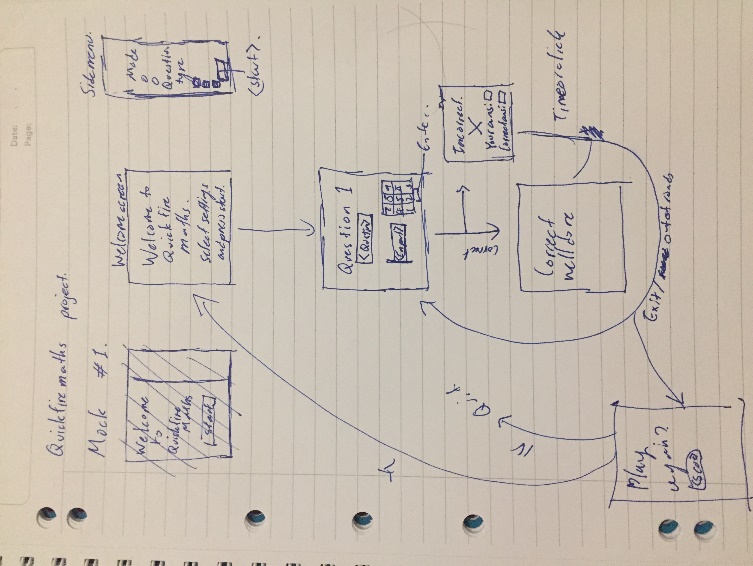
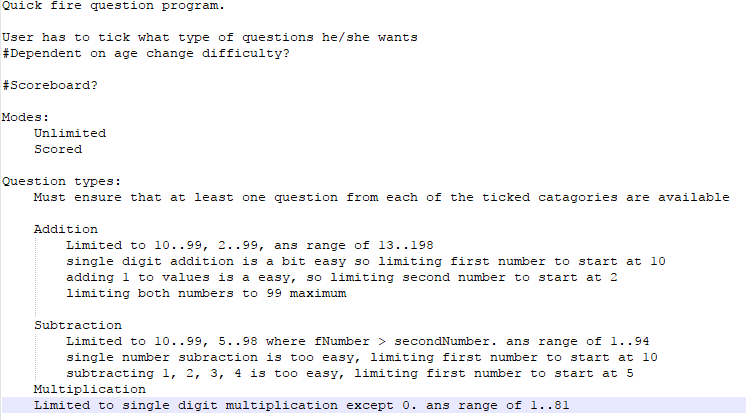
## July 31, 2017

Drew up a mock sketch of the program.

Thinking of a program which sends out randomized simple questions and the user must answer them.

Will be different modes, user being able to choose between round mode, which have a limit on the number of questions, a limit possibly set by the user, and unlimited mode, which has no limits and goes on forever until the user quits.

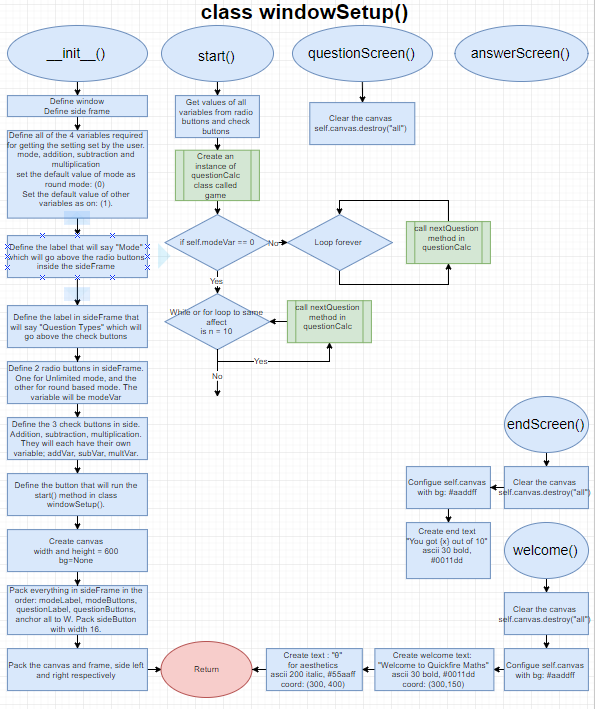
Also will have different types of questions that the user can choose to enable.

