Newlands College 2020

3TCD Internal Assessment: GIGO - Garbage In Garbage Out

AS 91906 v 1—Level 3, 6 credits (INT)
Use complex programming techniques to develop a computer program

Achievement Criteria		
Achievement/Paetae	**Merit/Kaiaka**	**Excellence/Kairangi**
Use complex programming techniques to develop a computer program.	Use complex programming techniques to develop an *informed* computer program.	Use complex programming techniques to develop a *refined* computer program.

Conditions:

- Time Allowed: Three weeks
- This assessment is due 12 Oct 2020.
- This is an Open Book Assessment. You may refer to your workbooks, Learning Log and previously worked programs. You may refer to sites such as Stack Overflow and python.org.
- You are encouraged to work in your own time, however, evidence of own work is required. You need to discuss with your teacher the work you intend to complete at home and then you need to show your teacher what you have achieved. Your Project Log must record this process as well.
- Students may only request assistance in the form of clarification or technical failure. No other assistance is to be given for coding or debugging.
- You may discuss your work, ideas and plans with other students – but may not show your work to other students.
- Submission:
  - All versions of program - by email or on usb stick
  - Project Log, including documentation of planning and testing/trialling - uploaded to Classroom
- All work submitted must be created by the candidate without assistance from any other person.
  Insert your name in the Authenticity Statement to acknowledge that this program is your own work:
  In completing this internal assessment, I agree that I will not:
    - *copy work from another student*
    - *collude with another student in the preparation of the assessment task*
    - *let another student copy my work*
    - *or copy from sources without the appropriate acknowledgement.*

I am fully aware that copying in such ways is plagiarism and that this is not allowed. I am aware that if I plagiarise, then the work will be awarded a 'Not Achieved' grade. I agree that this work will be my own.

Student signature:	Date:

Introduction/Kupu Arataki

This assessment activity requires you to write a computer program for a task selected by you incorporating complex programming techniques. Appendix 1 shows examples of complex programming techniques. You can use these or others as appropriate (approved by the teacher).

This is an individual assessment task. It is due on Monday 12 Oct 2020. You will be assessed on:

- whether your computer program meets the specifications of the task you have selected.
- the degree in which you meet the requirements for Achieved, Merit or Excellence detailed in the Achievement Standard (and reproduced in part below)

Task/Hei Mahi

Choose one of the following scenarios:

1. Data Capture Program: Capture and validate data records on an area of your choice (and agreed by the teacher).
   - It must incorporate a unique numeric identifier and at least 4 additional data fields of your choice
   - Produce an appropriate CSV file containing at least 10 records suitable for import into a MySQL database.

2. Student Report Generator: Write a program to generate Year 13 DT student reports from a range of standard comments. This will require generating appropriate correct 3 sentence report paragraphs for use by the teacher appropriate for use in KAMAR. A numeric student identifier is required.

3. Your Choice: Write a program of your choice (and agreed by the teacher) clearly incorporating at least two complex programming techniques detailed in Appendix 1. Text and numeric data entry will be required. eg BMI calculator, Maths Quiz, Star signs, Proverbs, Coffee/Takeaway Orders

Planning

Introduce your chosen scenario and the programming skills you will need to use to meet specifications.

The chosen scenario is Scenario 3 — develop your own choice. The chosen program is to create a Tkinter version of the popular game of 2048. The programming skills I will use are GUI's, reading from and writing to files, object-oriented programming and complex data structures.

Identify the "steps" you will take to develop this program

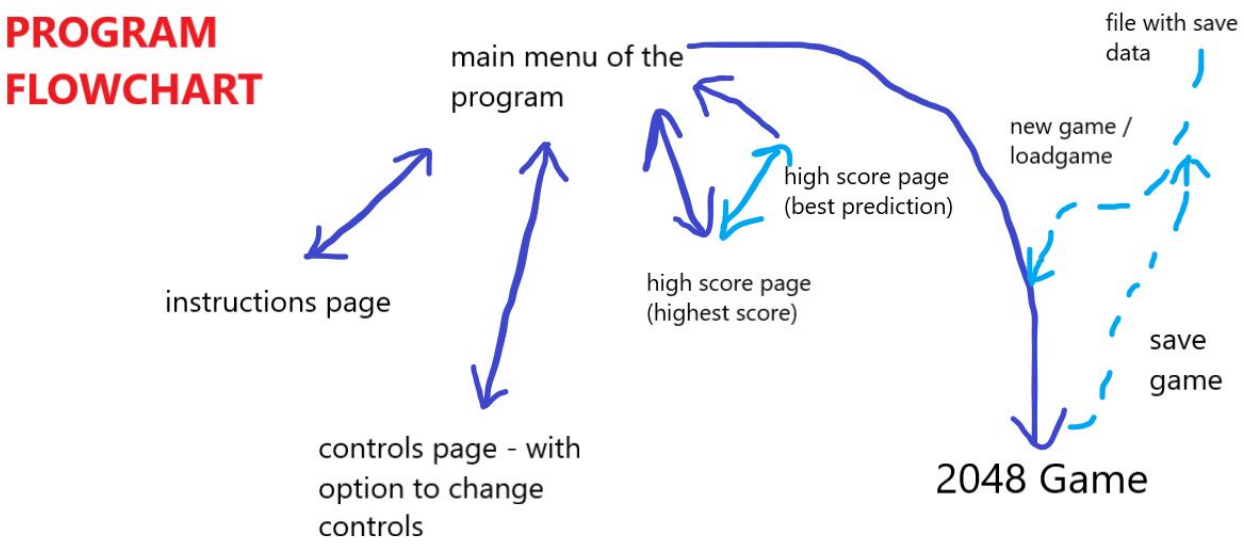
Initial plan:

The main component of the program is the 2048 game. This will be supported by other elements such as: highscore board, instructions page, page where you can change player controls. These elements will be tied together in a central main menu.

Component	Explanation
2048 game	Taken from the game developer website — "The objective of the **game** is to slide numbered tiles on a grid to combine them to create a tile with the number **2048**; however, one can continue to play the **game** after reaching the goal, creating tiles with larger numbers". Therefore, the final score of the player will be a weighted sum of the number of tiles the player has accumulated (higher tiles are weighted more). Before the game begins, players are asked to enter a number that is a prediction of their final score. The player will be able to save and quit at any time too.
Instructions page	The instructions of the game, which allow the player to slide numbered tiles, will be the arrow keys. This will allow the player to move all tiles (which are not against a wall) to the direction pressed. For example, if the left arrow button was pressed, then all free tiles are moved towards the left.
Controls page	The controls page will be the page which allows the player to alter their controls. For example, many players use WASD instead of arrow keys. However, many people also only can use the arrow keys. Hence, by providing flexibility to the user — it gives a better user experience.
Highscore page	The high score page will contain a leaderboard that displays — ranked — the highest scoring individuals of the game. It will also contain a separate high score table that contains the rankings for individuals with the best predictions (smallest absolute value between predicted and actual)

Drawing of the program

I have created a schematic drawing of the program to visualise what is happening between the components and to write the code in a more object oriented fashion.



Plan of construction through dependency analysis.
By looking at which components depend on each other, we can identify which components to work on first — saving time by not creating components and having to work on the required dependencies half way through.

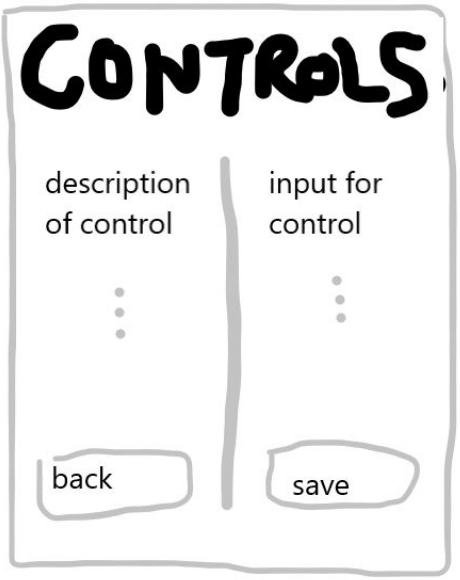
Page	Dependencies
Main menu	None
Instructions	Main menu (so it can go back)
Controls page	Main menu (so it can go back)
High score page (ranked and predicting)	2048 Game
2048 Game	Main Menu (so it can go back)
Save functionality	2048 Game
Pause functionality	2048 Game

Therefore, the development plan is as follows:
Main menu — Instructions — Controls Page — 2048 Game — Save Functionality —High Score page.

Deciding on the graphical theme of the game and creating initial designs
Here are wireframes of the game pages.

Main page will have a vertical block menu. Load game options will be greyed out, if there is no game load available.

Instructions to the game will be about getting the highest score, and how players can slide the tile around to achieve this.



Controls will have inputs for each control type and the ability to save changes and go back to the main menu

Main game program. Ability to save is at the bottom of the page. Stats about the current game are displayed at the top.

Testing plan for each component

If numeric / text input is used, boundary values will be checked and standard values. If the component is graphical, peer to peer testing (user testing) will be used.

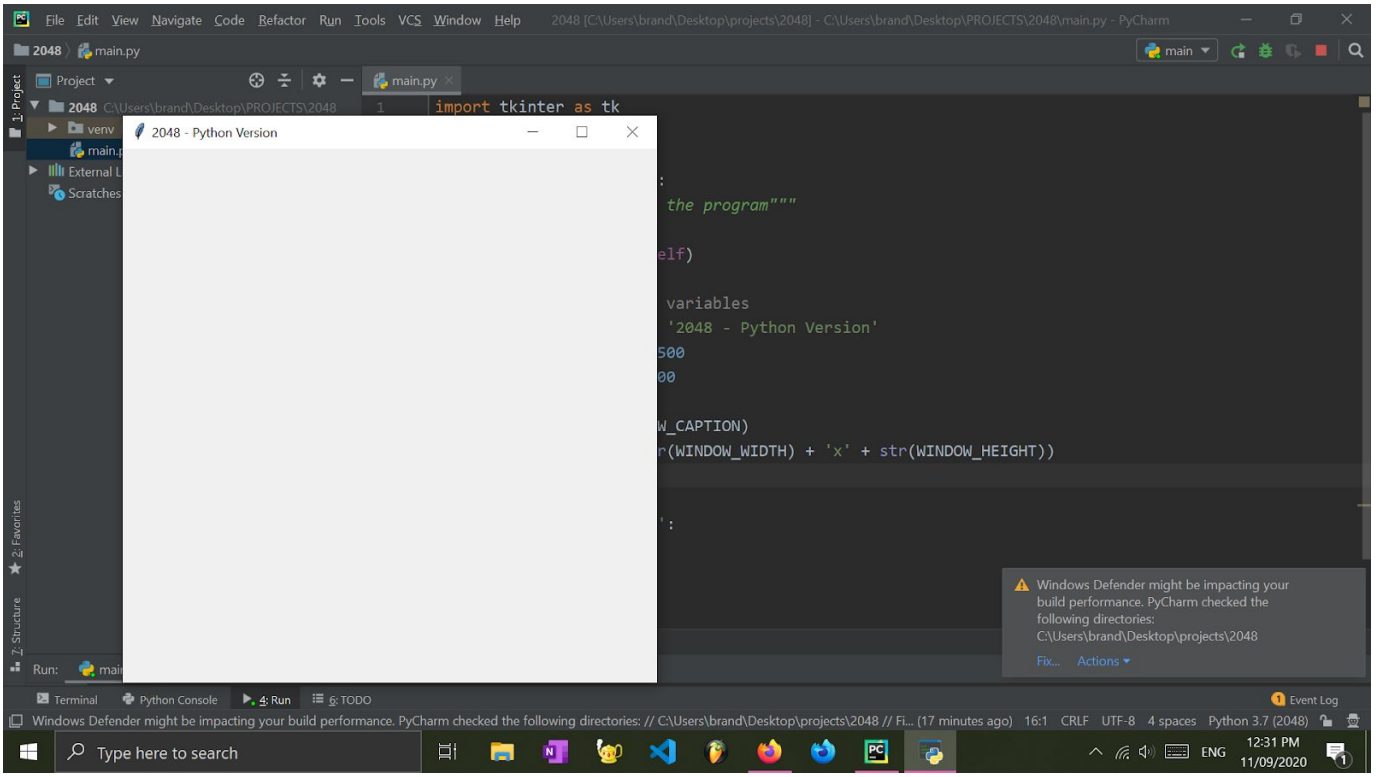
Version One: Program Skeleton + Menu

COMPONENT #1 - Main program

All pages will be wrapped in the MainProgram class, inheriting from Tk() class from tkinter module.

```
import tkinter as tk


class MainProgram(tk.Tk):
    """This class drives the program"""
    def __init__(self):
        tk.Tk.__init__(self)

        # program window variables
        WINDOW_CAPTION = '2048 - Python Version'
        WINDOW_HEIGHT = 500
        WINDOW_WIDTH = 500

        self.title(WINDOW_CAPTION)
        self.geometry(str(WINDOW_WIDTH) + 'x' + str(WINDOW_HEIGHT))


if __name__ == '__main__':
    app = MainProgram()
    app.mainloop()
```

This code creates a small window of size 500x500. However, this spawns the window in the top left of the window.

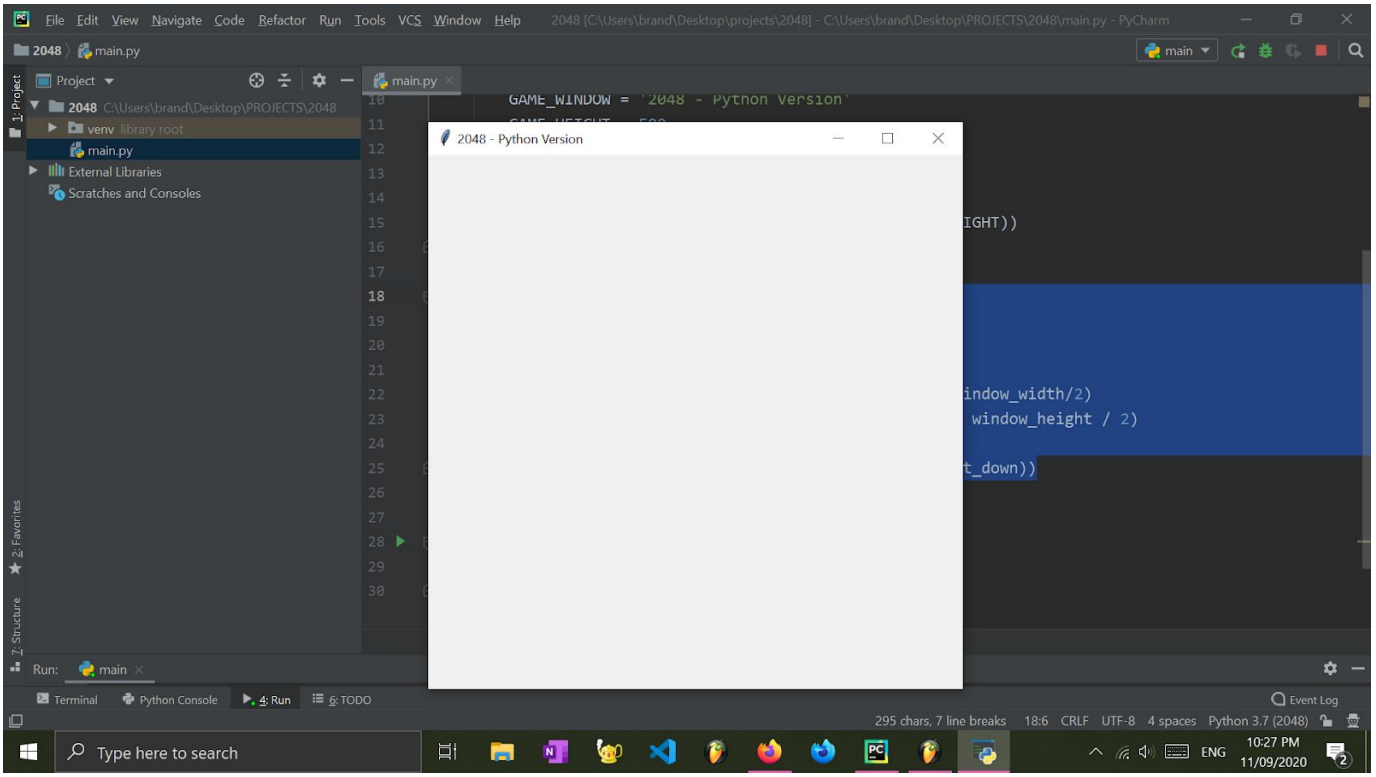


This is something I don't want, and it is something I want to spawn in the middle of the window. Thus, I have written a method that allows the program to do this. It offsets the window from (0, 0) (top left corner) by half of the screen width and height, resulting in a centralisation of the window spawned.

The main menu page will have a MainMenu class, inheriting from the Frame class from tkinter.

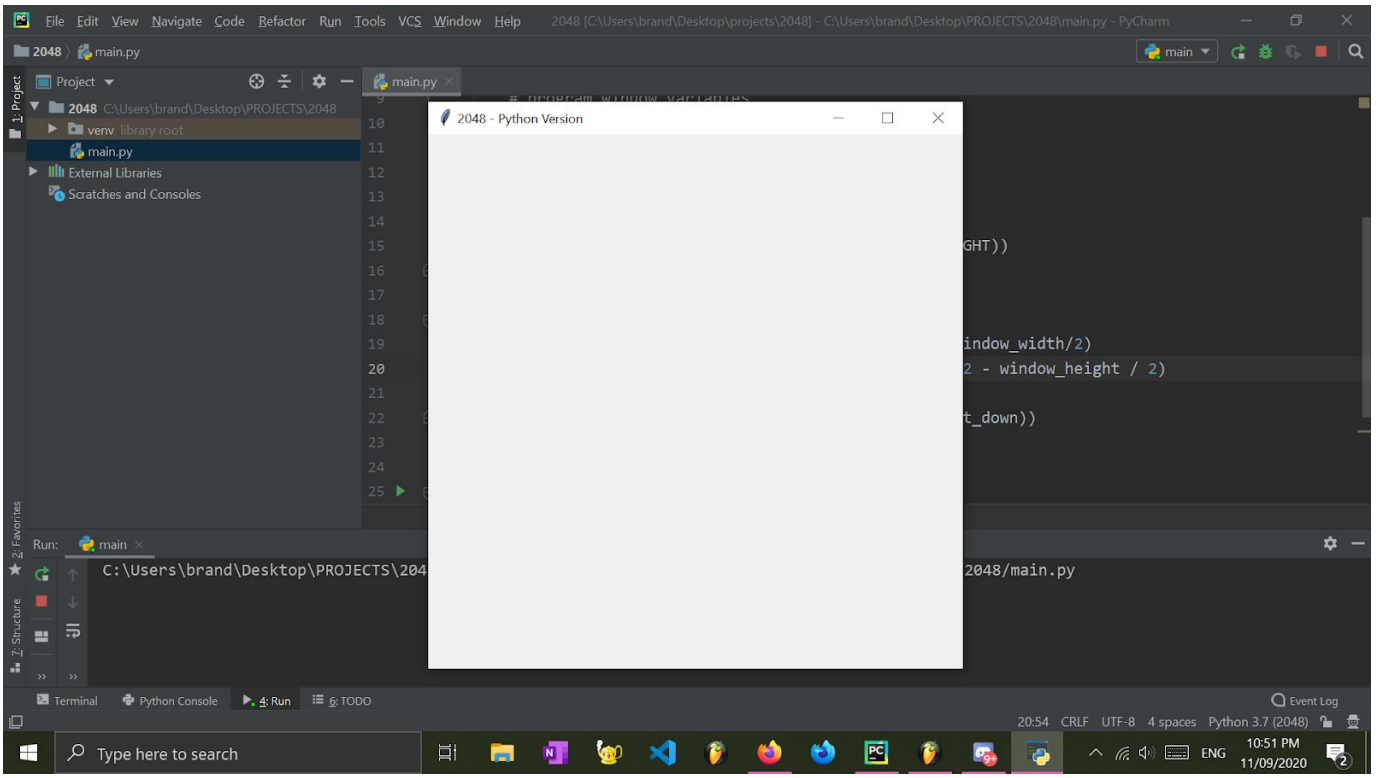
```
def center_screen(self):
    window_height = 500
    window_width = 500

    offset_right = int(self.winfo_screenwidth()/2 - window_width/2)
    offset_down = int(self.winfo_screenheight() / 2 - window_height / 2)

    self.geometry('+{}+{}'.format(offset_right, offset_down))
```

However, upon running the script, the centralisation doesn't work as expected. The screen is centralised horizontally but not vertically.



