Raffy Santayana

Net-Centric Computing

Sushi Bot: 18 September 2017

Documentation

Date: 14 September 2017

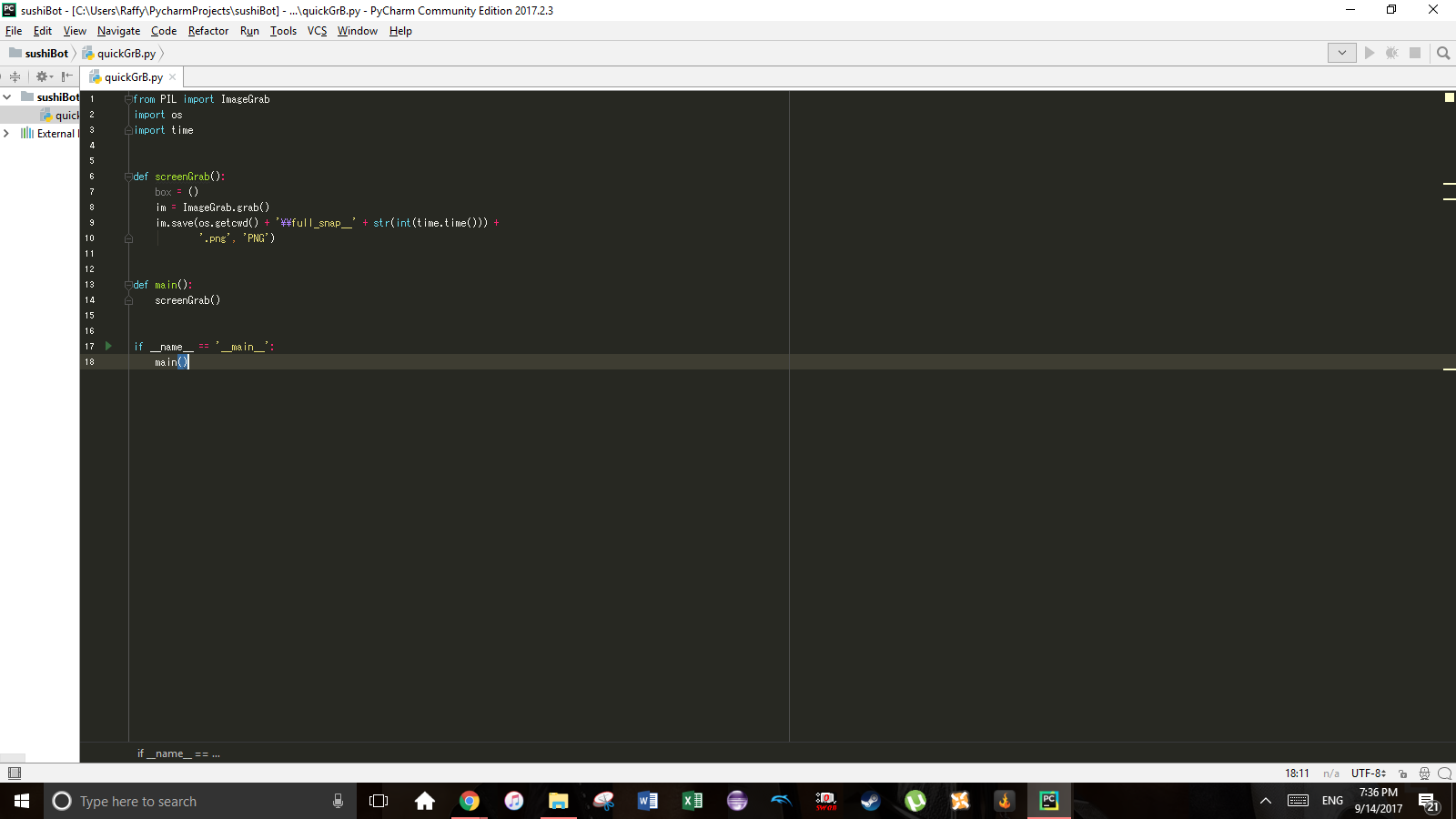
I have currently powered through the first 3 steps of the bot instructions, so allow me to update you on my progress.

**Step 1: Create a New Python Project**

Rather than creating a folder and making a python file within it as suggested by the instructions, I have made a new project within Pycharm titled sushiBot. I then right-click the project folder within the directory and went to New > Python File and accidentally named it quickGrB.py.