Notes below on question types:

The settings are going to be controlled by a side menu which will be always visible. The main gui is going to reside in a canvas inside the canvas

Starting flowchart

## Aug 1, 2017

Finished planning the first 2 steps of the program. But having trouble seeing how to move on. Going to write up a test program to mimic the current flowchart, check the plan’s validity, etc.

## Aug 2, 2017

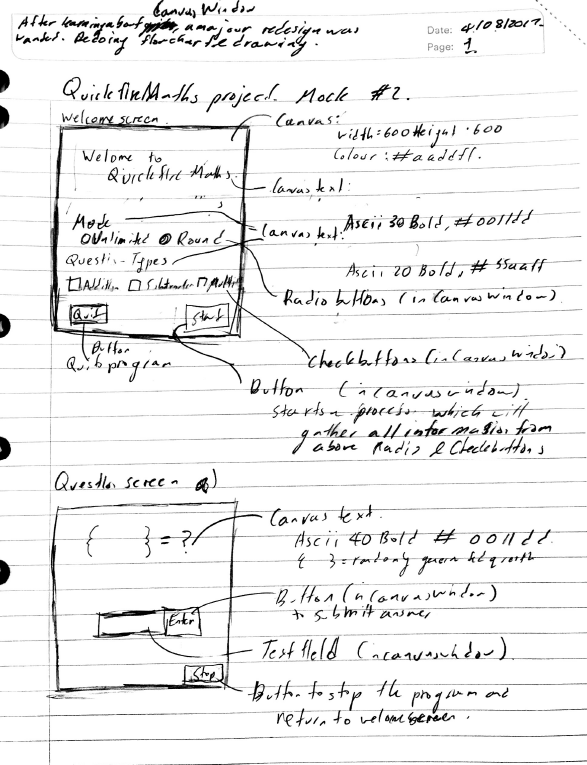
Program of test file of the first 2 steps completed to mimic the flowchart. No major issues to note. Changing certain elements of the flowchart such as names of variables or values etc.