I suspect this is because of the nature of the window geometry. The dimensions of the window are 500x500 but the window spawned does not seem square. As the centering method uses the actual dimensions of the window, I suspect the centralisation may have overshot because of this discrepancy. However, upon looking closely (and using a physical ruler), it appears this window is actually centred. The fact it isn't centered is because I haven't included the taskbar in my visual assessment.

After some research, the taskbar is 40pixels tall on windows 10. Therefore, I will subtract 40 off the window height parameter to accommodate for this.



I then created the mainframe of the window and named this ‘container’. This will be the main container of the window. The reason I have done this is it allows me to create a program wide margin easiy.

```
# declare master frame
GAME_MARGIN_X = 0
GAME_MARGIN_Y = 0
self.container = tk.Frame(self)
self.container.pack(padx=GAME_MARGIN_X, pady=GAME_MARGIN_Y)

# current frame used
self.current_frame = None
```

I then wrote another MainProgram method, which passed a Frame object class, will destroy the current frame and create a new one (the passed frame) and this method is called show_frame.

```
def show_frame(self, frame_class):
    """displays given frame"""
    if self.current_frame is not None:
        self.current_frame.destroy()

    frame = frame_class(self.container, self)
    frame.grid(row=0, column=0, sticky='news')
    self.current_frame = frame
```

This allows for quick switching of different frames, in the event a user wants to navigate between pages in an object oriented manner. Each frame object is inside self.container and is controlled by self of MainProgram which is the Tk master (as Main Program inherits Tk).

COMPONENT #2 - Main menu

For the main menu class, this will inherit tk.Frame (as each page is a frame).

```
class MainMenu(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller
        self.display_background()
```

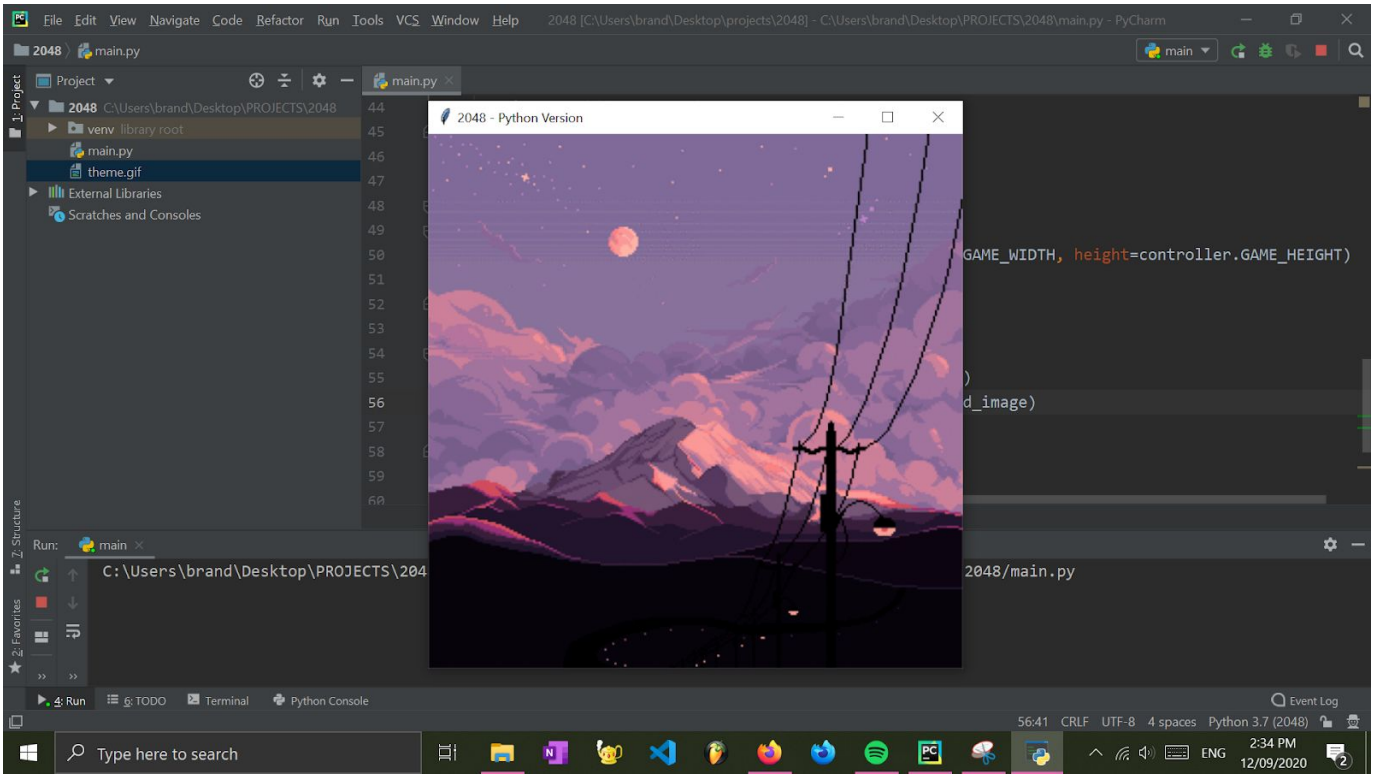
Controller is the MainProgram class. I then decided to test out if I could display a background image in the frame. Therefore, I created the display_background method for the MainMenu class and tested it out with a 500x500 picture. The image is managed with the geometry manager place not pack nor grid, as place allows me to insert the image absolutely (similar to position: absolute in CSS).

```
def display_background(self):
    background_image = tk.PhotoImage(file='theme.gif')
    background_label = tk.Label(self, image=background_image)
    background_label.place(relwidth=1, relheight=1)
    background_label.image = background_image
```

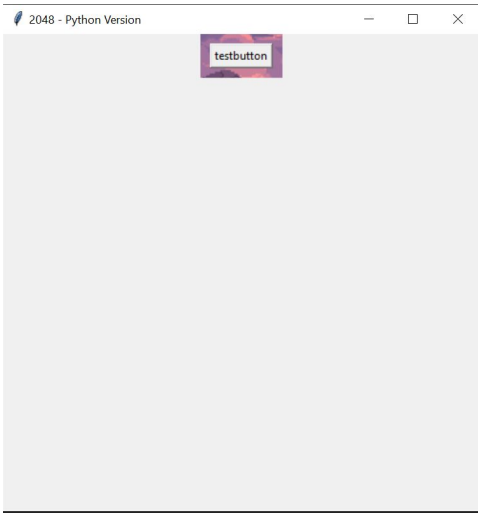
However, upon running the program, the image did not show up. This was perplexing. After some debugging, by printing out key statements, I was able to identify the problem. The Main Menu frame whilst initialised, was empty, therefore had a size of 0. Subsequently, by placing an object in, the frame did not stretch to fill and the image could not be seen. THis was fixed by initializing the tk.Frame object with height and width parameters that can be taken from the MainProgram class (controller).

```
def __init__(self, parent, controller):
    tk.Frame.__init__(self, parent, width=controller.GAME_WIDTH, height=controller.GAME_HEIGHT)
    self.controller = controller
    self.display_background()
```

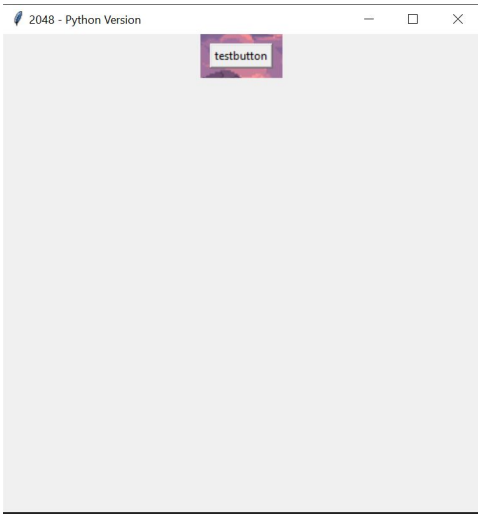
This means the main menu frame will fill the container it was in regardless on whether it is empty or not. After running the code, the image was displayed.



Note: this is a testing image.



However, I run into problems immediately as I try to add buttons to the main menu frame. It appears to just not let me at all. This is because the button is resizing the content of the frame. Therefore, a solution I will attempt is to migrate the background image rendering to the parent container (MainProgram.container).



Clearly, this did not work either and failed to solve the initial problem. After some research, I stumbled across an article on StackOverflow which addresses the problem. The proposed solution was a canvas drawn underneath with the image drawn on. Image behind buttons in tkinter (PhotoImage). After some tinkering with tkinter, the solution worked. I have also downloaded an external library using pip, Pillow, the Python Image Library as tkinter did not support png/jpg images.

```
class MainMenu(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent, width=controller.GAME_WIDTH, height=controller.GAME_HEIGHT)
        self.controller = controller

        # create canvas
        self.canvas = tk.Canvas(
            self,
            bd=-2,
            height=self.controller.GAME_HEIGHT,
            width=self.controller.GAME_WIDTH)
        self.canvas.pack(expand=tk.YES, fill=tk.BOTH)

        # display background
        self.display_background('theme.gif')
```

I added the canvas declaration, creating a canvas on MainMenu. I then call display_background to paint the background in the canvas. The display_background method has been updated to allow for jpg / png image processing.

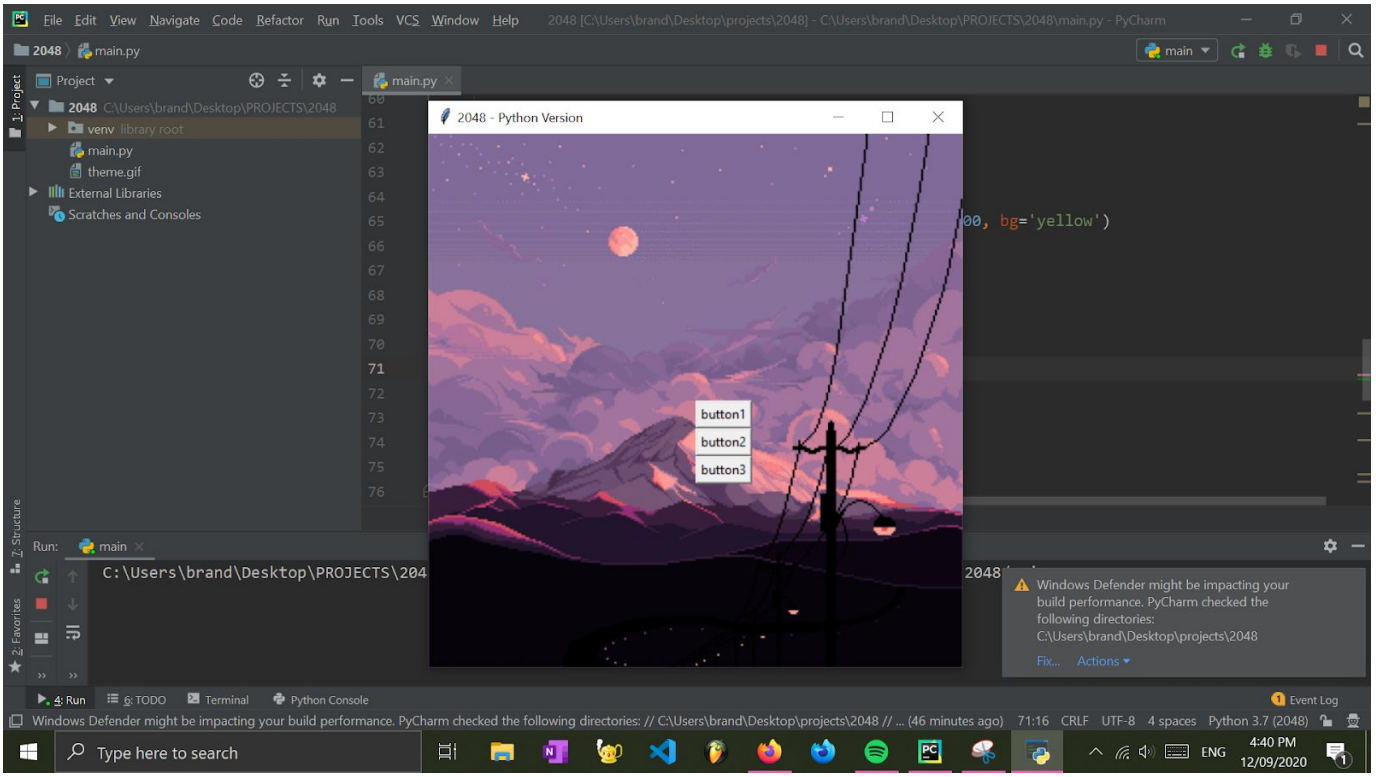
```
def display_background(self, imagepath):
    # draws and paints the background with image of given path
    background_image = Image.open(imagepath)
    self.canvas.image = ImageTk.PhotoImage(background_image)
    self.canvas.create_image((0, 0), image=self.canvas.image, anchor='nw')
```

I then drew buttons on top of the canvas with the display_button_on_canvas method which creates the object on the canvas.

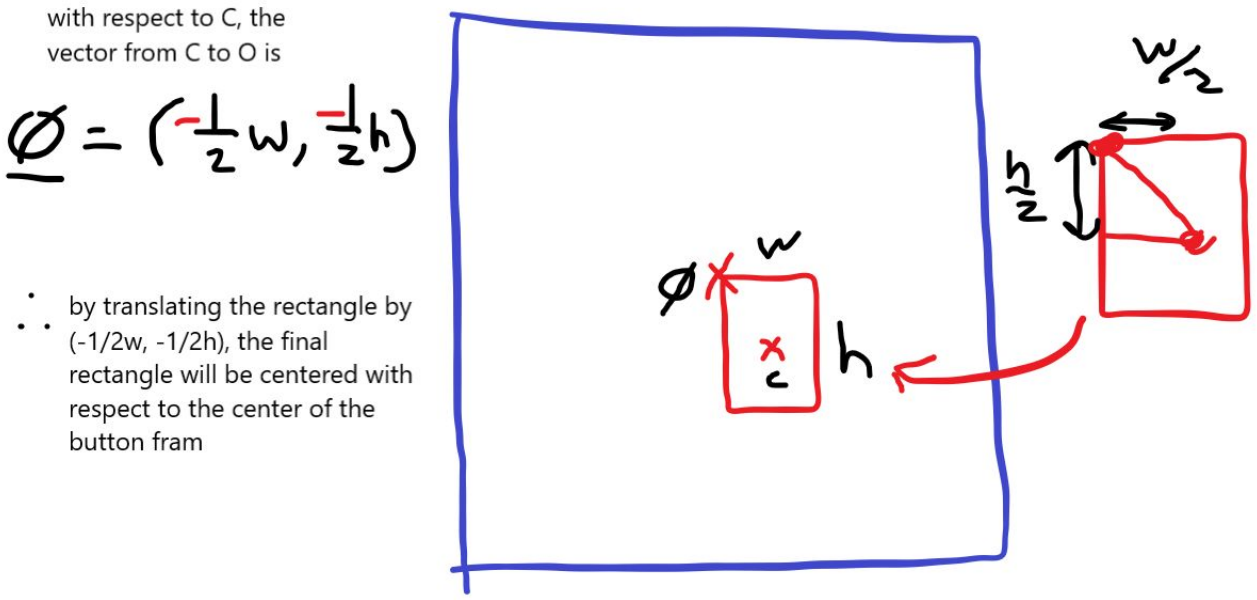
```
# displays all the buttons on main menu
button1 = tk.Button(self, text='testing button')
self.display_button_on_canvas(button1, 10, 10)
```

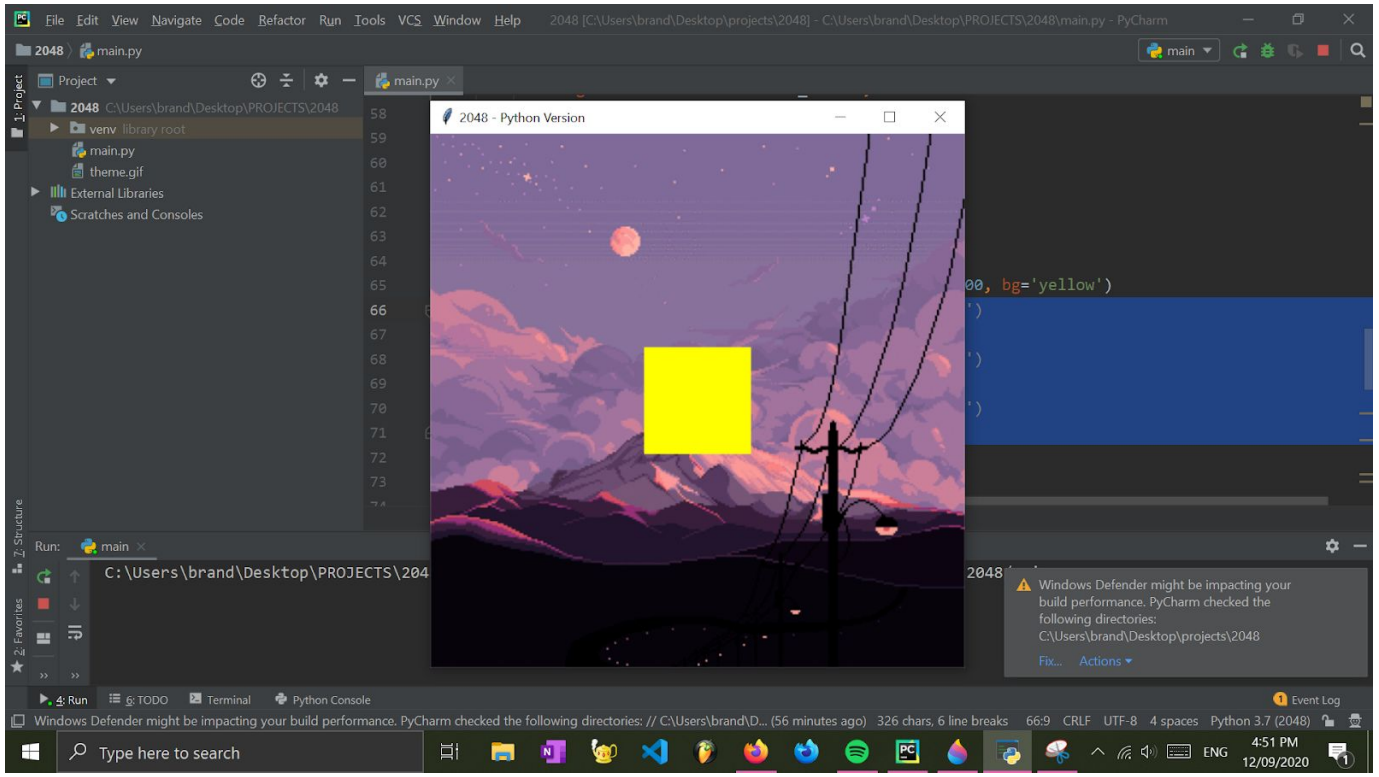
```
def display_button_on_canvas(self, button_object, x, y):
    button1_window = self.canvas.create_window(
        x,
        y,
        anchor='nw',
        window=button_object)
```

The drawback of this method is that the buttons need to be arranged manually via absolute positioning. For example, in the screenshot prior, the button is at (10, 10). One solution is to draw a frame that has buttons arranged inside it with a grid. This way, only a frame is required to be packed through absolute positioning.

