I Want Cake! An introduction to Pygame

Have you ever wanted to make your own video game? This guide will walk you through a simple game that was created using Python and the Pygame library. We will give you some helpful suggestions and inspiration on adapting the code in this game, so that you gain an understanding of how to use Pygame. Design your own game using what you learn in in this guide!

Getting Started

1. Install Python 2.7.11 for Windows (x86 installer/MSI file):
   Download the correct version from https://www.python.org/downloads/windows/
   Run the .msi file and accept all default options
2. Install Pygame 1.9.1 for Windows (win32 installer/MSI file):
   Follow same directions as above, downloading Pygame from:
   http://www.pygame.org/download.shtml
3. Install Notepad++ as a text editor:
   https://notepad-plus-plus.org/download/v6.8.8.html
4. It might be helpful to spend a few hours working through Python tutorials on CodeAcademy. The most relevant sections will be Units 1, 3, 4, 8, and 11:
   https://www.codecademy.com/learn/python

The I Want Cake! Game

Included with this guide is all of the code required for a game called I Want Cake! The game is pretty simple, and pretty dumb. It has three sprites: a Skunk, a Banana, and a Cake. When the player presses the left or right arrow keys, the Skunk will move around accordingly. The goal is for the Skunk to eat the Cake. When the Skunk collides with the Banana, it slips and spins out, which at this point is unavoidable because the Skunk can only move left and right and the Banana is directly in the path that the Skunk needs to walk to get to the Cake. There aren't points, and there isn't a timer, and there is only one level to the game, so obviously it needs some help from you in order to really be worth playing. That's what makes it such a great starting point!

In order to play the game, open up a command prompt in the same directory as all of the Python files included with this guide. You can do this by searching for the `cmd` program, copying the directory path from File Explorer, and running this command from `cmd` (hit enter after typing this):

```
cd <path_copied>
```

Once you are in the correct directory, launch the game by typing the name of the primary game file and hitting enter, which will be this simple command:

```
cake_game.py
```

That's it! You should see a window open and display the game area. Below we'll explain how to start figuring out how this code works.

Navigating the Game Files

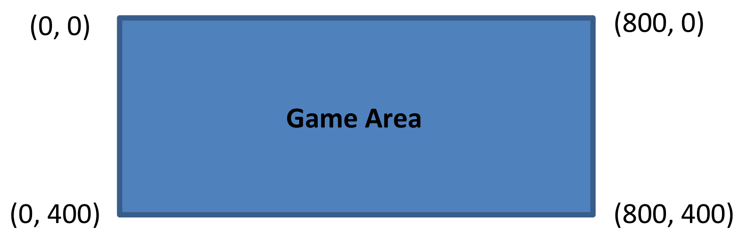
Look at the contents of the game directory in File Explorer. You should see three Python files and a few extra files with type .pyc (you can ignore those for now) in the **`src`** directory, and image files used for sprites in the **`images`** directory. Open up a few of the Python files in Notepad++ and start reading the code and comments. Comments are inserted by the programmer to let other humans know how to read their code, and are indicated by lines that start with a **`#`** or blocks of text surrounded by triple quotes (“””).

- **`cake_game.py`**
  This file is where all of the magic happens. Ignore the other files until you have a pretty good understanding of what is going on in this file, which contains the **`main`** method. The **`main`** method is the piece of code that actually runs when you start up a program – everything else is just there to be used by **`main`** and placed in other files to keep the code organized.
  The main method in a game program consists of two different phases:
  1. One-time setup (before the **`while 1:`**– Just like it sounds, this stuff needs to be done only one time to create the scenery, sprites, etc. that will occupy your game.
  2. The game loop (everything within the **`while 1:`** block) – This loop will execute forever as long as your game is active. It will check for user input to the game (key presses, mouse clicks, etc.) and respond to them appropriately by calling sprite action methods, then redraw the screen.
- **`utility.py`**
  This file contains some helpful methods for loading image and sound files. The image loader is already being used in sprites.py, and you can make use of the sound loader in either cake_game.py or sprites.py. These utility methods came from a Pygame example, and you find out more about them here (and get some other game ideas!):
  https://www.pygame.org/docs/tut/chimp/ChimpLineByLine.html
- **`sprites.py`**
  This file contains a class definition for each type of sprite used in the game. It would be helpful to do a little reading about Object-Oriented Python before examining this code. Each sprite class needs to define two very important methods:
  1. The constructor – This method starts out with the code **`def __init__(self, xcoord, ycoord)`** and defines what happens when you create a sprite object to place in your game. A class definition makes it easy to create as many instances of a sprite as you want, and the constructor is like the blueprint for how each one should be created. A good constructor method will at least load an image for the sprite and assign the sprite to a location on the game screen. You may choose to give a sprite other properties such as a speed, point value, etc. that you would want to set up in the constructor as well. Compare the constructor used for the Skunk with those used for Cake and Banana to get an idea of what the constructor can be used for.
  2. The update – This method will start with **`def update(self):`** and is used by Pygame to automatically perform an action on your sprite every time the screen updates. You should add code to the **`update`** method any time you want a sprite to do something

continuously or in a fixed pattern. You could use **`update`** to make a sprite spin, walk back and forth, disappear and reappear, etc.