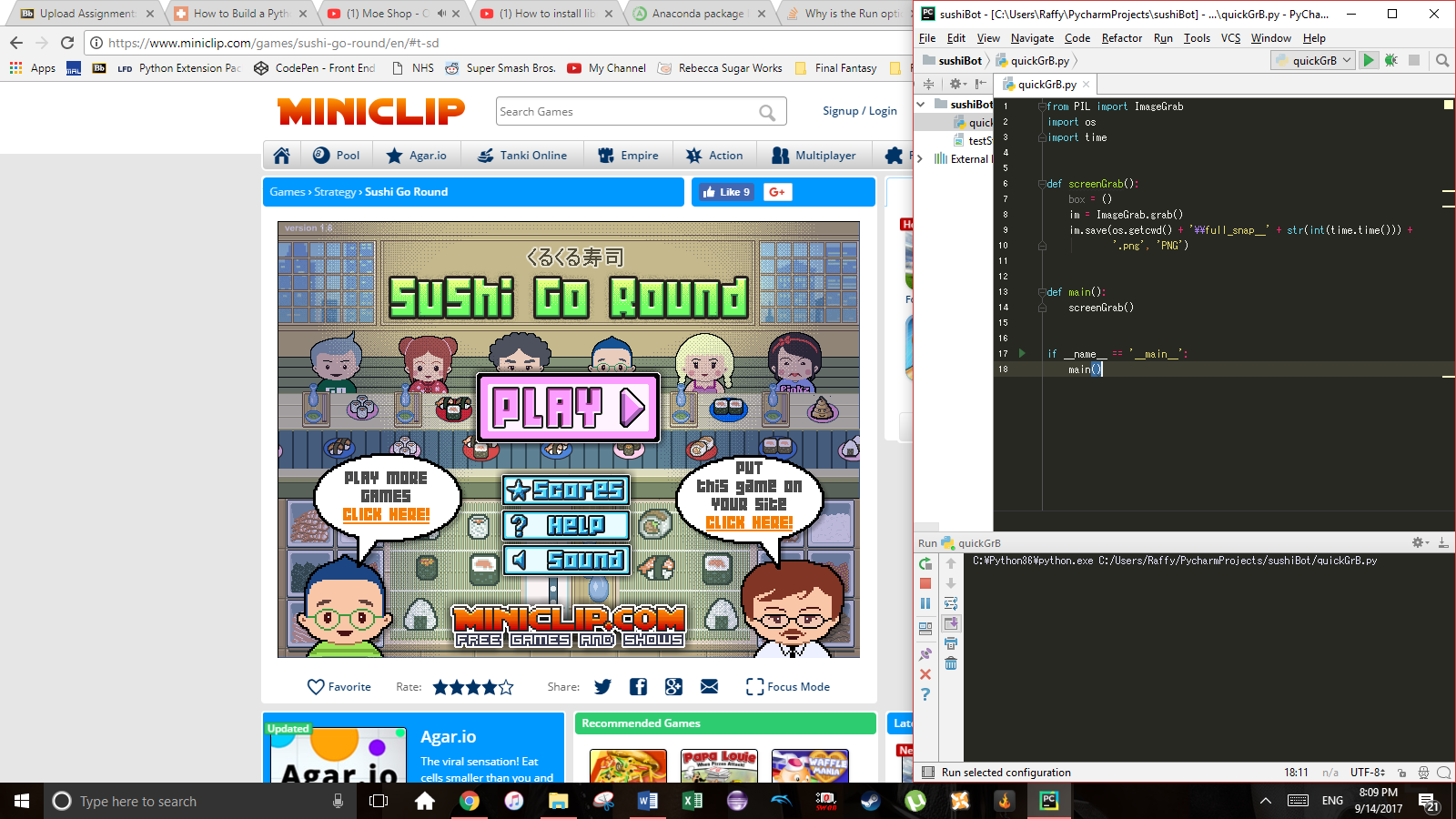
**Step 2: Setting Up Your First Screen Grab**

Onward to more important notes, I have copy&pasted the 18 lines of code provided by the instructions and only made an edit to the first line.

Before the edit, line 1 was “import ImageGrab”, but then an error appeared and stated that there was “No module named ImageGrab”. While collaborating with a student from another Net-Centric class, Sam Steiner, we have discovered that we can import ImageGrab from PIL. So line 1 became “from PIL import ImageGrab”. We then tested the code and found the following image in our project folder. I renamed this image to be testStepOne.png for future reference.

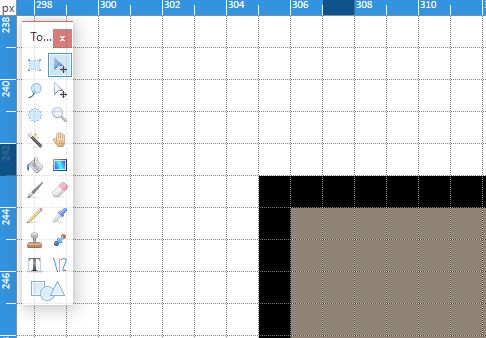
**Step 3: The Bounding Box**

This step states that we must define the box that contains the game Sushi Go Round. It is defined within the code as box(x,y,x,y). The first pair of xy-coordinates are the top left corner of the game and the second pair is the bottom right corner. In order to do so, we must take a screenshot of the game. I have titled the following image gettingCoordinates.png.