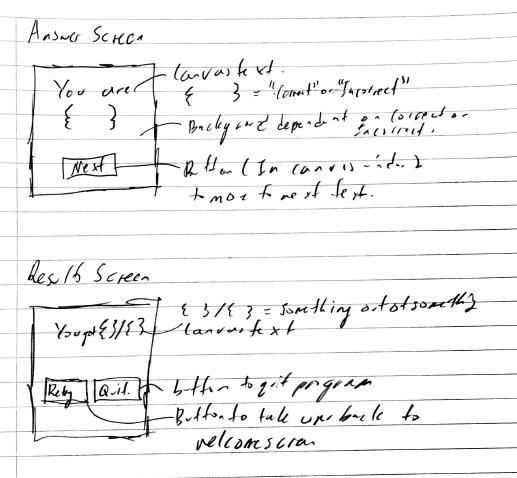


## Aug 8, 2017

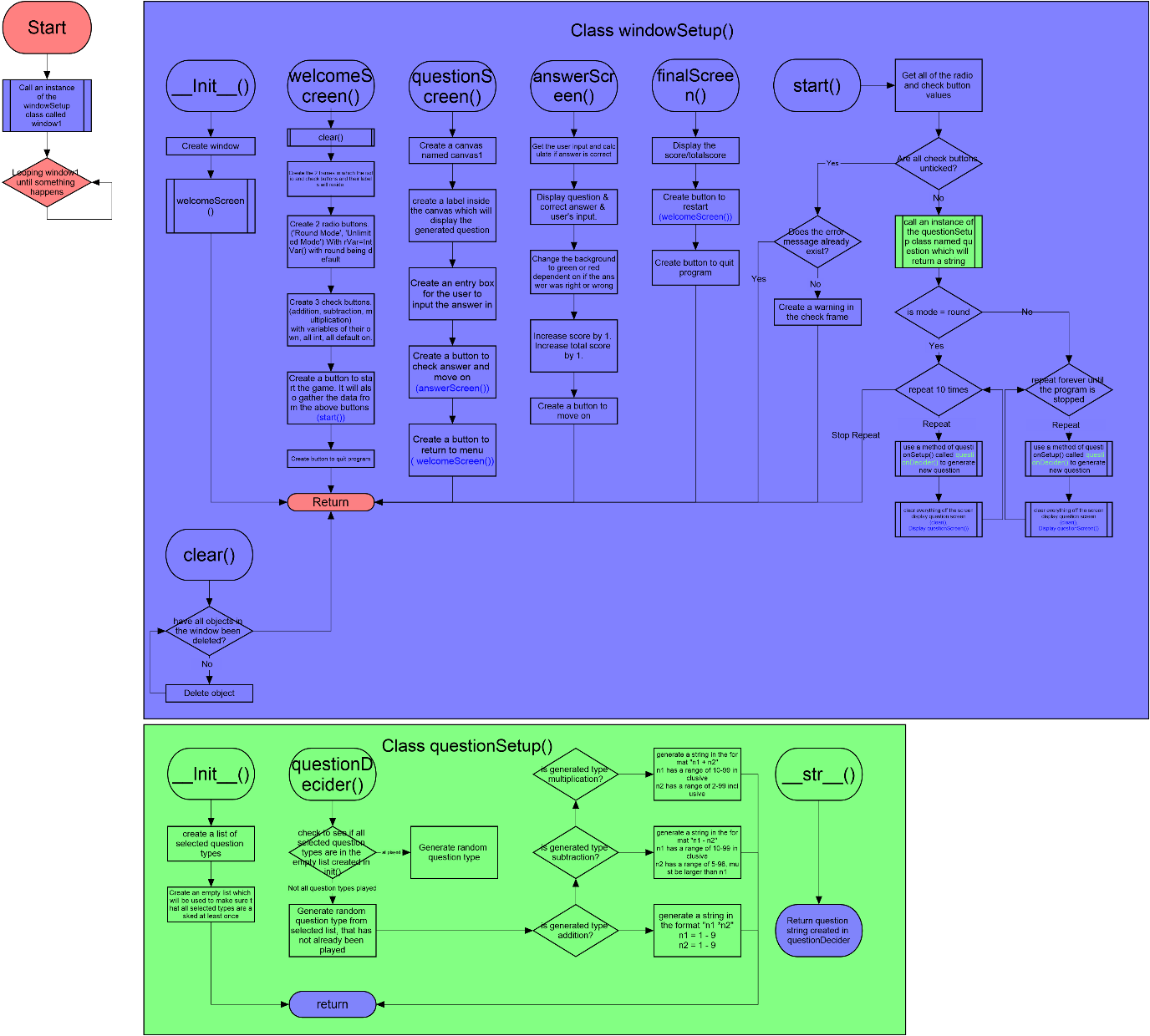
After learning about the ability create windows inside the canvas, decided to rethink the program GUI layout

Ditching the side menu idea. Going for a main front page in which you choose the settings you wish

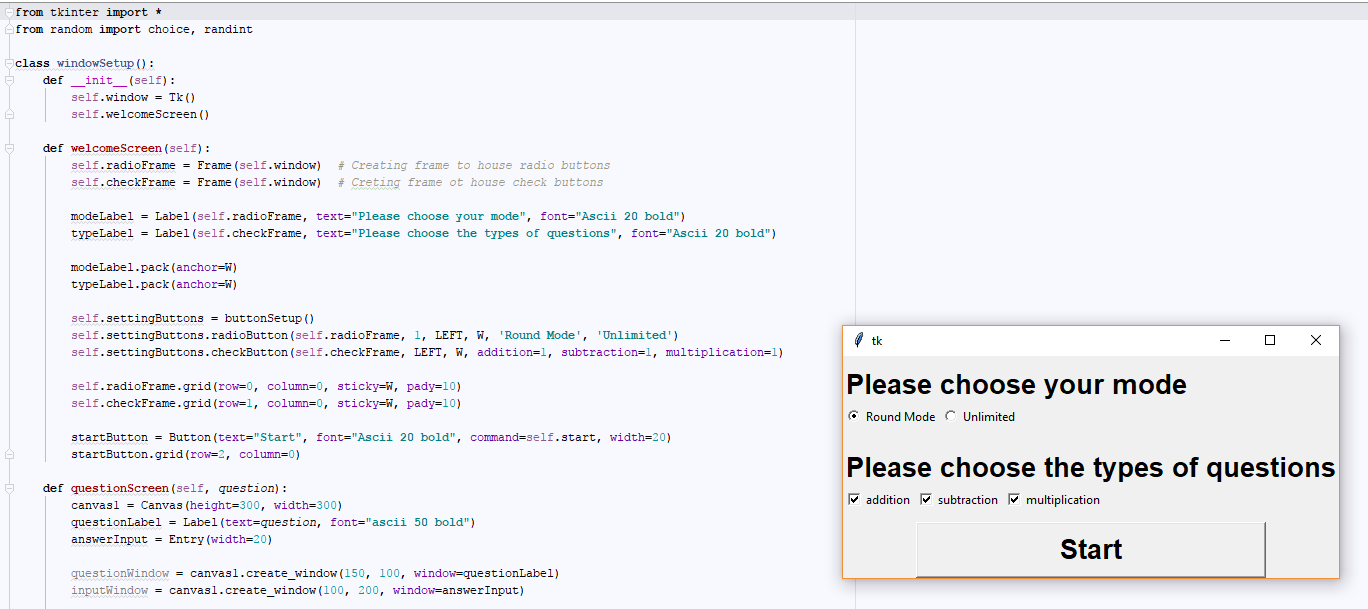
Restarting flowchart as this is a quite a significant change.