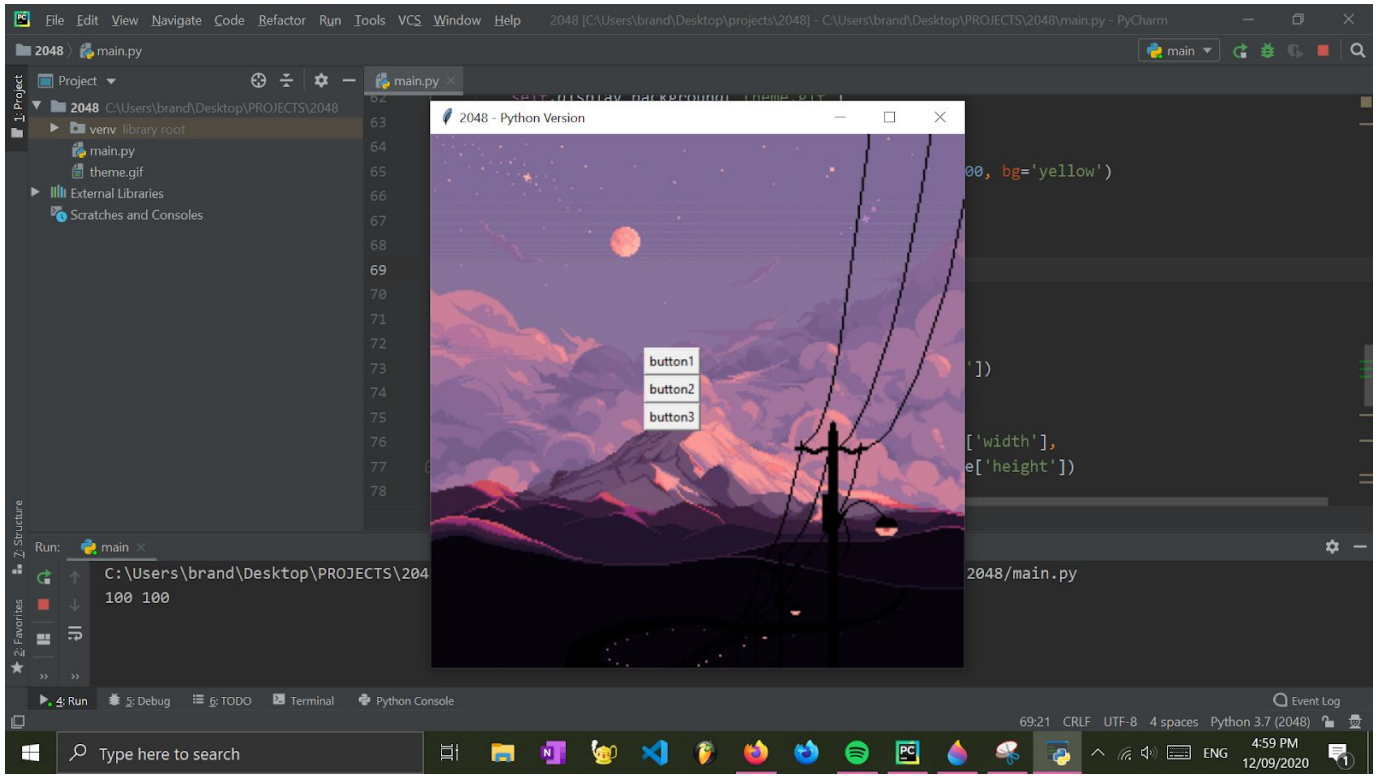


This solution worked, and now I need a method to dynamically center the frame using absolute coordinates. At the moment the frame is centered with respect to the top left corner of the button container. I would like the button container's center to be the point it is centered with respect to.

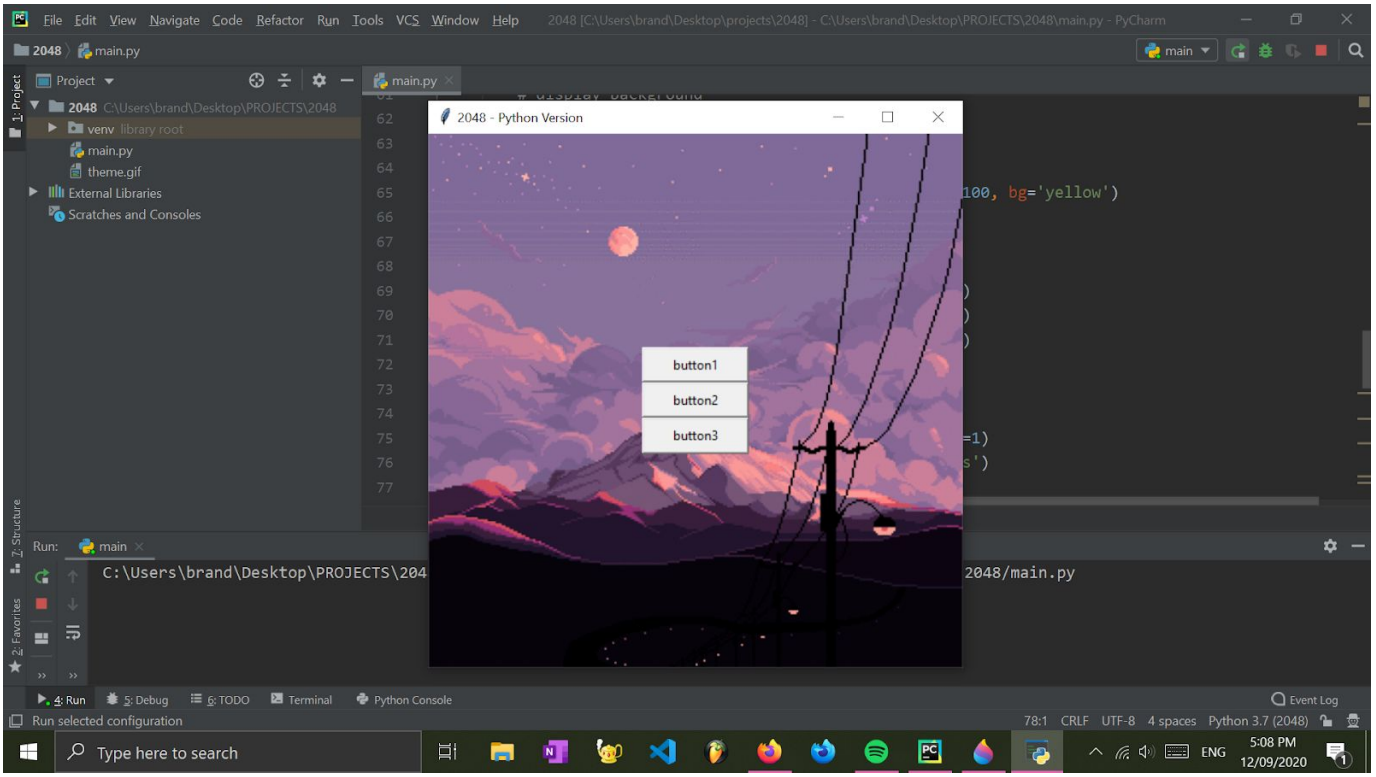




However, upon adding the buttons in, the centering math didn't work? Furthermore, the yellow background of the frame disappeared.



Upon printing some values (notably, the width and height of the window), I realised the frame was being resized after elements were added but the supposed width and height remained the same. This would have led the page to be centered wrong. Hence, I turned grid/pack propagation off, which means the frame does not resize upon child addition. Furthermore, I added a weight to each of the rows and columns which allowed the buttons to grow.



Code, which has been been placed in a loop for readability and efficiency.

```
    # create button frame
    button_frame = tk.Frame(self, width=100, height=100, bg='yellow')
    button_frame.grid_columnconfigure(0, weight=1)
    button_frame.grid_propagate(False)

    # create buttons
    button1 = tk.Button(button_frame, text='button1')
    button2 = tk.Button(button_frame, text='button2')
    button3 = tk.Button(button_frame, text='button3')
    buttons = [button1, button2, button3]

    # grid buttons into the button frame
    for index, button in enumerate(buttons):
        button_frame.grid_rowconfigure(index, weight=1)
        button.grid(row=index, column=0, sticky='news')

    self.display_object_on_canvas(
        button_frame,
        controller.GAME_WIDTH // 2 - 0.5*button_frame['width'],
        controller.GAME_HEIGHT // 2 - 0.5*button_frame['height'])
```

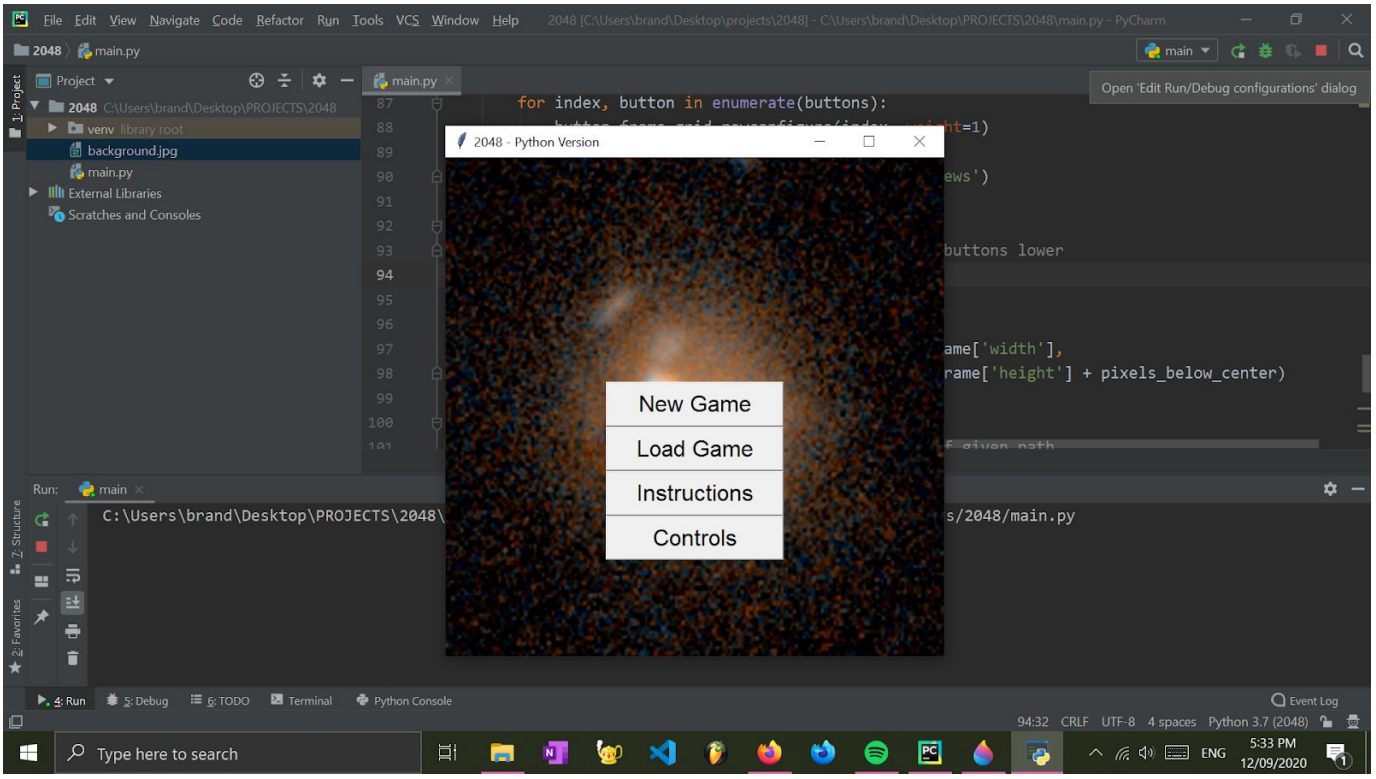
Now that a minimum viable design of the main menu has been created, I will now fill in the buttons (and rename their variables) and design a proper background for the game.

```
# initialise all program wide fonts
self.BUTTON_FONT = Font(family="Helvetica", size=16)
```

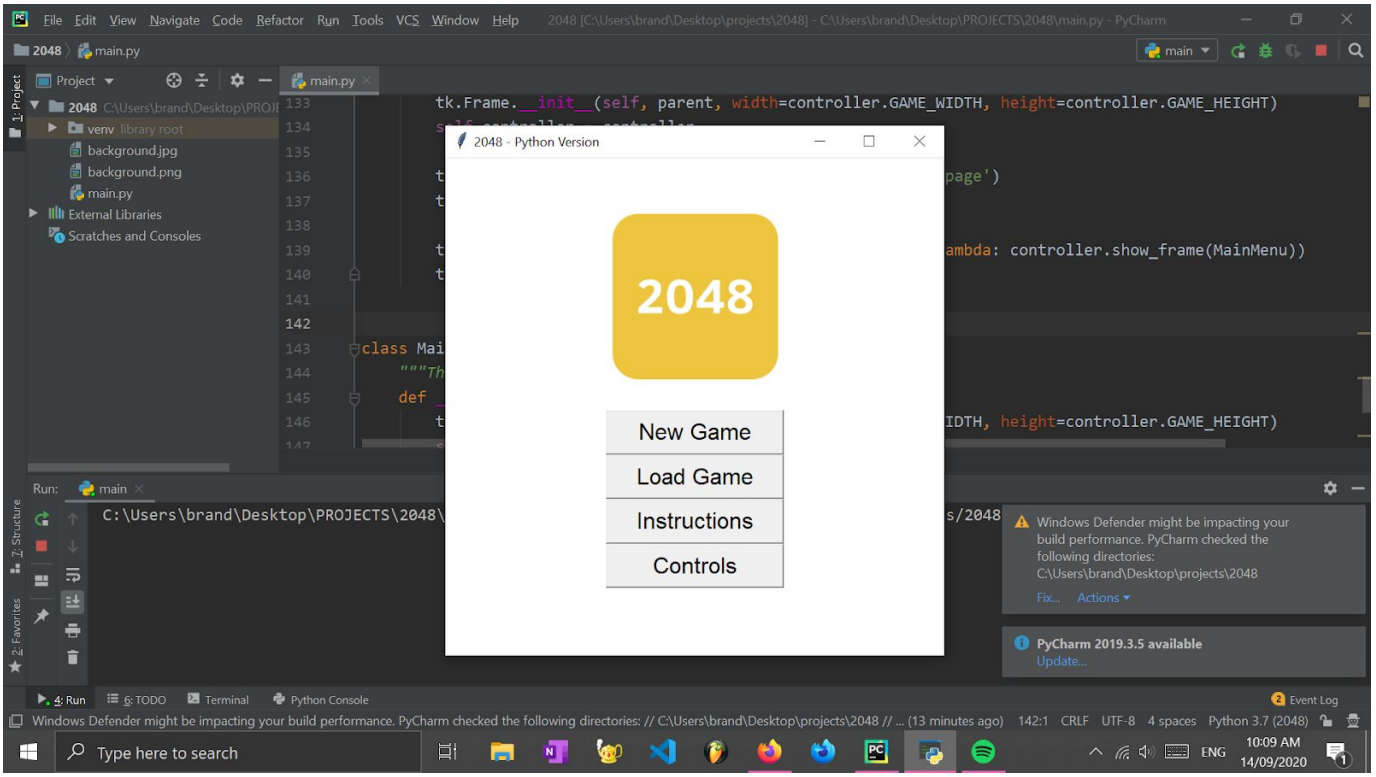
Furthermore, I noted that the text of the tkinter program was espcially blurry. This was fixed by making the program dpi aware.

```
# fix blurry font
from ctypes import windll
windll.shcore.SetProcessDpiAwareness(1)
```

Here are the updated buttons. Now I will design a proper background image for the game.

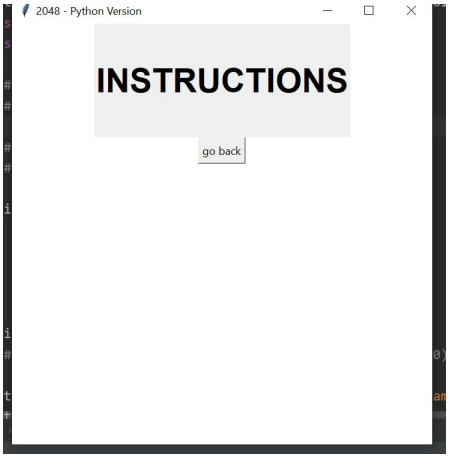


Here is the final design of the game.



COMPONENT #2 - Instructions

I then produced the skeleton of the instructions page. I also wanted to have the instructions page to have a background

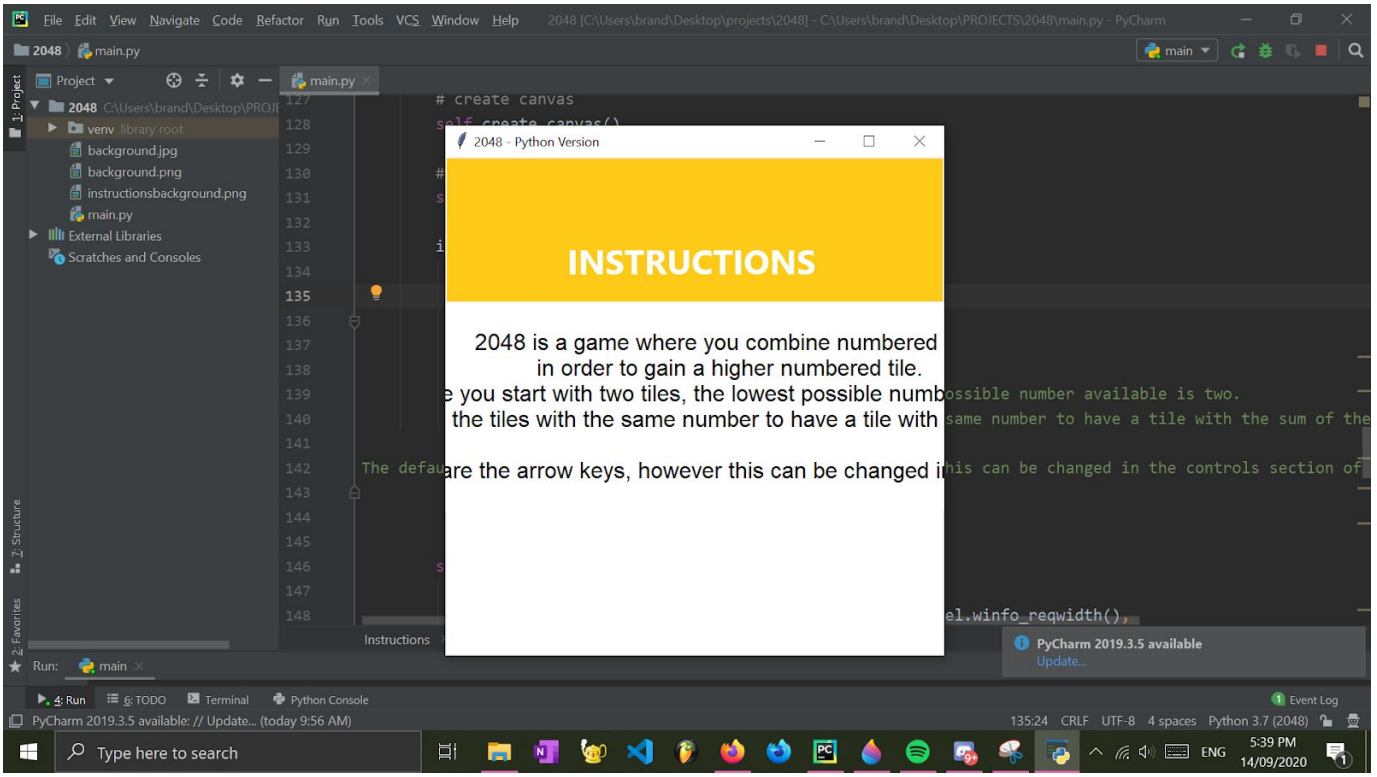


The problem was, the instructions label did not have a transparent background option in tkinter, and I wanted it to have a non square background and I planned to add this in a background image. In order to address this, I would simply have the text label in the background image and not through tkinter.