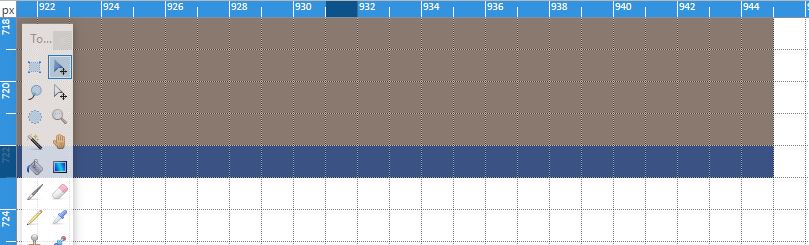


**Step 4: Getting Coordinates**

Using paint.net, I was able to find out that the top left corner of the game is found at (305, 243) on a screen that has a resolution of 1600 x 900.



I also found the bottom right corner.



According to step 3, the box should be defined as (305, 243, 944, 722). However, I was confused when the instructions have coordinates of (157, 162, 796, 641), but then define then they define box as (156, 346, 796, 825). So I test the coordinates I found and found the following image (testStep4.png).



Date: September 15, 2017

**Step 5: Planning Ahead for Flexibility**

With the given coordinates I have found to create the frame of Sushi Go Round, the provided code will work exclusively if the resolution is 1600 x 900. In an effort to make the bot perform on systems of various resolutions, we have created a global variable for the xy-coordinates of the top left corner. By doing so, the user will edit the global variables to adjust to the proper resolution that they are using.

I have tested this edit and received the same image as about, but named this one testStep5.png.

**Step 6: Creating a Docstring**

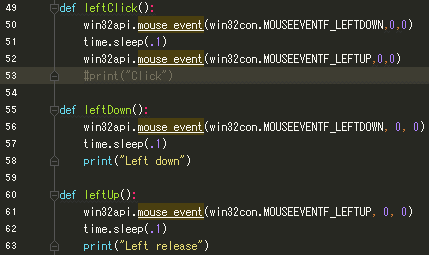
Nothing crazy has been done. I have included the following comment to help readers understand my code:

"""  
All coordinates assume a screen resolution of 1600x900 and Chrome window is maximized with  
bookmarks enabled.  
The page is scrolled to the very top.  
x\_pad = 304  
y\_pad = 242  
Play area = x\_pad+1, y\_pad+1, 944, 722  
"""

**Step 7: Turning quickGrab.py** **Into a Useful Tool**

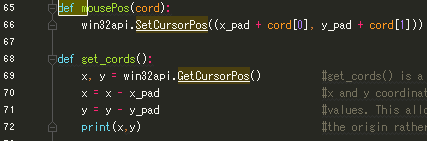
As instructed, I created a copy of my quickGrB.py file and rename it to code.py. I finally change the file type of quickgrB.py to pyw.

**Step 5(?): Basic Mouse Clicking**



The above code was added and is described in code.py.

**Step 9: Basic Mouse Movement**



The above code was added and is described in code.py.

**Step 10: Navigating Game Menus**

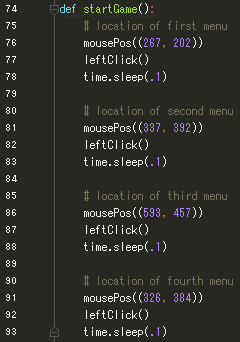
This step recommended using the python shell and running the following:

get\_cords()

By running this, the shell would return the cursor’s current pixel position with respect to x\_pad and y\_pad. Instead of using the shell, because I could not get it working for some reason, I put get\_cords() in the main() and ran it using the command ctrl + f5. Of course, my cursor would need to be in the position that I would like to record.

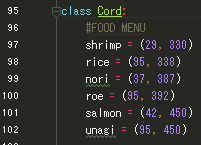
Starting the game from the main menu, I would get the coordinates of Play > Continue > Skip > Continue.

Using those coordinates, I have implemented this block of code.



Running startGame() would get the cursor to click on those positions and start the game.

**Step 11: Getting Food Coordinates**



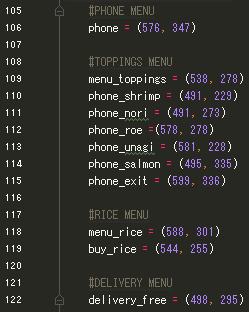
Added the above class. These are coordinates of the icons for each ingredient. This is so that when I use the mousePos() method, I can use Cord.rice. By doing so, the cursor will move over to coordinates (95, 338).