## Aug 11, 2017

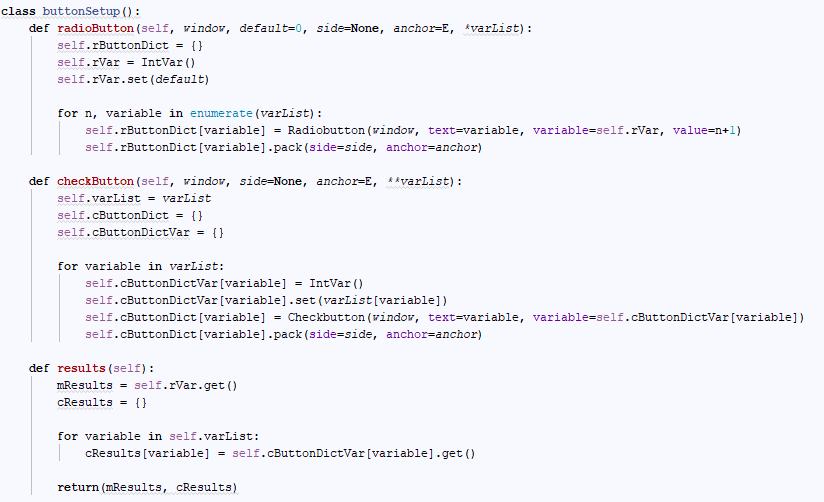
Finished flowchart, and the beginnings of a variable and testing table. I assume as time continues, it is simply going to get longer. Got it approved. beginning coding. The test.py document I’ve been using while I’ve been planning to test the validity of the flowchart is no longer necessary.

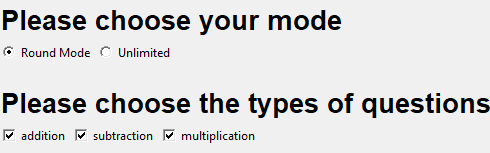
## Aug 13, 2017



Finished coding a few lines, half way through my plan, but I can already see this is not going to work. Too messy, with variables and widgets being defined everywhere and every time. Is a real chore to go through. Thinking of diverting from plan and restarting to a more efficient approach, loading everything first in \_\_init\_\_() in the windowSetup() Class, and simply using the methods to either grid, pack, or canvas.create\_window() them. That way, even if I clear the screen, I’m technically never destroying any widgets and can display them again any time I want. This I feel is significantly more efficient than creating things everywhere and deleting them, and then when I need something again having to make it again, and coming up with errors because a thing was destroyed that I needed.

Regardless, I’ve also already diverted from the flowchart to toy around with the idea of a button create class. This will be a class I call if I want to create a radio button or check button.