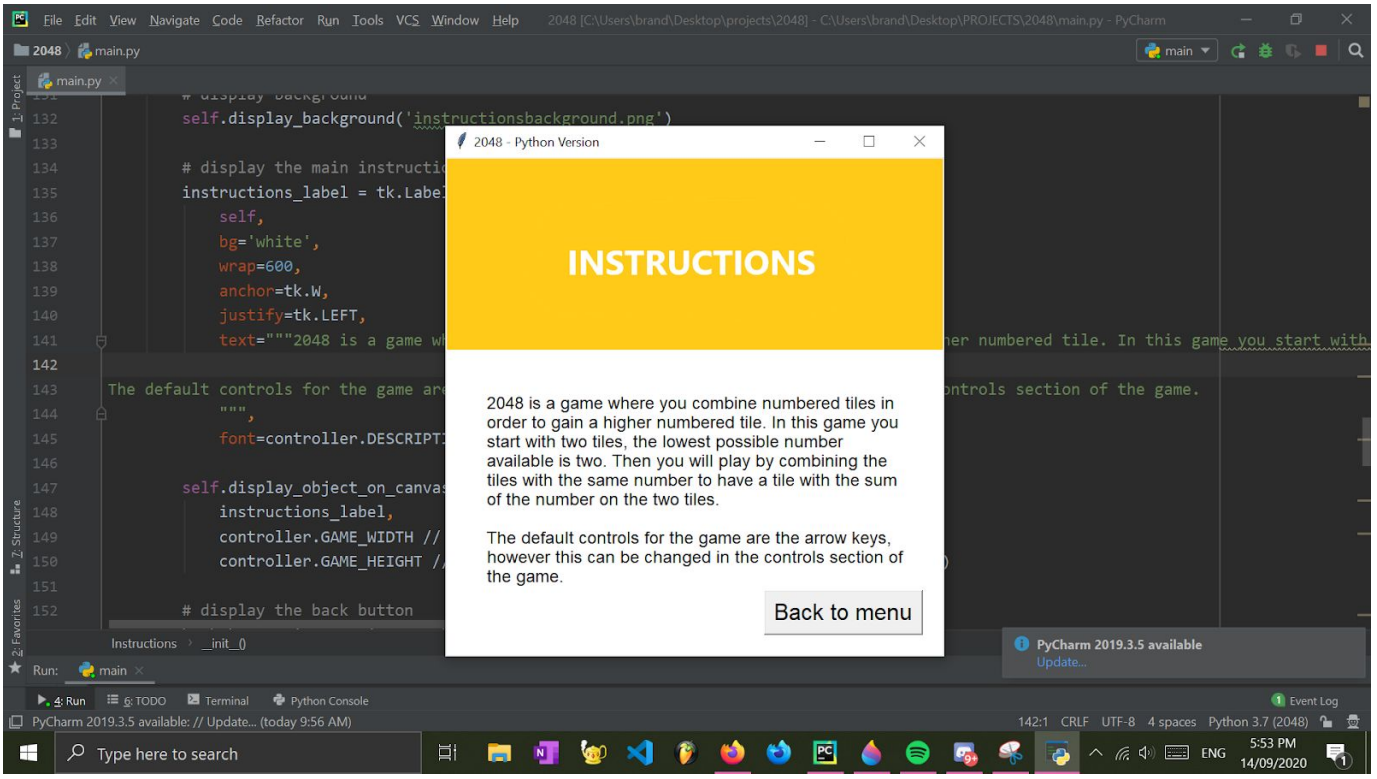
The background image designed.

The actual instructions are as follows:
“2048 is a game where you combine numbered tiles in order to gain a higher numbered tile. In this game you start with two tiles, the lowest possible number available is two. Then you will play by combining the tiles with the same number to have a tile with the sum of the number on the two tiles.”

“The default controls for the game are the arrow keys, however this can be changed in the controls section of the game\”

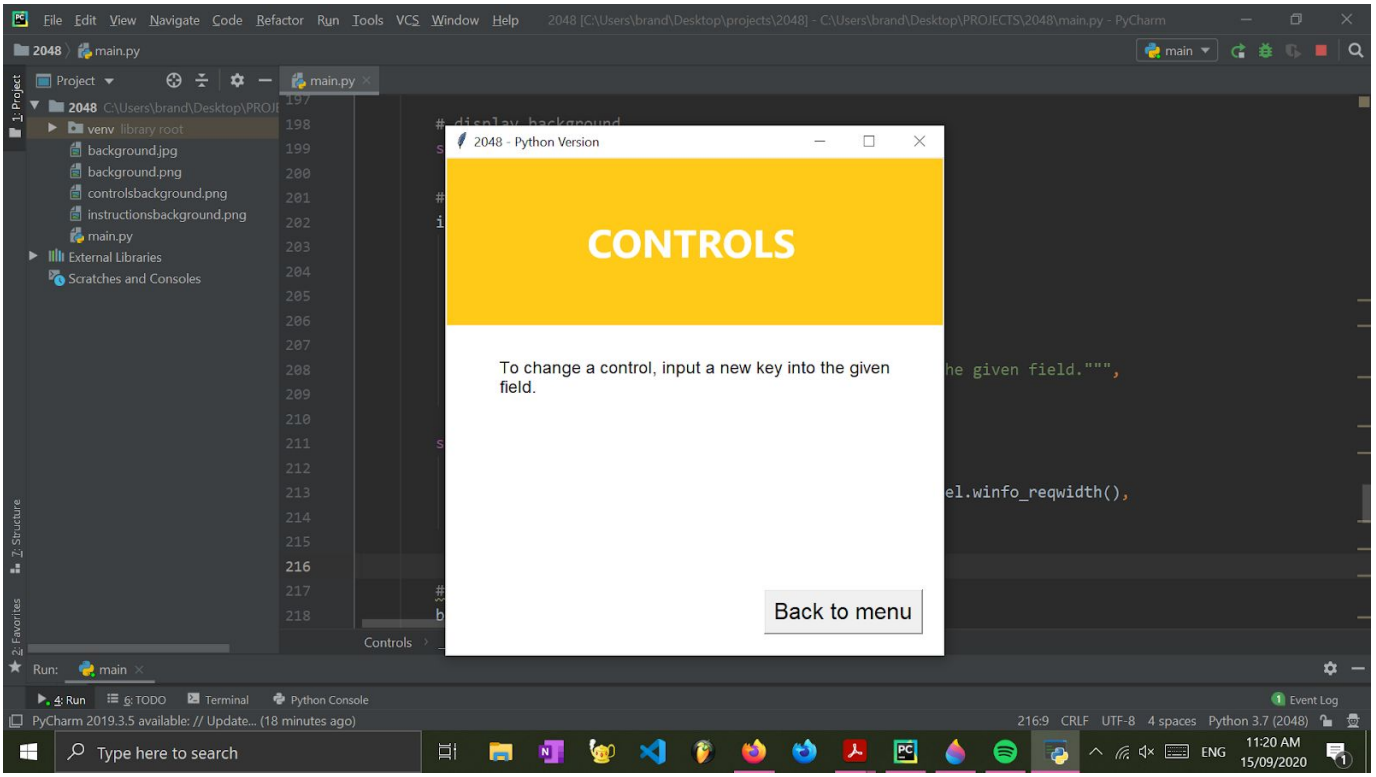


Initially, however, the text did not wrap. However, this was easily resolved by adding the wrap parameter to the label object created. I also wanted the text to be left aligned so I added an anchor WEST and justified LEFT to the label contents. Furthermore, I added a back button so the player can get back. Instructions page complete.

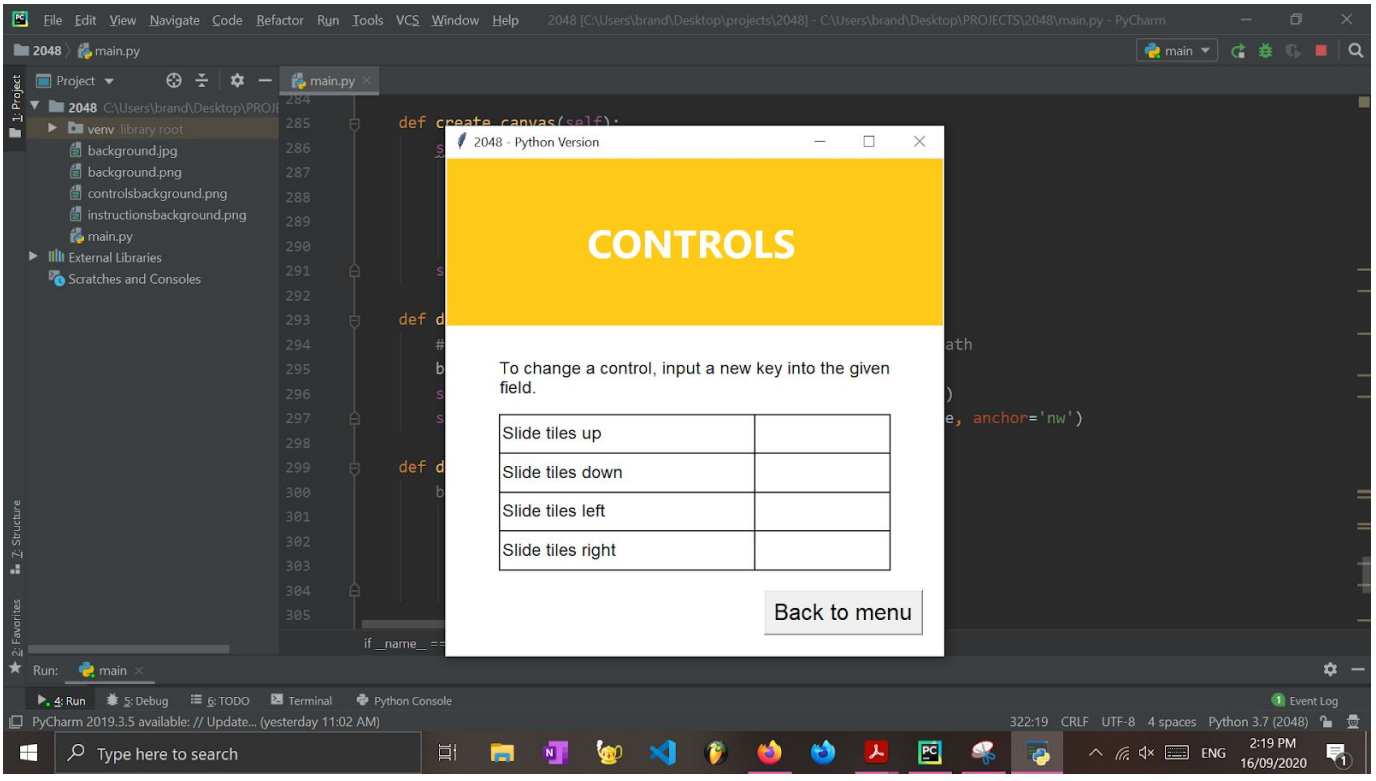


COMPONENT #3 - Changing controls.

I first drew out another background for the controls page. I then added a label for the controls page which directs the user in what they needed to do. This was mostly reusing code from the instructions page.



To create a border, I added padding to grid elements against a black background.



Then I added the code that drives the control picking mechanism. Firstly, I declared some frame (Controls page) specific variables in the initialisation function of the page — the slide up/down/left/right control and the is_binding variable.

```
# declare the control variables
self.slide_up_control = 'Up'
self.slide_down_control = 'Down'
self.slide_left_control = 'Left'
self.slide_right_control = 'Right'

# is in the process of binding
self.is_binding = False
```

The control variables will be changed by the button, and then written to a save file later on. The is_binding boolean is to let the program know if it is already choosing a control for a given direction, and it can't begin another process until the last one has finished.

```
# Allow all the inputs to have this switch_key ability on click
control_up_input = tk.Button(table_2x4, text='Up arrow key', command=lambda: switch_keys(control_up_input, 'up'))
control_down_input = tk.Button(table_2x4, text='Down arrow key', command=lambda: switch_keys(control_down_input, 'down'))
control_left_input = tk.Button(table_2x4, text='Left arrow key', command=lambda: switch_keys(control_left_input, 'left'))
control_right_input = tk.Button(table_2x4, text='Right arrow key', command=lambda: switch_keys(control_right_input, 'right')
```

I then mapped a parameter specific function, switch_keys, to each of the buttons. On click, it would call this function, and retrieve the last pressed key by the user and save it to the variables declared beforehand.

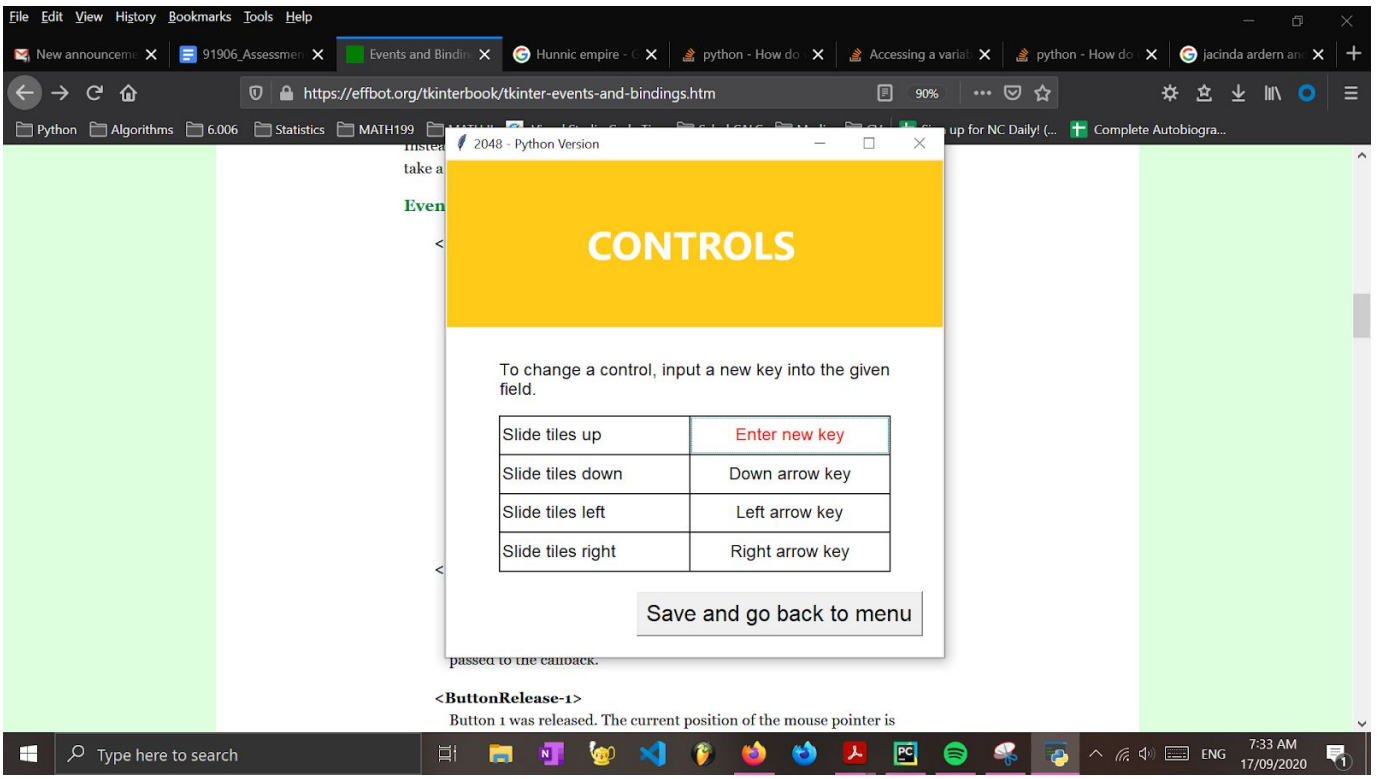
Inner workings of switch_keys:

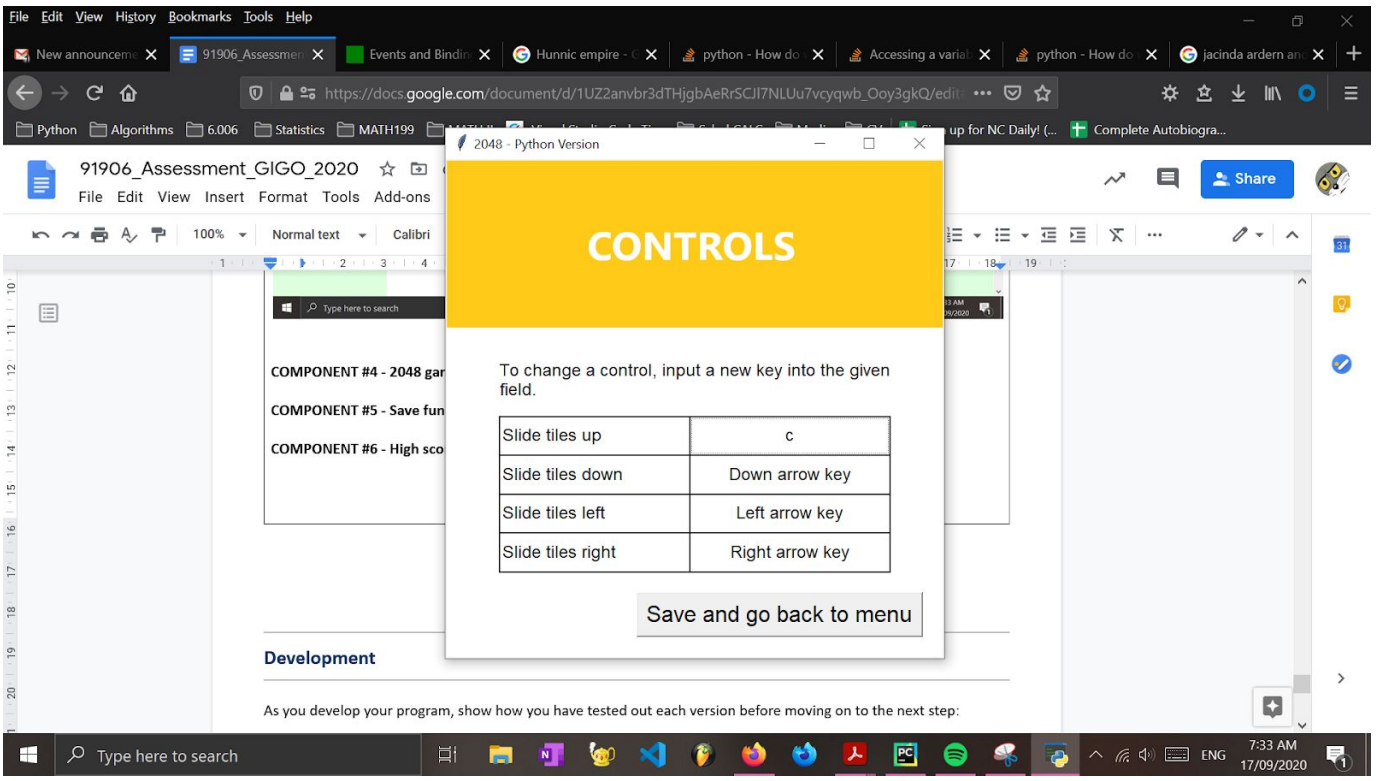
On click of the button, either display 'Enter a new key' (if it hasn't been clicked and another choosing isn't happening), switch back to the current value it had and terminate the choosing (if choosing was already in process and it was clicked again) or simply do nothing, (if another choosing was in process). These different conditional paths are so that the program can respond to different user inputs and act as a robust program.

If the control change is occuring though, is_binding is set to true, I bind all pressable keys and the special keys of the up/down/left/right arrows to the button. Upon the first click, I save this first click to the initialized variable in self prior, and unbind all keys mapped prior, so that it doesn't change anymore and set is_binding to false again.