Make the Game Better!

1. **Make it your own.** The first rule of this exercise is to use your imagination when updating the game's features. It certainly doesn't have to be a game about a skunk that wants to eat cake, and it doesn't have to follow any rules. Use the suggestions below to get started editing the game as you see fit.
2. **Load an image as the game background.** This is just as easy as filling the background with a solid color! Look closely in **`main`** (hint: around lines 30-35) for some code that we've left commented out that would load an image instead of the white background. Notice that you have to `blit` (Pygame for draw) the rectangle floor on top of the background and **`blit`** both to the screen every time the game area redraws in **`while 1`**.
3. **Change the sprites.** Find image files on the internet and use them as your game sprites. Clipart-style images will work the best, and be aware that you will probably need to shrink the files in order for them to fit the game area (see **Tips** section below). Simply save your image file in the same directory as the other game files and update the filename of the image being loaded in the cake.py, banana.py, and skunk.py constructors.
4. **Draw other shapes in the scenery.**
Pygame provides utility methods for drawing various basic shapes on the background. For an example, see how this game draws a rectangle to serve as the game floor in lines 31-34. More examples of Pygame drawing utilities can be found here:
https://www.pygame.org/docs/ref/draw.html
5. **Change the speed that the Skunk (or whatever your sprite is now) moves at.**
The Skunk class uses the variable **`self.move`** to indicate the number of pixels that the Skunk should move every time the left or right arrow is pressed. Changing the value of this variable in the constructor will change the pixel shift, which ends up making your sprite move at a different rate. Try experimenting with the value of this variable.
6. **Change the dimensions of the game area.** Keep in mind that you will likely have to move the sprites and other objects drawn onto the screen to match the new dimensions. The coordinate system within the game area designates the upper left corner as (0, 0), so think of object placement as relative to that corner. See coordinates below!



7. **Make the Skunk jump!** Wouldn't it be nice if you could avoid that darn banana? Look at how the game triggers the Skunk's walk methods from the event handler loop, and add a jump method

to the Skunk class. You could let the Skunk magically float upward with every key press, which wouldn't be out of place if this were actually a top-down game or your sprite were a magic carpet, but if you like the side scrolling view then you may want to figure out how to animate a smooth jump/fall sequence. Pay attention to how the Skunk class animates the spin-out that is triggered when you hit a banana, using the **`dizzy`** variable and the **`update`** method.

8. **Add more sprites to the game area.** You could introduce a second banana, or put birds in the sky. The Cake and Banana objects are very simple sprites because they don't move or do anything. A quick way to create a more interesting sprite would be to add code to the **`update`** method of either Cake or Banana to make it move. Use the Skunk's walk methods to get an idea of how to move objects.
9. **Add an event handler for the spacebar key (or another key).** It could be whatever you want – the skunk could shoot some sort of projectile, or another object could start moving. Just add another case within the main event handler loop and insert code to call an action method of one of your sprites.
10. **Add a score counter.** Define a new variable in the game, and add 1 to it every time a certain game action happens. The action could be whatever you want – the Skunk sprite colliding with another object, every time the spacebar is pressed, etc. Look at line 99 in **`main`** for an example of how to create text to display on the screen. Also keep in mind that you may not want to draw the text directly onto your background image/rectangle, since you expect the score to change. Try **`blit`**ting the text onto the screen directly between lines 114 and 115?
11. **Add sound to your game.** We've provided you with a handy Pygame utility function for loading sound files in utility.py. Give it a try, and check out this example on how to use it: https://www.pygame.org/docs/tut/chimp/ChimpLineByLine.html
12. **Use an array and loop to setup multiple sprites of the same kind simultaneously.** What if you want to create many different instances of Banana and place them all around the screen? You'll have to call the Banana constructor with new coordinates for each object you want to place, and it would be really slick to do this with a loop and array. Make sure to update the collision-checking code (if your game still uses it) to check all Bananas!

Tips

- Google is the programmer's best friend! If you have an idea about something that you want to do with Pygame, or don't know what a certain piece of code means, search for it online and you will usually be able to find quick and detailed answers.
- Make sure that your sprites aren't too big to appear on the screen! Image files will give their dimensions in pixels, and if these are larger than your game screen then the object likely won't appear at all. You can quickly resize the sprite by opening the image file in paint and using "resize" to set the pixel dimensions to something that makes sense for your game.
- If you want to use other keyboard keys as game inputs, you can find the key values here: https://www.pygame.org/docs/ref/key.html
- After you're comfortable with what you've learned here, find more information online! Look for videos on YouTube of games that people have made with Pygame, and take a look at the official

Pygame website StackOverflow for more resources. This page (below) has a few interesting tips that could be used to improve your game as well:
http://www.pygame.org/docs/tut/newbieguide.html