I mean it works. Very well actually, I’m quite proud of it. Normally this would involve like a line every radio button I wanted to make, and additional lines for defining the variable it updates, and for check buttons it’s even worse, requiring a variable for every button, and a line to actually create the button. And if I want to set default values, that adds one lines per each check button. This class simply compacts all of that down to one line. Getting the results is relatively easy as well, and integrating new modes and question types will be very easy, simply add more parameters when calling any of the methods. The only problem is its quite large. Its 29 lines, and by my quick (perhaps incorrect) count it would only take around 17 lines of code to add those buttons normally, and that’s including packing. So I’m not sure if this is worth it. It’s not really saving an awful lot of work either. The radio buttons are really easy to add more of. As there is already a variable they report to, it would literally take one more line (to define it at least) to add it and another line to grid/pack it. And check buttons, it would take an additional 3-4 lines at most to add. Also, the class isn’t really flexible if I want to change between gridding and packing, and although I could add another parameter which accepts that, that’s adding more lines of code. It would only really become efficient in terms of the lines if I was adding many more buttons, but considering the size of the window I’m planning of around 600 px, if you add 2 more question buttons, which would get the code without using the class to around 25 lines, it would look horrible, and at that point you are expanding the program so much you should just design a new GUI display. So I’m not sure this class will stay.

## Aug 25, 2017

Since the last log, 2 major things have happened. Like I was thinking in the last log, I decided to restart in favour of a more efficient “load everything first” option. The program now loads in everything that every window needs before it is displayed. The 2 thing that has happened is that I’ve finished the program! I’ve made significant from the flowchart, basically, I’ve completely segregated the classes based on their functions. The windowSetup() class is purely for managing the Gui. Version 2 of the program had the windowSetup() class have if statements and such, but all of that backend stuff has now been pushed into the processes() class. This did however mean that I had to learn how to do inheritance, as I had to use methods and objects from the windowSetup() class from the processes class and vice versa, and although there are ways around it, at some point it just becomes easier for the brain and the efficiency of the code to just use inheritance. And I think I got the gist of it. It works. I have no complaints. Apart from when it doesn’t work. But that hasn’t happened yet. So I’m happy. The tipping point towards inheritance came in the final stages, when I was trying to glue all the individual bits of the code together. I had coded every single other bit of the code; the input check, every window, ans check etc. All I needed to do was tie the code together with one method. The problem was that that method, needed methods and objects from both windowSetup() and processes() (for example, displaying a button from windowSetup() would lead to calling a function in processes() via the gameRunner()). And at that point in development, i had ran the code by simply calling an instance of the welcomeScreen() method of the windowSetup() Class. The windowSetup() Class would then create an instance of processes() with parameters so that processes could still call objects and methods in windowSetup(). This was a road block to the segregation of the classes, gameRunner had to be in windowSetup() as to call processes() with the correct parameters, parameters such as the canvas and the actual root window. So inheritance was the only (observed) option in separating the two classes. Segregating the two classes meant that the code could be kept really clean. If there is something wrong with the formatting, I know for a fact where that is. If some background code isn’t working, I know where I need to look. It makes debugging the code much easier, and expanding the code much easier too as you don’t have to comb through lines of redundant (to current work) code to try and find the one that matters.

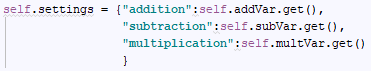
I’ve also tried to design the code so that the only parameters that a method requires would be those in the other class. I’m not sure this is effective in terms of efficiency, but it seemed logical to me to do this. Because I used inheritance, I can access any method and object from both objects when ever I want, but this will just be confusing when I’m referencing some variable that’s in another class, and trying to find that varible during debug. So if a variable needed is inside the same class as it is being defined, it is for most cases not a parameter needed for the method it is being needed in. If it is in another class, it is a parameter of the method.

\_\_init\_\_() is a bit messy due to the fact that everything is defined there. I’ve grouped the widgets there by the page they are displayed in apart from error texts, which have their own sections. This made finding the thing I want easier.