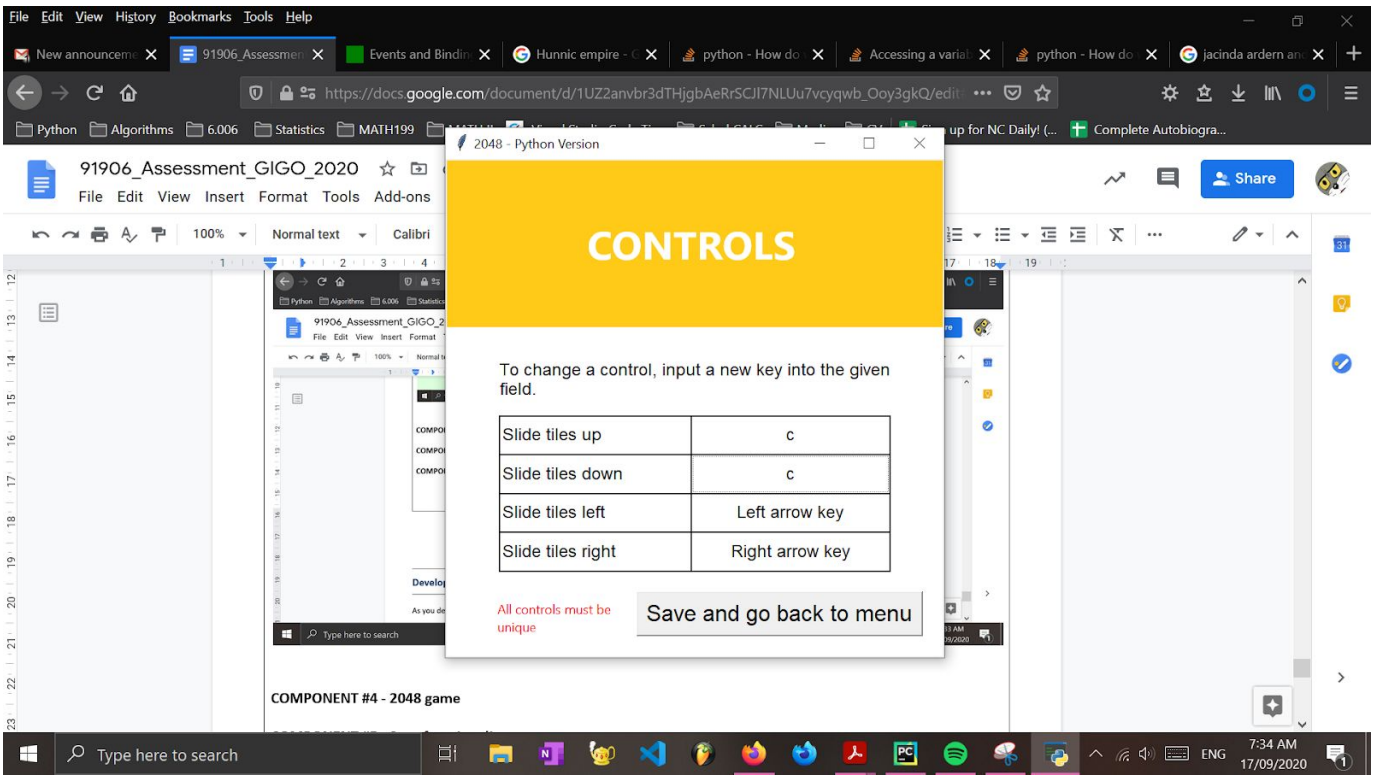
Upon clicking save, the program will attempt to check that all of the controls are unique. If so, it is able to exit to the main menu (and in a future alteration, will allow for saving). Otherwise, two possible error messages can be displayed. One for when the number of unique controls does not equal to 4. One is for when a binding process is occurring.

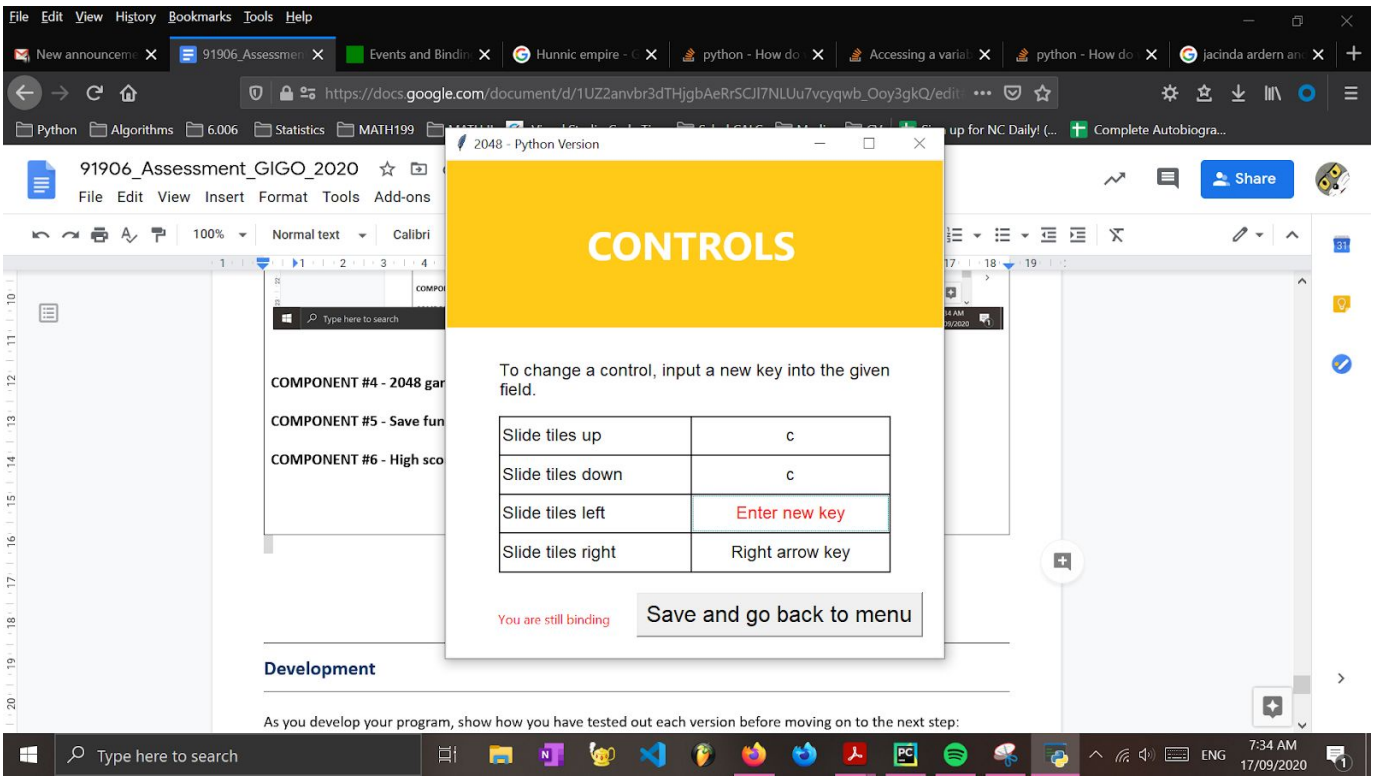
Here is the finished example.





Error messages


2048 - Python Version
CONTROLS
To change a control, input a new key into the given field.
Slide tiles up
c
Slide tiles down
c
Slide tiles left
Left arrow key
Slide tiles right
Right arrow key
All controls must be unique
Save and go back to menu
COMPONENT #4 - 2048 game


2048 - Python Version
CONTROLS
To change a control, input a new key into the given field.
Slide tiles up
c
Slide tiles down
c
Slide tiles left
Enter new key
Slide tiles right
Right arrow key
You are still binding
Save and go back to menu
COMPONENT #4 - 2048 gar
COMPONENT #5 - Save fun
COMPONENT #6 - High sco
Development
As you develop your program, show how you have tested out each version before moving on to the next step:

Version Two: Actual gameplay

\Development of the actual game.

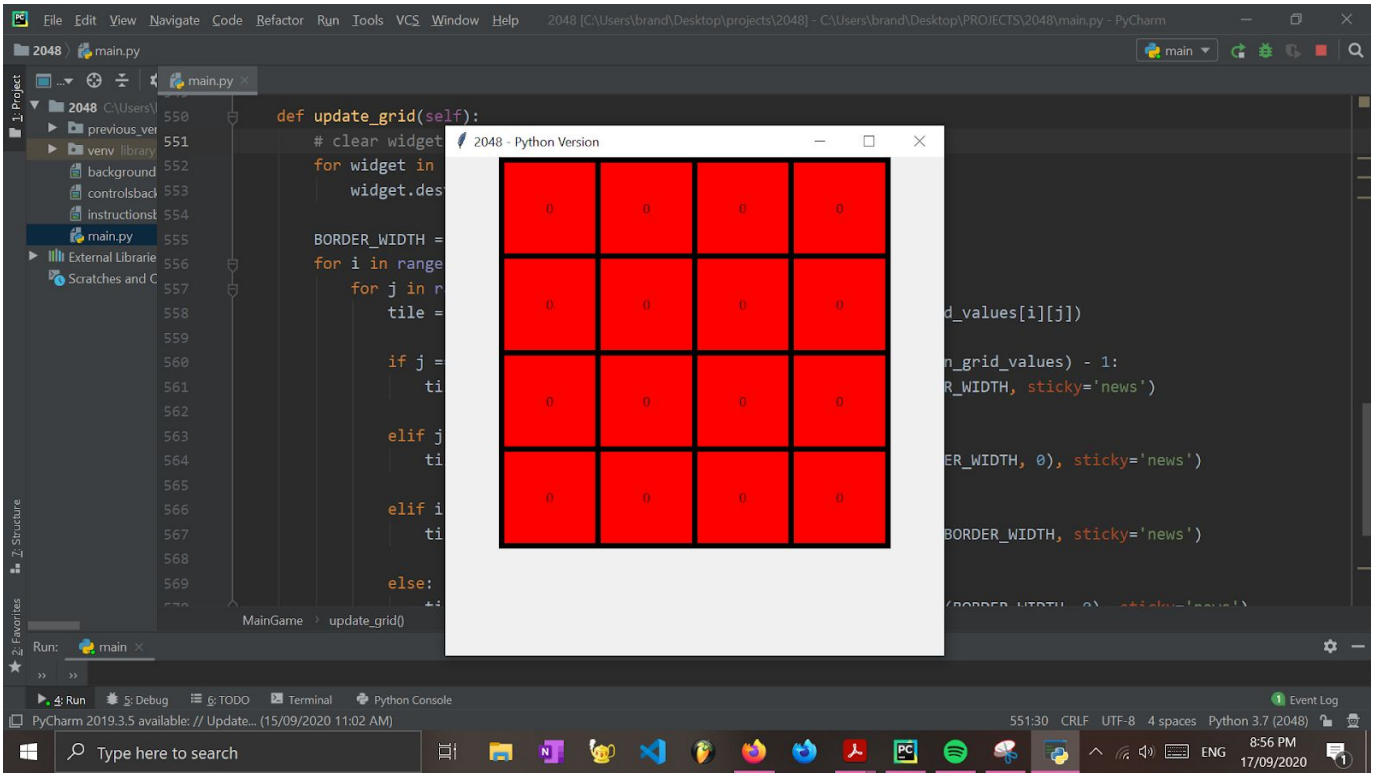
I first created the frame to hold the tiles. I then created a matrix to hold the values of each tile according to its coordinate (0,0) being the top left tile, (0,1) being the tile to the very right of this and so on.

```
# create the grid frame
self.GRID_WIDTH = 550
self.TILES_PER_ROW = 4

self.main_grid = tk.Frame(self, width=self.GRID_WIDTH, height=self.GRID_WIDTH, bg='black')
self.main_grid.grid_propagate(0)

for i in range(self.TILES_PER_ROW):
    self.main_grid.grid_columnconfigure(i, weight=1)
    self.main_grid.grid_rowconfigure(i, weight=1)

# place it on screen
self.main_grid.place(
    x=controller.GAME_WIDTH // 2 - self.main_grid.winfo_reqwidth() * 0.5,
    y=0)

# generate the grid values (matrix)
self.main_grid_values = [
    [0]*self.TILES_PER_ROW for _ in range(self.TILES_PER_ROW)
]
```

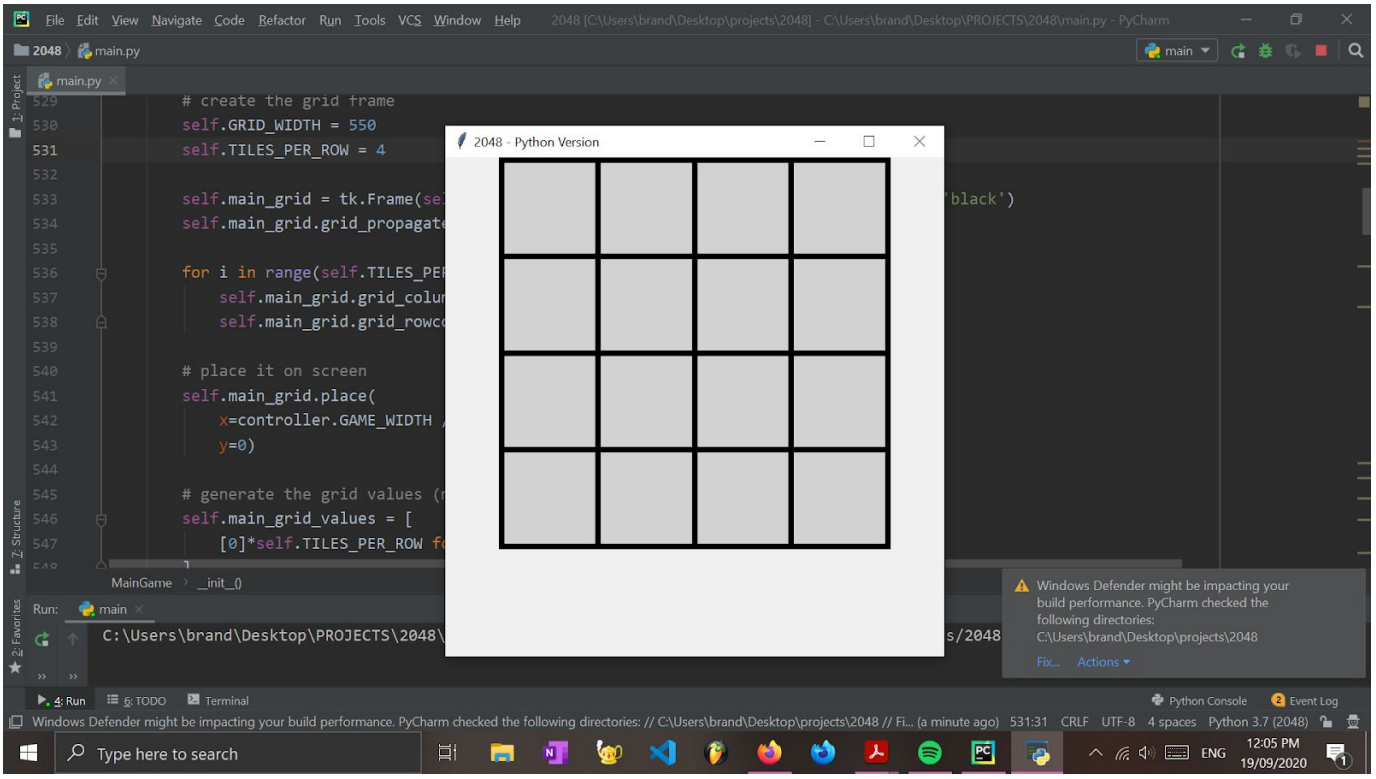
I then created a method update_grid, that uses the main_grid_values and renders them onto the screen. It clears the current frames tiles first though, to prevent a memory leak.

```
def update_grid(self):
    # clear widgets first
    for widget in self.main_grid.winfo_children():
        widget.destroy()

    BORDER_WIDTH = 8
    for i in range(len(self.main_grid_values)):
        for j in range(len(self.main_grid_values[i])):
            tile = tk.Label(self.main_grid, bg='red', text=self.main_grid_values[i][j])

            if j == len(self.main_grid_values) - 1 and i == len(self.main_grid_values) - 1:
                tile.grid(row=i, column=j, padx=BORDER_WIDTH, pady=BORDER_WIDTH, sticky='news')

            elif j == len(self.main_grid_values) - 1:
                tile.grid(row=i, column=j, padx=BORDER_WIDTH, pady=(BORDER_WIDTH, 0), sticky='news')

            elif i == len(self.main_grid_values) - 1:
                tile.grid(row=i, column=j, padx=(BORDER_WIDTH, 0), pady=BORDER_WIDTH, sticky='news')

            else:
                tile.grid(row=i, column=j, padx=(BORDER_WIDTH, 0), pady=(BORDER_WIDTH, 0), sticky='news')
```

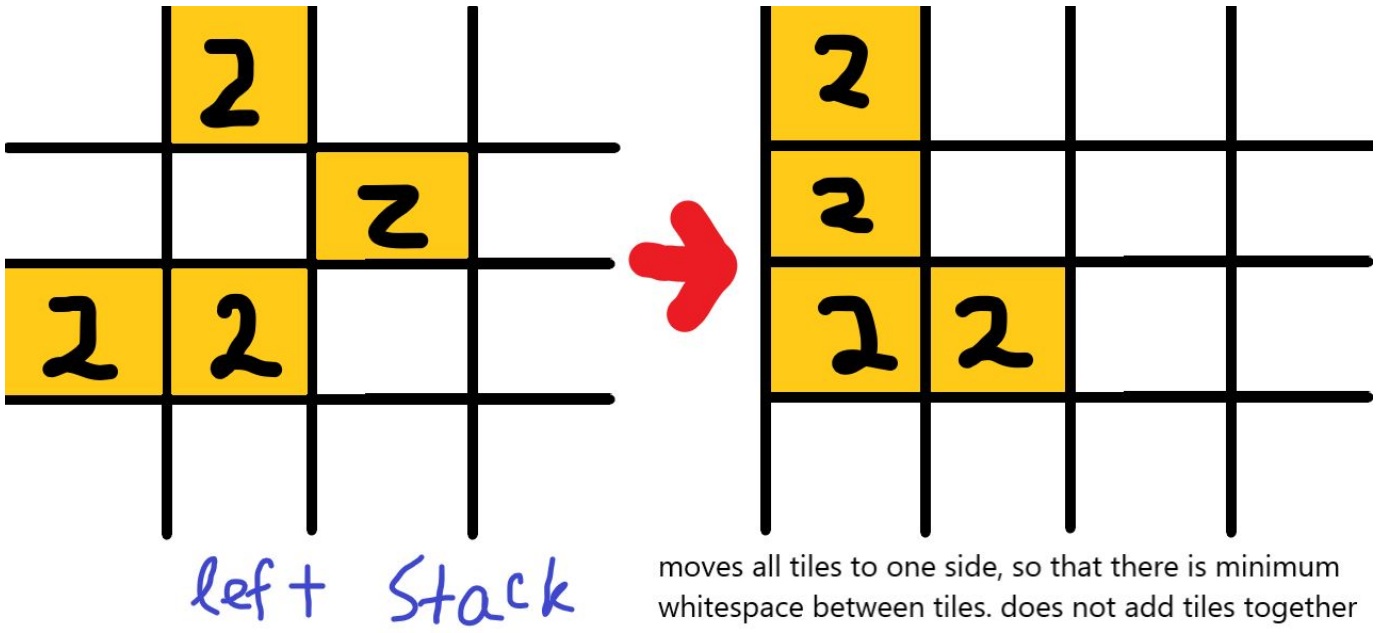
The resulting grid is as follows.



In the actual game however, I will not be showing the 0's on the board. Hence, I added an if statement that omits adding the number into the tile if it is 0 and altered the color to grey (as red is too obnoxious)



I then began writing the stack algorithm (which moves all elements to one side of the board), so that there are no gaps (0 tiles) in between adjacent non-zero tiles. To illustrate this concept, I have drawn an instance of stack left below.