**Step 12: Getting Empty Plate Coordinates**

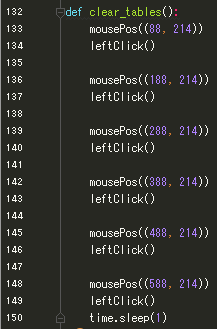
Still using getCords(), I find the coordinates of the plate positions so that the cursor can click those positions and clear them. I used these coordinates in a few steps.

**Step 13: Getting Phone Coordinates**

I used getCords() to get the coordinates of the following:

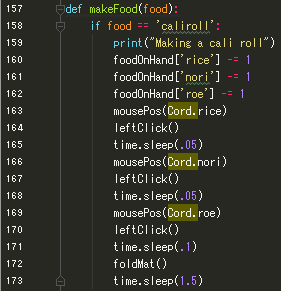


**Step 14: Clearing Tables**



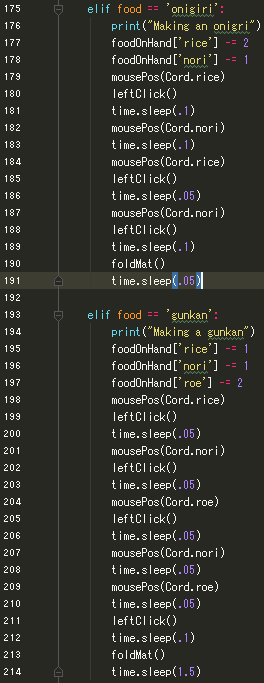
Using the coordinates I received from Step 12: Getting Empty Plate Coordinates, I create this function that clicks at each plate position. If there is a plate, then it will be cleared. If there is no plate, then nothing happens.

**Step 15: Making Sushi**

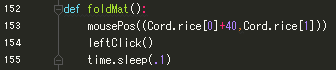


Above is a segment of code that makes california rolls.

When making gunkan or rice balls, there would be an occasional error and poop would be produced instead of the demanded sushi. This would happen because gunkan and onigiri require clicking the same coordinate twice. For some reason, the bot would occasionally drop the second click on the ingredient that is needed twice in the recipe. In onigiri, the ingredient is rice. In gunkan, the ingredient is roe. I toyed around with the code that was supplied from tutsplus.com and found the following code to be a solution.



After clicking on rice or roe for the first time when making onigiri or gunkan, I would move the cursor to an arbitrary position, then back to the ingredient and click.

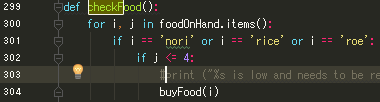


To finish this step off, I added the above code to fold the mat when called.

**Step 18: Making the Computer See**

Skipping over to Step 18, the instructions recommend that I edit my screenGrab() method so that I can define the screenshot as **im**. From there, they ask that I use the python shell, run screenGrab(), and then use the getpixel(xy) method to grab the RGB values of the coordinate that was called. I then input this information into the buyFood() method.

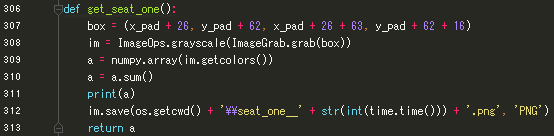
**Step 21: Checking Food on Hand**



This function will check which ingredient in our inventory is less than or equal to 3. If the inventory count is any number below 4, then it will call the buyFood() method and buy the ingredient that is low.

**Step 23: Setting New Bounding Boxes**

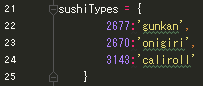
I took a screenshot of the game area when every seat had a customer in it and used paint.net to get the coordinates of a specific pixel. I was searching for a pixel that in the bubble that displays the customer’s order and from there, create a box. I continue this for every seat and create:



The functions of the other seats are identical aside from the coordinates. What happens is that a screenshot of the box is taken in grayscale. Then each pixel is placed into an array and the color value is summed up.

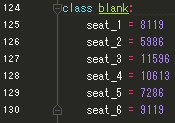
**Step 24: Create a Sushi Types Dictionary**

Using the sum of values calculated above, I was able to create the following dictionary:



**Step 25: Create a No Bubble Class**

I perform steps 23 and 24 again, except for the case of no customers in the store. I then get the grayscale values of the seats when there is no order placed. I then create another class as:



**Step 26: Putting It All Together**

I will not be including a screenshot of the check\_bubs() function because the entire function is too long to fit in my screen. In short, the code

1. Checks inventory count
2. Checks to see if the seats are empty
3. If seat is not empty
   1. Make food that is ordered
4. Clear any empty plates

This is performed for every seat.

**Step 27: Main Loop**