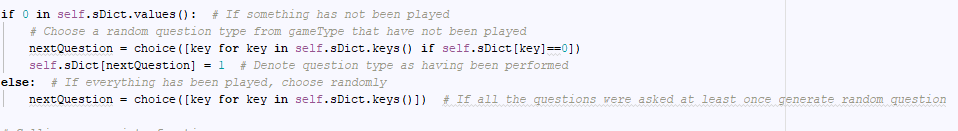
There was also a very interesting workaround I had to find as well. This is how I generate my questions for multiple rounds:



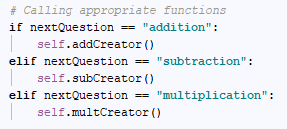
The windowSequence() method then generates a question, and displays the question screen and after that, the answer screen. The problem is because question screen and answer screen simply create\_window a few things and do nothing in terms of processes, the program doesn’t actually wait for the button press on question screen before it moves on to generate an answer screen. And then, the method, although no button had been pressed by the user, has technically been finished, so it loops. To get around it, I had to introduce a variable to the buttons, and use a lambda function as a command to update that variable. I then used .wait\_variable to get the program to wait until that variable was updated. It’s a bit of a work around but I can’t think of any additional solutions which would be more efficient.

About how the program can be easily expanded upon, I have removed the button class. It was rather unnecessary. However, I have kept a few things from that code. Setting check still uses a dictionary to organise the setting inputs, so its actually rather easy to expand on the program. All that would need to be done to add a new question type is add the button, which should take no more than 4 lines, (define variable, create the button, set the default value to on, grid them appropriately), add the name of the question type and the new variable to this dictionary in checkSetting() method, and the program will automatically do settings checks and kick up errors if all of them are ticked off and wat not. It will also automatically take the new question type to an if statement via this dictionary comprehension, 

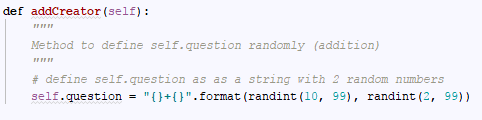
Which checks for enabled settings, including for the new question type. An if statement is then run on that dictionary of enabled settings in questionCaller() without the coder having to do anything else.



All the coder would have to do additionally is add one elif to the if statement:



And code the appropriate method in processes() which will generate a random question in terms of a string like hence:



That would be it. The code should run smoothly.

The I’m slightly disappointed about with the flexibility of this code is the fact that as the questions are string based, and as the question is displayed as a text on a label, adding geometry questions with shapes would require some significant changes. I think I would have to overhaul questionCaller() to either generate a Label or a polygon depending on the question type. It wouldn’t be hard, but not as easy as adding a “power” question type.

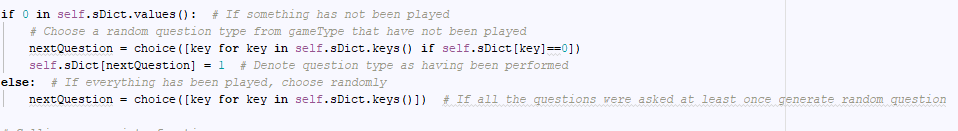
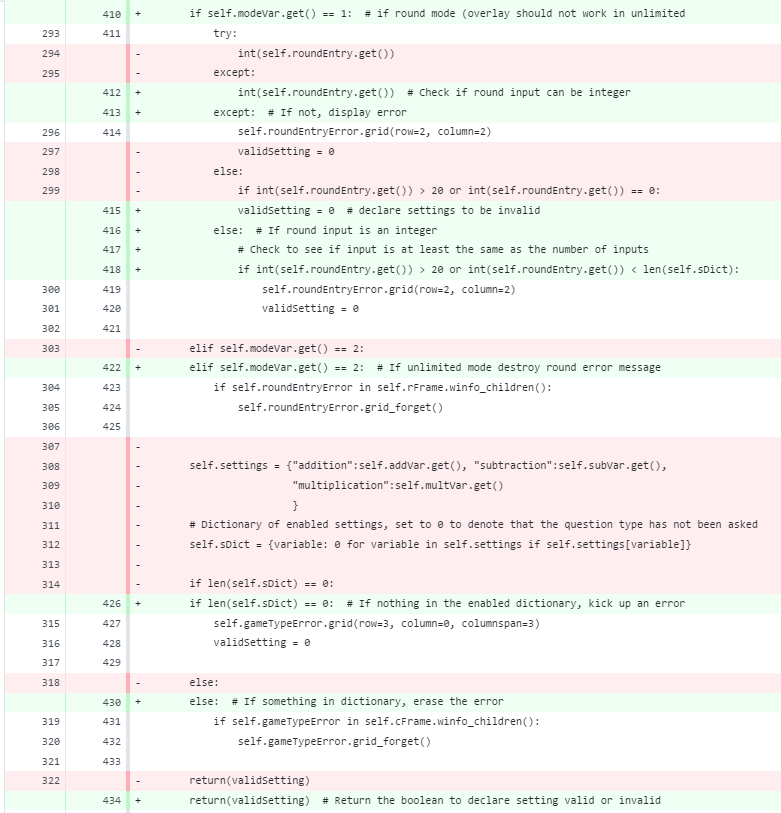
That’s basically it. I still need to finish the program off. Add a little finishing touch here and there. Add commenting, learn how to art and create backgrounds. Oh. And the big one. Update the flowchart, variable table and testing table to actually mimic the program. Great.