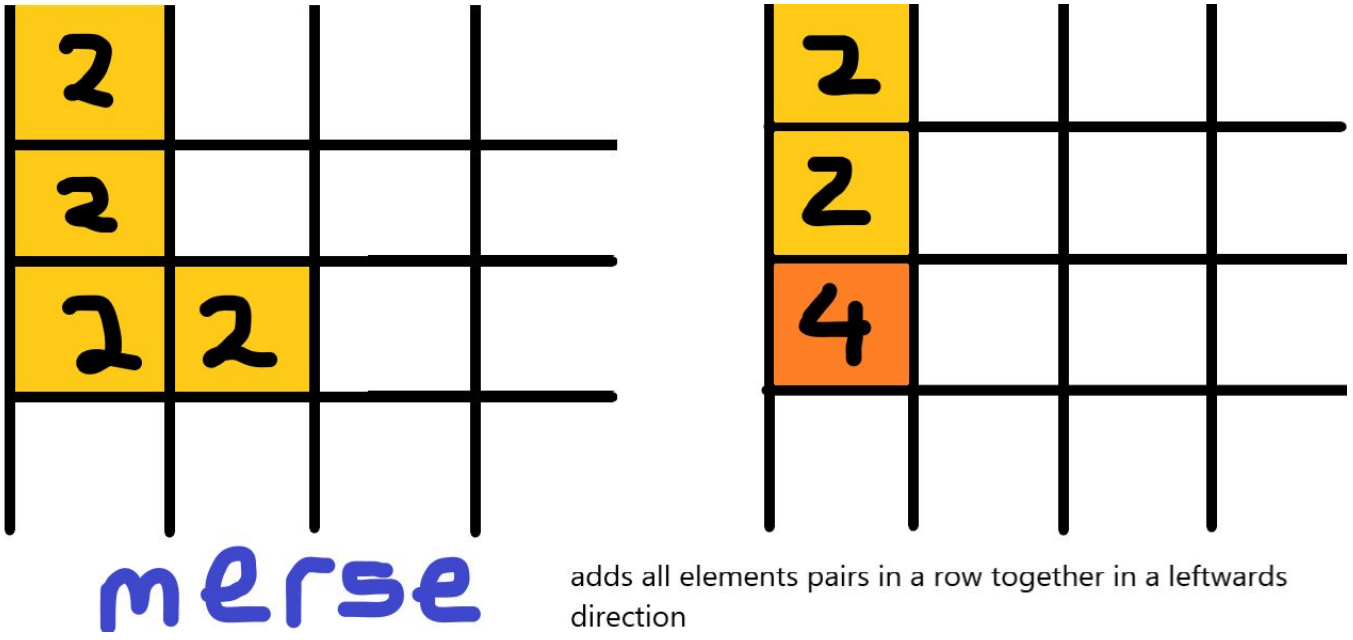
Firstly, the stack function only ever moves all the tiles to the left side (not the other 3 sides, and this will become apparent soon). Secondly, the stack function operates as following (in pseudocode):

```
create a temporaryMatrix (same dimensions as actualMatrix)
for each row in temporaryMatrix:
    create counter emptySpotLocation and initialise to 0
    for each element in same row in actualMatrix
        if element is not 0:
            temporaryMatrix[row][emptySpotLocation] = element
            empty spot location += 1

actualMatrix = temporaryMatrix
```

This works, as it only packs the elements into the temporaryMatrix in optimal order, and removes any possibility of whitespace. Actual code used.

```
def stack(self):
    """minimises 0 tile whitespace on the horizontal X axis in the negative direction (west)"""
    # create temporary matrix
    temp_matrix = [[0] * self.TILES_PER_ROW for _ in range(self.TILES_PER_ROW)]

    # for each row in actual matrix
    for i in range(len(self.main_grid_values)):
        empty_spot = 0
        # for each value in actual matrix
        for j in range(len(self.main_grid_values)):
            # if non zero, copy into optimal position into temporary matrix
            if self.main_grid_values[i][j] != 0:
                temp_matrix[i][empty_spot] = self.main_grid_values[i][j]
                empty_spot += 1

    # copy temp matrix to actual
    self.main_grid_values = temp_matrix
```

Then I wrote the Merge algorithm, which given an optimally stacked matrix in the left direction of the horizontal axis, will merge all tiles that are next to each other and return a matrix with the minimal number of non zero tiles. The reason I have only written this algorithm for the left horizontal direction will become clear soon. To illustrate this algorithm, I will demonstrate it with a continuation of the previous picture.



Here is the pseudocode for the merge algorithm:

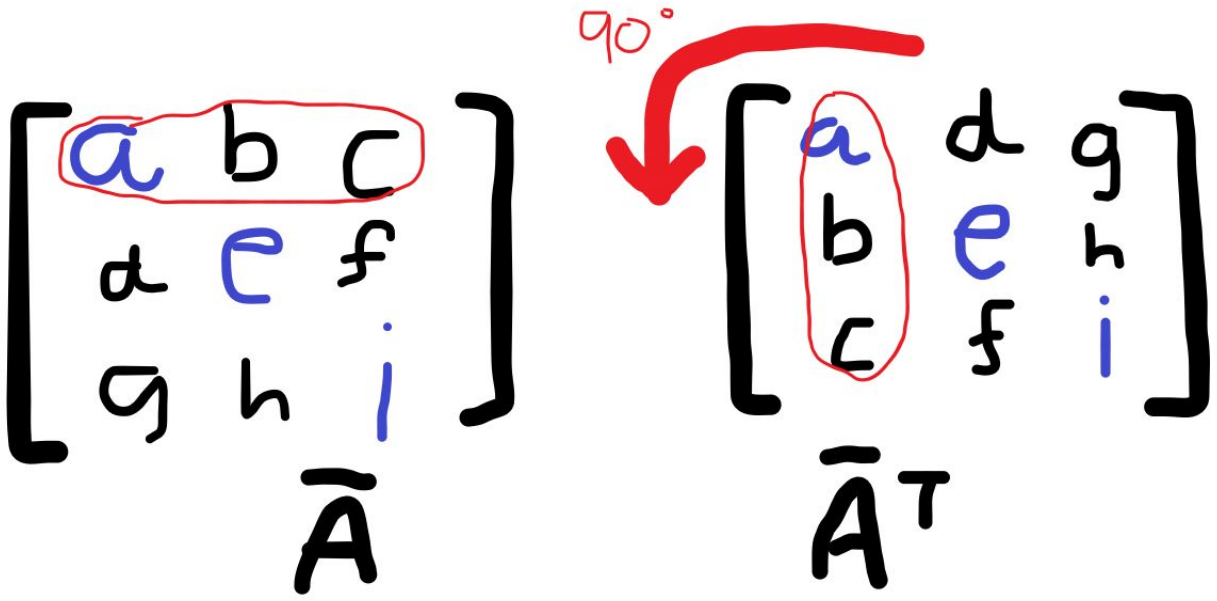
```
for each rowIndex in range(length of actualMatrix)
  for each columnIndex in range(length of actualMatrix - 1)
    if actualMatrix[columnIndex] == actualMatrix[columnIndex + 1]:
      actualMatrix[columnIndex] *= 2
      actualMatrix[columnIndex + 1] = 0
```

Actual code.

```
def merge(self):
    for i in range(len(self.main_grid_values)):
        for j in range(len(self.main_grid_values) - 1):
            if self.main_grid_values[i][j] == self.main_grid_values[i][j+1]:
                self.main_grid_values[i][j] *= 2
                self.main_grid_values[i][j+1] = 0
```

Now here is why I haven't written any of the code for all the other sides of the matrix. A matrix can be manipulated quite easily with matrix operations. What's more interesting is these operations can be reversed, so that the matrix is back into its original state. The two matrix operations I will use are TRANSPOSE and REVERSE.

TRANSPOSITION: The matrix is mirrored on the left-top to bottom-right diagonal. That is, for every element in the matrix, its coordinates are flipped in the new matrix. Here's an illustration:



As you can notice, the transpose of a matrix A, A^T, is actually just A but rotated 90 degrees counterclockwise. This means if I was to TRANSPOSE the game matrix, STACK the matrix, MERGE the matrix, and as $(A^T)^T = A$, TRANSPOSE the matrix again to restore original orientation — I would have just performed the STACK and MERGE operations on in the UPWARDS direction.

Here is the code

```
def transpose(self):
    temp_matrix = [[0] * self.TILES_PER_ROW for _ in range(self.TILES_PER_ROW)]
    for i in range(len(self.main_grid_values)):
        for j in range(len(self.main_grid_values)):
            temp_matrix[j][i] = self.main_grid_values[i][j]

    self.main_grid_values = temp_matrix
```

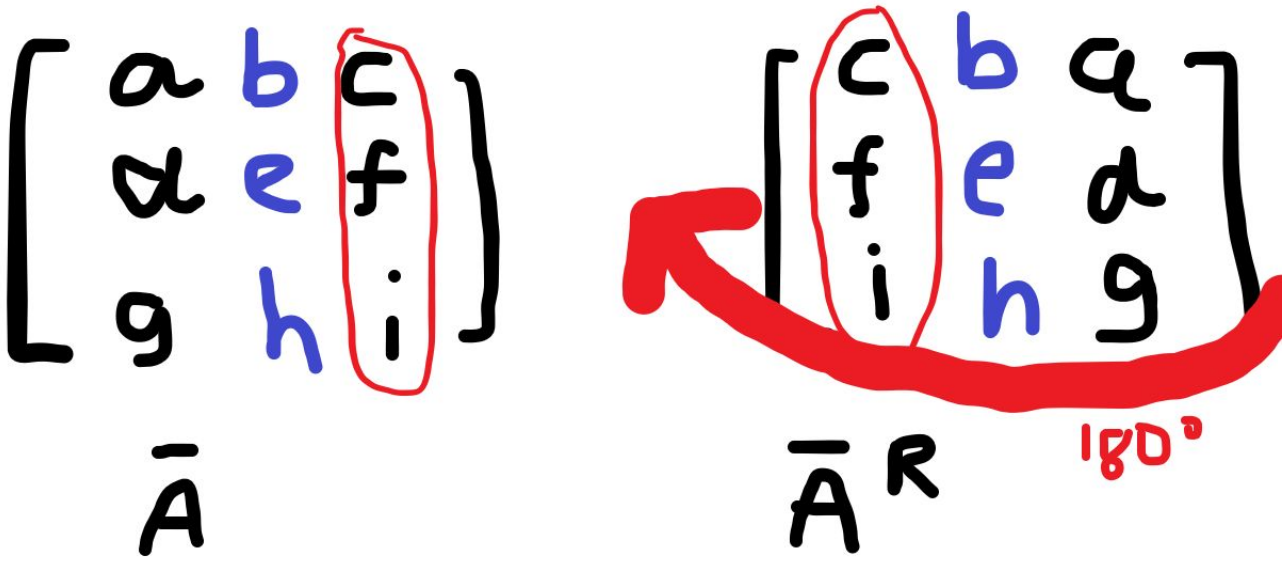
Then we have the REVERSE matrix function. Where the function is mirrored on the Y-axis. Formally, this is defined as the multiplication of matrix A by the inverse identity matrix - the exchange matrix.

Here is the pseudocode for the REVERSE function:

temporary matrix = empty matrix with same dimensions as actual matrix
for index in range(length of row in actualMatrix):
  for index2 in range(length of column in actualMatrix):
    temporaryMatrix[length of row - 1 - index][index2] = actualMatrix[index][index2]

This swapping is reversible as by mirror indexes via index: lengthOfRow - index - 1 —> by performing this operation again —> lengthRow - index - 1: lengthOfRow - (lengthOfRow - index - 1) - 1: lengthOfRow - lengthOfRow + index + 1 - 1: index.

Here is the graphical representation of such an algorithm.



Here's the original code:

```
def reverse(self):
    temp_matrix = [[0] * self.TILES_PER_ROW for _ in range(self.TILES_PER_ROW)]
    for i in range(len(self.main_grid_values)):
        for j in range(len(self.main_grid_values)):
            temp_matrix[i][self.TILES_PER_ROW - 1 - j] = self.main_grid_values[i][j]

    self.main_grid_values = temp_matrix
```

This means if I was to REVERSE the array and STACK it and MERGE it and the REVERSE it once more, to correct the orientation, I would have effectively STACKED and MERGED the right hand side of the original matrix.

This also means if I was to TRANSPOSE the array (putting the North side on the Left side), and the REVERSE IT (Putting the side on the right (south) in the left), STACK and MERGE the array and then undo the REVERSE (by reversing once more) and TRANSPOSING once more to reach original orientation, I would have effectively performed a STACK and MERGE on the south side of the original matrix.

Then I made the adding two methods. As noted, the matrix spawns a randomly placed two after every move and also at the start of the game. This is just a randomly generated coordinate pair (i, j) that is regenerated if it lands on a non-zero pair.

```
def add_two(self):
    i = random.randint(0, self.TILES_PER_ROW - 1)
    j = random.randint(0, self.TILES_PER_ROW - 1)

    while self.main_grid_values[i][j] != 0:
        i = random.randint(0, self.TILES_PER_ROW - 1)
        j = random.randint(0, self.TILES_PER_ROW - 1)

    self.main_grid_values[i][j] = 2
```

However, if you have looked at the way the twos are added, it becomes quickly clear there is a problem: the while loop. The while loop runs forever if there is no possible place to add a two. To mitigate this problem, I have created another method — any_empty_tiles — to determine if this is true. This method simply returns True if there is a 0 in the matrix anywhere, and False otherwise.

```
def any_empty_tiles(self):
    for i in range(self.TILES_PER_ROW):
        for j in range(self.TILES_PER_ROW):
            if self.main_grid_values[i][j] == 0:
                return True

    return False
```

Therefore, adding a two to the board only occurs if there are empty tiles on the board.

Finally, I have written two methods which determine if there are any possible moves on the board at any given time. This checks the X axis for any tiles which are adjacent , the Y axis for any tiles that are adjacent and the presence of empty tiles. If all 3 criteria are NOT present, and only when, the game concludes and the controls are unbound and the game over sign shows.

```
def any_possible_moves_horizontal(self):
    for i in range(self.TILES_PER_ROW):
        for j in range(self.TILES_PER_ROW - 1):
            if self.main_grid_values[i][j] == self.main_grid_values[i][j+1]:
                return True

    return False

def any_possible_moves_vertical(self):
    for i in range(self.TILES_PER_ROW - 1):
        for j in range(self.TILES_PER_ROW):
            if self.main_grid_values[i][j] == self.main_grid_values[i+1][j]:
                return True

    return False
```

Finally, I have added the score functionality and the restart functionality.

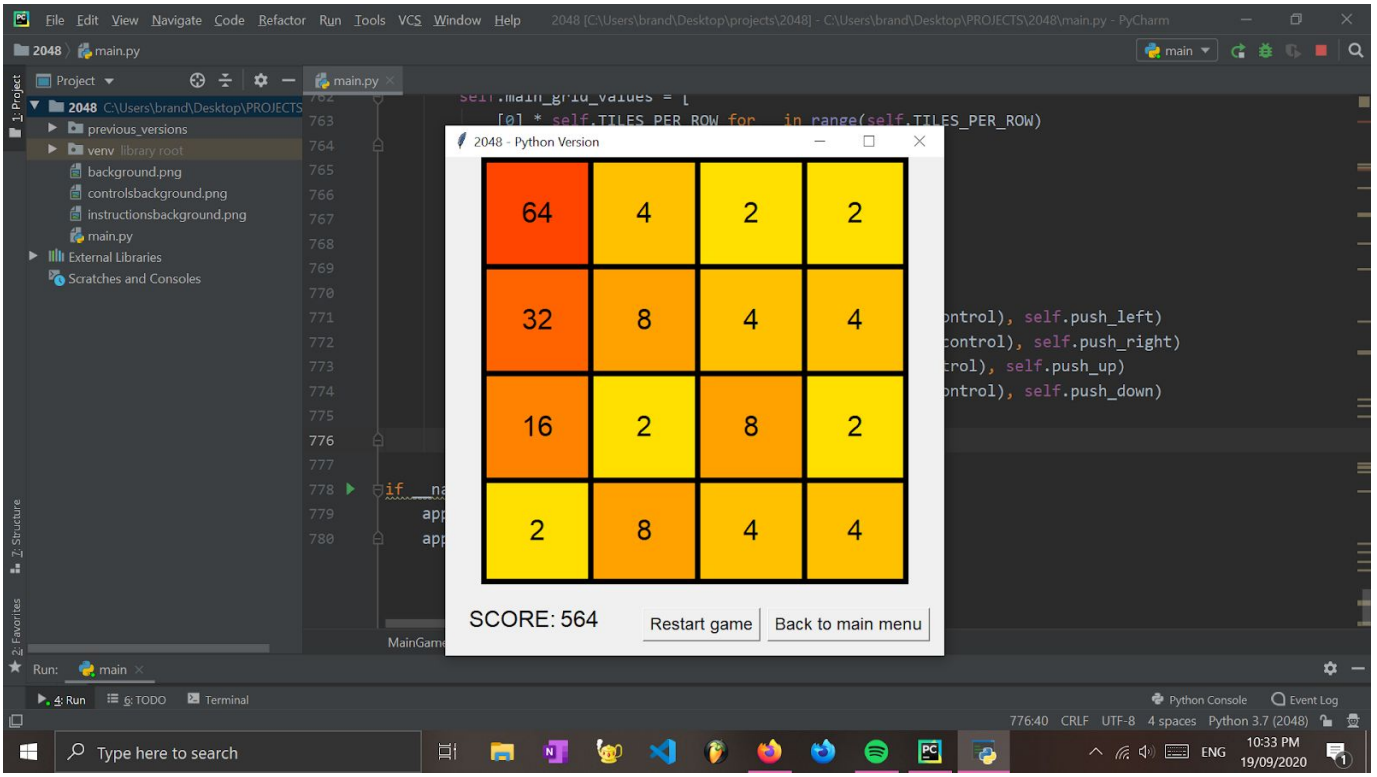
```
self.score_value = tk.StringVar(value='0')
self.score_value_label = tk.Label(self, textvariable=self.score_value, font=controller.BUTTON_FONT)
```

The score variable will be represented by a stringVar. The scoring system is as follows: every time a merge operation takes place, the score is incremented by the sum of the two tiles. i.e if 2 and 2 merge, the score goes up by 4.

The restart method resets the matrix, re-renders the game board, resets the score and removes the Game over label if this is present.

Here is the finished product of version 2: the main game page.

https://gyazo.com/ed8209c01bb0aa8eb57bd1459bb76f83



Version 3: Save functionality

To get the save working, I need to save two things: the score of the user and the board position. These are stored in self.score_value and self.main_grid_values. This is an integer value (fetched) and a 2D list, respectively. As these are JSON serialisable data types — I have stored the game save file in a JSON file.

```
def save_game(self):
    with open('2048save.json', 'w+') as f:
        results = {
            'game_matrix': self.main_grid_values,
            'score': self.score_value.get()
        }
        json.dump(results, f)

    self.controller.show_frame(MainMenu)
```

And this is the load game function, which reads from the json file and stores these back in the Frame variables self.score_values and self.main_gridvalues

And I've modified the show_frame function to take an optional parameter, new_game, which determines whether or not the game is loaded or not when calling show_frame(MainGame)

Version 4: Enter your name and predict your score

In order to record the name and predict the score, the user will do this before the game starts. Therefore, I will have it so that a frame is drawn over the game grid, prompting the user for their details before the game starts. On a **valid** submission - the controls will be binded (so that the user can only start playing after the game starts) and the details prompt frame destroyed (so the game can be seen).

Here's the code. Firstly, I pre declare the name and prediction variables.'

```
# pre declare the name and prediction value
self.name = ''
self.predicted_value = -1
```

Then, after I LOAD the game (if there is one), I begin checking if there is any name and score that needs to be entered. This is done so that the user doesn't have to enter their name and prediction score if they are loading from a game. If not (either one), then I create the details frame.

```
# load game if needed
if not new_game:
    self.load_game()

# ask for name and prediction
if not self.name or self.predicted_value == -1:
    # details frame create
    self.details_grid = tk.Frame(
        self,
        width=485,
        height=340,
        bg='white',
        highlightbackground="#bbbbbb",
        highlightthickness=8,
        padx=40,
        pady=40
    )
    self.details_grid.grid_propagate(0)
```

I then grid it with components (the entries, labels and submit button) and then place it on the screen. This is the final product.

