and to PyGame than this. However, now you should have some idea about how things work with PyGame, and hopefully you realize just how easy it is to use.

[edit]