## Sep 2, 2017

I updated the flowchart, to a respectable degree. I didn’t create a box for every process, that would make it ridiculously long, so I squished some multiple processes into one box. I don’t really see much point in doing anything else. I’ve also added a few things to the code. To avoid the answer screen from trying to display a very long user input that’s far too big to format properly on the screen, I’ve limited the len of the input to 12 digits. If its above 12, it will kick up an error, 12 or below, it will accept the answer. Its probably a bit too long for what is possible as an answer, but I’ve decided to basically allow as long a limit as formatting would apply. This program is aimed, primary school children, and the more they get wrong by being able to input more numbers, the more they can learn. I’ve also added the functionality for the user to input the number of rounds they desire. Previously it was simply 10 rounds if it is round mode. With it however, came the need to add additional if statements for boundaries. I’ve decided to have a upper boundary value of 20 inclusive. Its large enough for a quiz, and any longer, you might as well go unlimited mode. The lower boundary value was a bit more tricky. The problem is, I’ve made an effort in the last version of the program to ensure that all of the game types the user selects will be played at least once. It was the purpose of this comprehension setting the values to 0:



To denote which variables were played for this code:



If I simply set a lower boundary value of 1 so that at least one question plays, if the user selects 3 question types, the user would not be able to play all 3 of the question types. Hence, to get around this error, the lower boundary value was set at the length of sDict, i.e. the number of enabled settings. This was also a testament to how easy the code was to expand on.

The green is the changes I had to make to the code checkSettting() to account for input errors and set boundary values, the giant block of red is because I moved that code up a bit. I don’t think I added more than 10 lines?

I did have to add a bit more in the windowSetup() class to acutally create the round entry, but all in all. It was really easy to work with. Segregating the two classes kept the code easy to work with. After the struggle of the last version of code, this was really good.

I also added another feature. In round mode, if you right click, you can enable an overlay which displays the previous questions. This is the code.

This was really difficult to set up. Not because of the coding. The coding is relatively simple. But just because of the formatting. I wanted to create a blue background on the overlay, so I coloured in the frame I was using. However doing that meant that the gaps between the gridded labels were showing. So I had to basically trial-and-error and mess around with the width of the labels. grid\_propagate simply makes sure that the grid stays the given size. Anchor centers the grid on the center of the frame, and place places the frame on the canvas. Lift ensures that it is above every other widget (This change was a product of error messages being on top of the overlay, because they were technically created last). I would normally use canvas.create\_window to place this on the canavs, instead of place, but unfortunately, lift is only available for widgets not canvas objects, and hence I needed to use it. It kinda ended up simplifying the code. Didn’t need a secondary overlay to denote the window in the canvas, it’s all just done with the frame. If you ignore the formatting, it was defininately easy to code. Segregating the variables based on function was definatly a good idea. I didn’t need to weave through any code I could just bash out the new GUI, make sure it looked ok, and figuring out how to actually get it to work with the background code.

The code is finished now. Going to add the new changes to the flowchart, and then work on the variable table.

Also, minor changes in the GUI forced a change in the sketch.

## Sep 8, 2017 – 9pm

Uhhhhhhhhhhhhhhh.

I now this is a formal internal assessment. So whoever is marking this, take that as a formal statement, for that is the only noise that can come out of me after spending 7+ hours across 2 days updating the variable table and the testing table.

I’ve added a few points where it was appropriate explaining the point of the testing, and the bug which forced its addition to the table.

Uhhhhhhhhhhhhhhhhhhhh.

## Sep 8, 2017 – 11pm

I’ve put everything relevant in one folder to make it easier to hand in.