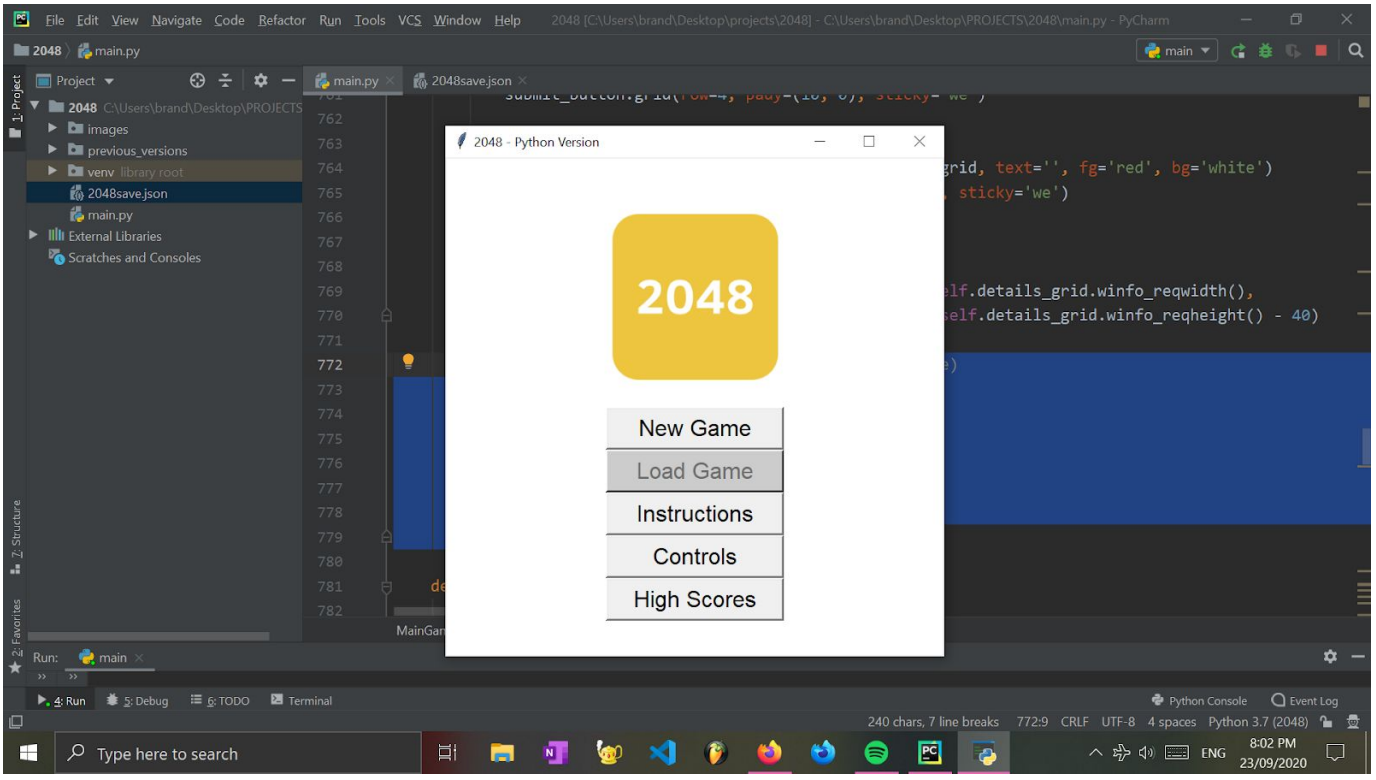
On submit (hitting the play game button), the inputs are checked. The name must be below 50 characters (to occupy one line) and the score field must be numeric and (below 1 trillion and above or equal to 0).

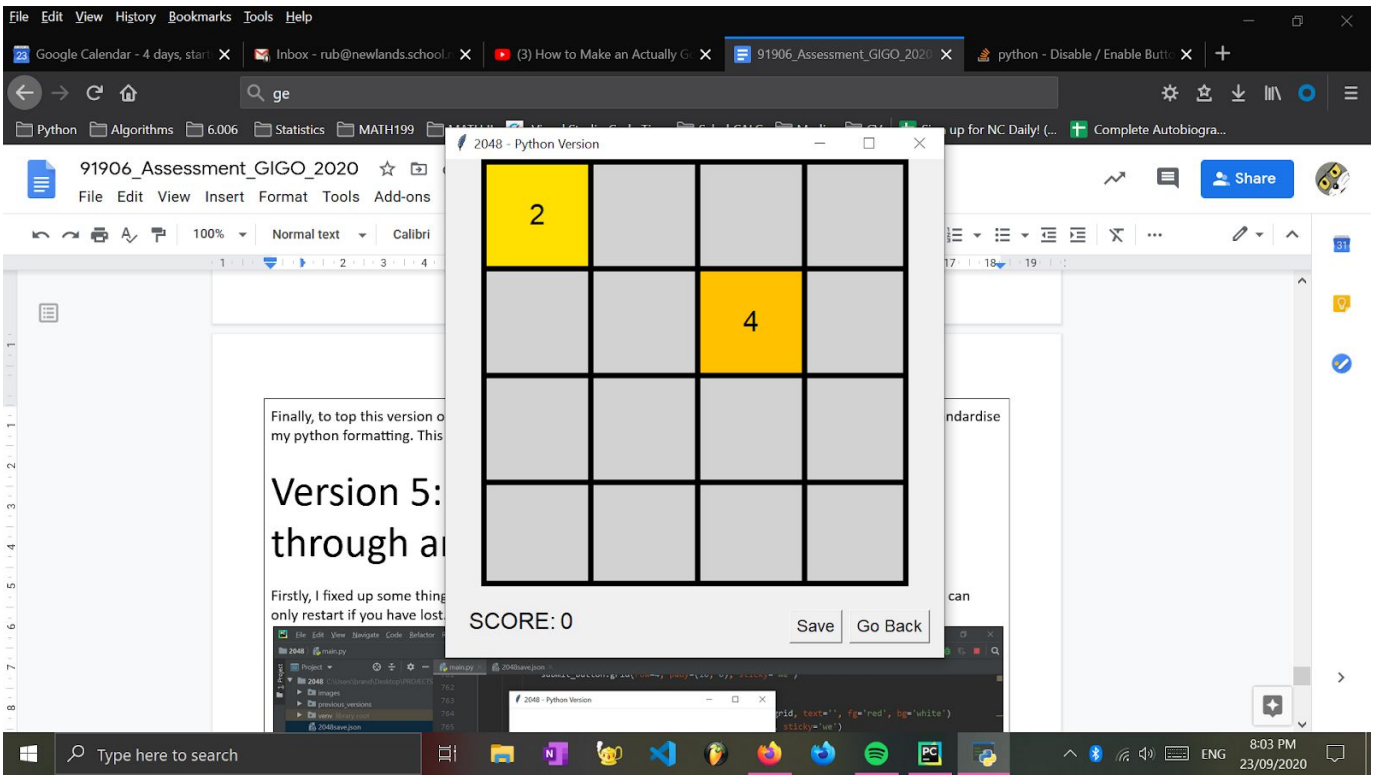
On game finish, the game over frame now displays extra information regarding what the user entered.

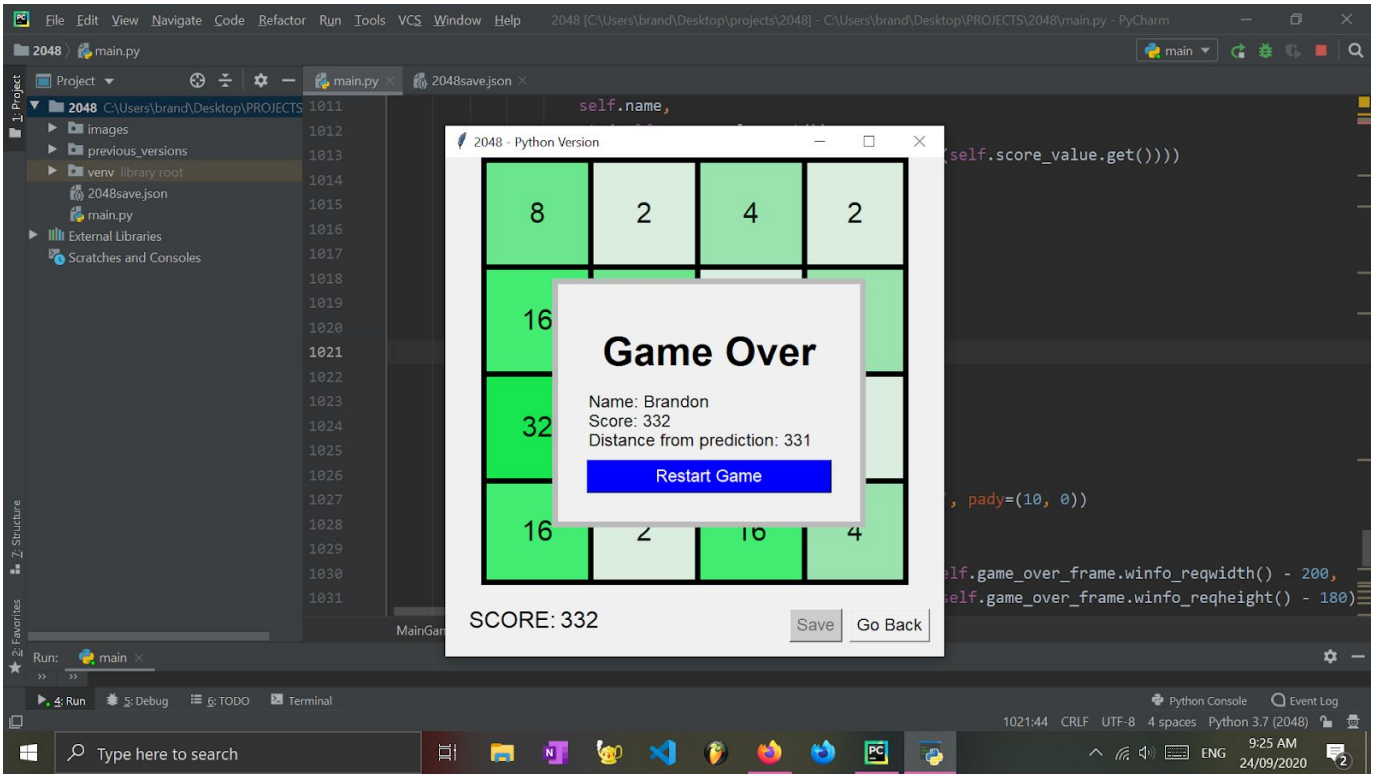
Finally, to top this version off, I have made edits to the code formatting to align with PEP8 standards, to standardise my python formatting. This is available in the IDE I am using natively.

Version 5: standardisation of coloring and fixing of game logic

Firstly, I fixed up some things first regarding the logic of the game. You could no longer save if you lost. You can only restart if you have lost. You can now save and go back or not save and go back.







I then normalised all of the colors that the tiles had. This was done by creating a dictionary of values for the font color and tile color that had a key of the absolute value on the tile.

```
TILE_COLORS = {2: "#daeddf", 4: "#9ae3ae", 8: "#6ce68d", 16: "#42ed71",
               32: "#17e650", 64: "#17c246", 128: "#149938",
               256: "#107d2e", 512: "#0e6325", 1024: "#0b4a1c",
               2048: "#031f0a", 4096: "#000000", 8192: "#000000",}

LABEL_COLORS = {2: "#011c08", 4: "#011c08", 8: "#011c08", 16: "#011c08",
               32: "#011c08", 64: "#f2f2f0", 128: "#f2f2f0",
               256: "#f2f2f0", 512: "#f2f2f0", 1024: "#f2f2f0",
               2048: "#f2f2f0", 4096: "#f2f2f0", 8192: "#f2f2f0",}
```

And when the program rendered, I accessed the color I needed by accessing it via key from a dictionary.

```
tile = tk.Label(
    self.main_grid,
    bg=TILE_COLORS[int(self.main_grid_values[i][j])],
    fg=LABEL_COLORS[int(self.main_grid_values[i][j])],
    text=self.main_grid_values[i][j],
    font=('Arial', 20)
)
```

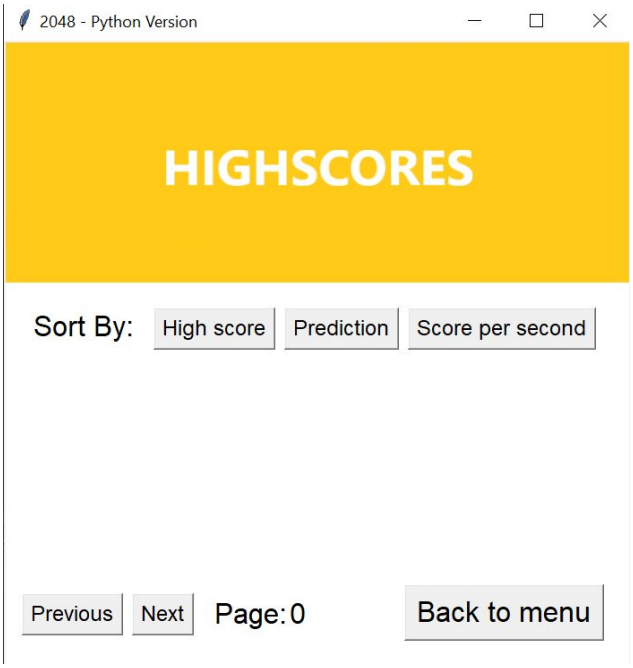
The start of these colors can be previewed in the image above.

The bug in the game mechanic — spawning of twos on moves that did nothing — was fixed by checking the matrix for changes on every move.

```
pre_matrix = self.main_grid_values

"""Control to swipe tiles down"""
self.transpose()
self.reverse()
self.stack()
self.merge()
self.reverse()
self.transpose()

if self.any_empty_tiles() and pre_matrix != self.main_grid_values:
    self.add_two()
```

Version 6: highscore table

To create the high score page, I created a frame canvas, and drew up the high score image on the canvas. I then added buttons (that don't have actions yet) along the top of the page for sorting the high scores by certain criteria.
Furthermore, I added buttons on the right of the page that allowed for next and forwards. Finally, there was a back button. I plan to add a table in the middle of the page.

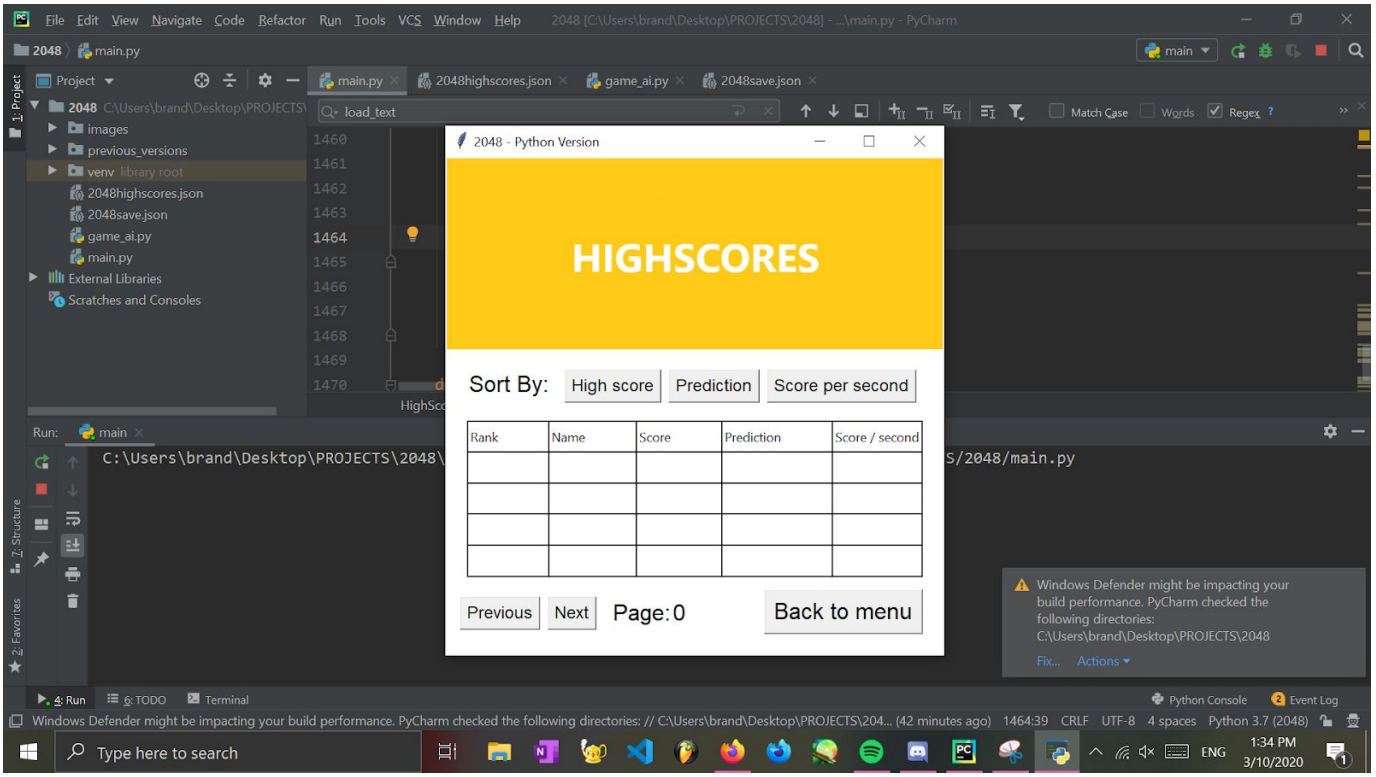


I then made a table into the middle of the page — into the blank space I had allocated for it. It would be 4x5 (rows and columns). Each row entry would be an entry for each player. Each column will show a certain stat for each player.
The columns will be: Rank, Name, Score, Prediction, Score per second.

The initial row will be a header — displaying the column details (what this column shows).

The code for the table was reused from the controls section, as this can be easily modified by changing the arguments of the for loop.

Here is the initial table.



To store the values of the user highscores, I have also used a JSON file to save entries. The json file will contain a LIST of OBJECTS where each object is a user entry. Each object will have attributes of name, score, predicted score, and score / s. The rank can be calculated within the program — as this is subjective. For example, here are some test data entries.

```
[
  {"name": "a", "score": 52, "prediction": -281, "score / s": 0},
  {"name": "a", "score": 4, "prediction": 2, "score / s": 0},
  {"name": "brandon", "score": 60, "prediction": -40, "score / s": 0},
```

I then wrote the load_textmatrix method — which loads the text entries from a JSON file if it exist, and places it into the text_matrix attribute for this frame.

```
def load_textmatrix(self):
    # load the textmatrix up
    try:
        f = open('2048highscores.json', 'r')
        data = json.load(f)

        for index, entry in enumerate(data):
            self.text_matrix.append([
                index + 1,
                entry['name'],
                entry['score'],
                entry['prediction'],
                entry['score / s']
            ])

    except Exception:
        pass
```

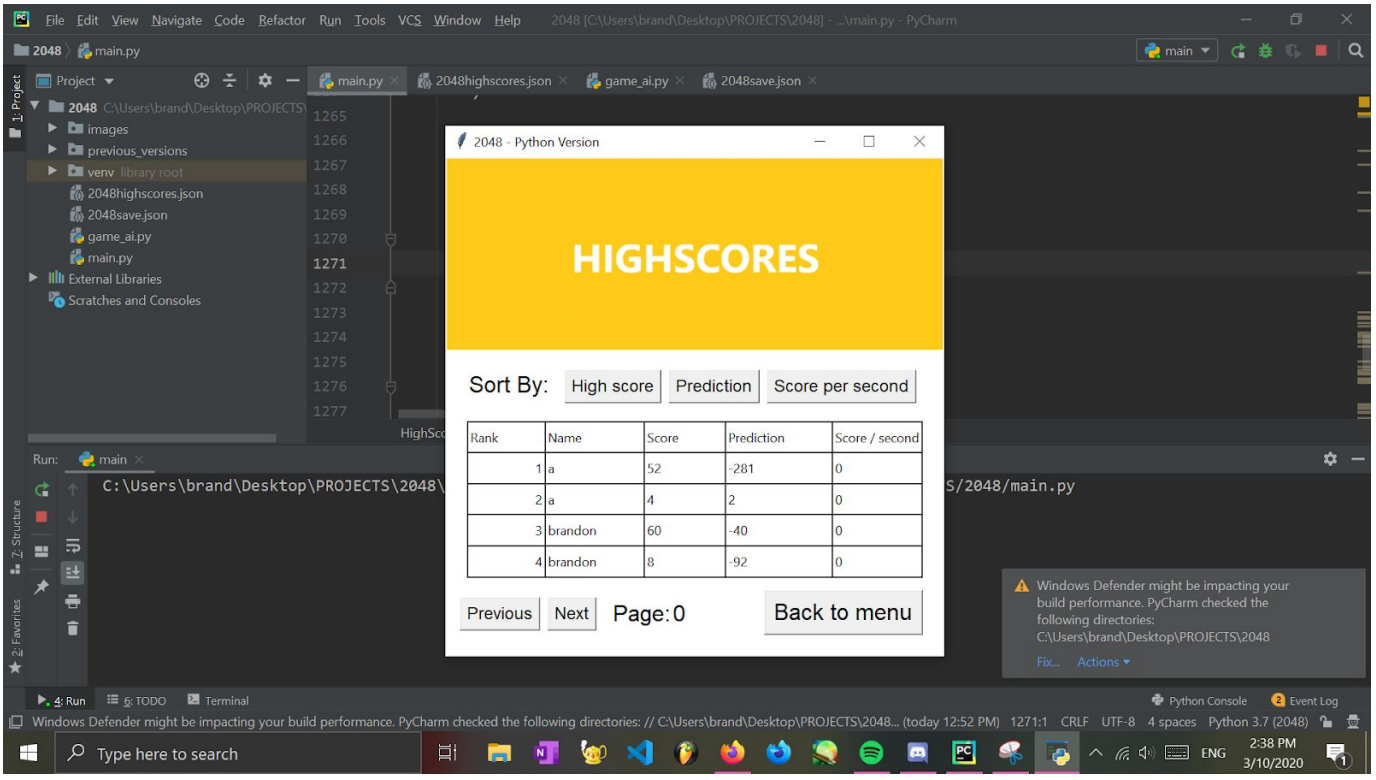
For each entry in the JSON file, I append it as a list into the matrix.

I then wrote the update_scoreboard method — which draws up the table and packs it with all the elements of the text_matrix.

Firstly, I check the text_matrix so that there is at least 4 elements within it. If not, I append blank entries so that this condition is met.

```
# check textmatrix for emptiness
if len(self.text_matrix) < 5:
    J = 5 - len(self.text_matrix)
    for iteration in range(J):
        self.text_matrix.append(['' for i in range(5)])
```

I then grid into 0th row the heading labels for the high score table. This is also reusing code from the controls table section. Finally, I loop through the entries of the text matrix in the following rows, creating the remainder of the table. For each element in the text_matrix, in order of appearance, I assign them an incrementing rank. Here is the finished result.



To create the sorting mechanism — because of the architecture of the system — all I needed to do was sort self.text_matrix and then redraw the page.

update_textmatrix_score (sort by score) method.

```
def update_textmatrix_score(self):
    data = []
    for line in self.text_matrix:
        if type(line[0]) != type(0):
            continue
        else:
            data.append(line)

    data.sort(reverse=True, key=lambda x: int(x[2]))
    self.text_matrix = []
    for line in data:
        self.text_matrix.append(line)


    self.update_scoreboard()
```

Firstly, I created a temporary array, data, and appended all non-blank entries into this array. I then sorted this array in descending order based on it's second index (the score index), and then copied this information back over to text_matrix — and redrew the page.

The update_textmatrix_prediction (sort by prediction) method followed a similar suit, however, because of the nature of prediction — I wanted the closest prediction, and

therefore had to sort in ascending order by the absolute value of the prediction.

```
data.sort(key=lambda x: abs(int(x[3])))
```

The sort by score/second was simple, and followed the same method as the sort by score.

Allowing for different pages to be displayed.

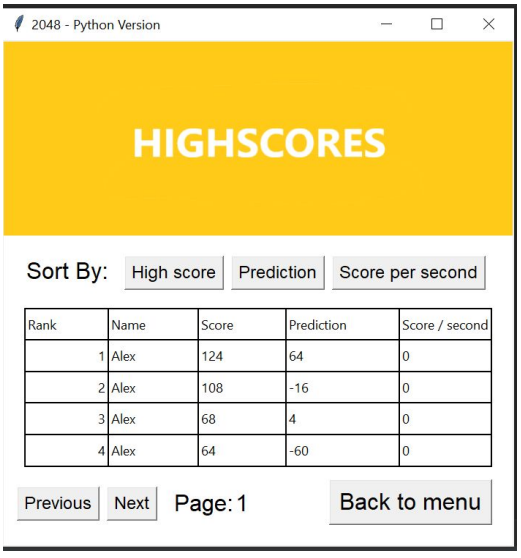
To achieve this, I let the method that drew the score table — update_scoreboard use 2 object attributes, self. and self.b. A was the start pointer and B was the finishing pointer of the matrix. By default, A =0 and B = 4. Therefore, by looping between these indexes of the text_matrix, I am able to render only these entries of the text matrix. Then, clearly, the next page method will increment A and B both by +4. To prevent overflow, if B is already at a length that is equal to the number of entries in text_matrix — then there obviously are no more entries, and the function returns. Otherwise, even if B+4 overflows, I must display the table, as there are still entries available and the overflowed entries will be displayed as blanks.

The previous page method follows an identical suit, except the overflow boundary is 0 and instead of incrementing by 4, we decrement by 4.

In both methods, the page_number StrVar is incremented/decremented by 1. Here is the code for the next_page method.

```
def next_page(self):
    if self.b >= len(self.text_matrix):
        return

    self.page_number.set(str(int(self.page_number.get()) + 1))
    self.a += 4
    self.b += 4
    self.update_scoreboard()
```

These methods were then subsequently bounded to their corresponding buttons, and this resulted in the final product of the 6th version.



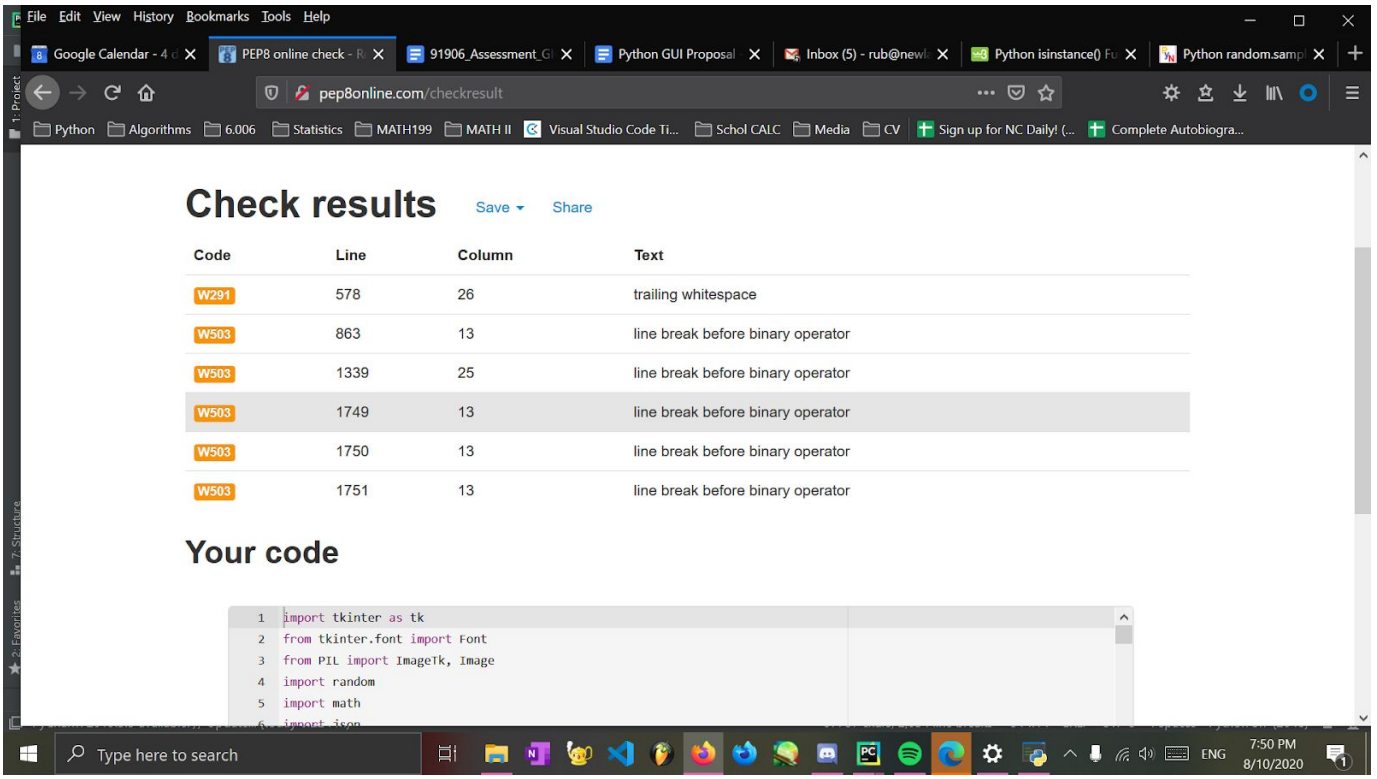
Version 7: PEP8 validation, commenting and final touches

Validation and commenting

The code has been fully commented, so that sections are easy to understand. The logic flow of the program has been refined so it is fully object oriented, and follows an easy-to-understand flow.

All errors according to the PEP 8 python standardisation documentation has been corrected, and the code is PEP8 compliant.

I have moved the instruction strings to a separate document so they are more easily changed - and also so the python code abides by PEP8.

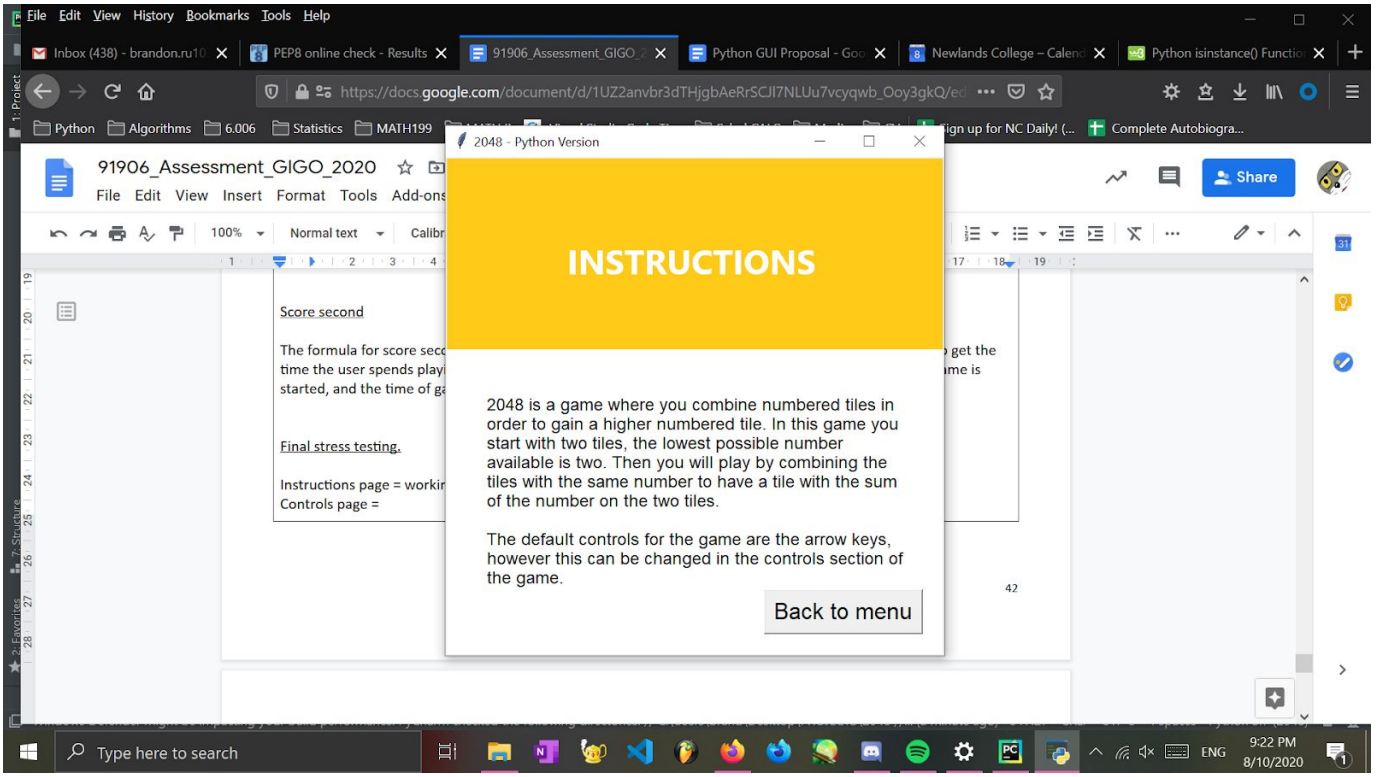


Score second

The formula for score second is (total score / time between start and finish of game). Therefore, I need to get the time the user spends playing. This was done by having an initial variable that stores the time when the game is started, and the time of game end is therefore the current time at that instant - initial_time_variable.

Final stress testing.

Instructions page = working as intended, and no english mistakes present.



Controls page = working as intended, no identifiable bugs and error checking is fully robust. Here is a short gif showing it working - catching errors.
https://gyazo.com/c1745f4b37418833cf9e0340c40f0515

Highscores page:

The sorting mechanism works as intended, no bugs found
https://gyazo.com/61e928b69d7959b515e40c71e804ee6d

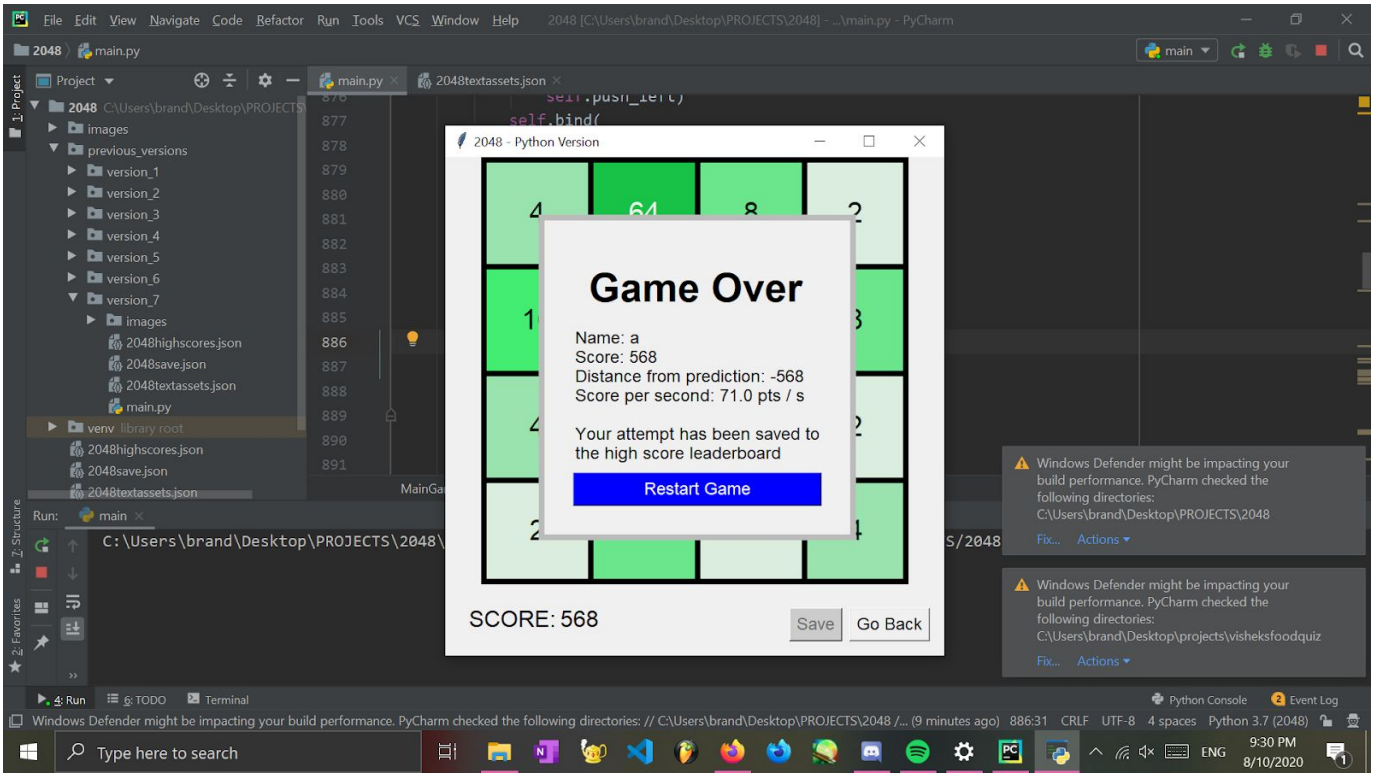
Page switching works as intended. https://gyazo.com/88513c61d115646807d3c8bddde68ed3
Minor bug found that appears when the game has no high score data. IT allows the user to go to the second page even though there is no data on the first. This was fixed by adding a specific case if statement that disallows this

```
if self.b >= len(self.text_matrix) or len(self.text_matrix) == 5:
    return
```

Main game
Game prompt works as intended: https://gyazo.com/416aed1550344605bf1859706447580c, all errors caught (apart from not being able to render unicode egyptian hieroglyphics)

Game works perfectly, with matrix operations and no bugs have been found.

Ending message is fine: displaying all content, on the page in a symmetrical manner



Game save is working fine too
https://gyazo.com/fc4dc40148447b75c0c603a245e2fc42

As the whole game is working fine, and peer testing has been extensively done I can conclude that the development of the game has been finished.

Development

As you develop your program, show how you have tested out each version before moving on to the next step:

Version 1 - Program Skeleton, Main Menu, Instructions and Controls. The main skeleton of the GUI is complete, and page switching works. Main Menu is working. Controls are able to be changed and altered. Instructions page is accurate.	
Testing carried out Peer testing was carried out.	**Next steps**

Alec De Leon did not manage to break the Instructions and main menu pages. He broke the controls page by pressing space when changing keys. This is because pressing space ultimately resulted in the button being toggled again whilst setting it to Space — resulting in an infinite loop.	Program development is going well. Continue to work on the project as planned, working on the actual gameplay of the application as follows. Fix any bugs identified through testing (space bug)

Version 2 - Main Game Space bug fixed. This was fixed by passing the space event to a tank function, which did nothing — meaning this problem was no longer present. First draft of gameplay was working — and the 2048 game was playable.	
Testing carried out Peer testing as carried out. Alec De Leon Game works, however is not working as intended when he holds one key down. This results in the game becoming ridiculously easy, as only 2's are spawning and thus the board will optimally merge onto one side.	**Next steps** The game is progressing well. In terms of timing, the project is on track for completion. Continue to develop as the plan says — developing the save functionality.

Version 3 - Save Functionality The one key pressing was not fixed completely, but the current solution was to spawn 2's and 4's randomly, with a 3:1 ratio. This prevents the user from doing it, as it ruins their gameplay as it is no longer optimal. Save functionality is now working, and the game is saved every time the user exits	
Testing carried out Peer testing was carried out. One critical bug was identified by Alec — the game is still saved when the game is over. This was fixed by clearing the save file when the game is over — and adding an option that greys out the load game option if the file is empty.	**Next steps** Fix any bug identified — save bug and improve the two bug, and continue to follow project plan

Version 4 - Prediction and Name Prompts	
Testing carried out	**Next steps**

Peer testing was carried out. Alex (brother): Identified that if the name entered was too long, it didn't wrap around and the prompt ran off the page. This bug was quickly fixed by adding a text wrap parameter. Angus K. (alpha tester): Identified that the two tiles still spawns on an invalid move, providing me with a lead for the two bug Brandon R: The game breaks upon reaching the 256 tile. This is because of the algorithm which allows for the gradient of tiles increments the R of the RGB value of the tile. Eventually, there will be an overflow, which happens to occur at the 256 tile — resulting in a game crash. This is a critical error.	We must take a detour from the project plan to fix this critical error — the 256 bug, which is game threatening and fix the two bug too.

Version 5 - Standardised Color scheme and mending of the game logic

The algorithm which allows for the tiles to change colors has been changed to a dictionary approach. The color of each tile has been stored in a dictionary (along with its text color), and therefore, it can be accessed via key. This fixes the 256 bug, and allows players to continue playing.

I fixed the two bug by checking the game matrix final vs initial and checking if there's any difference. If none, there has been an invalid move and no 2/4 tiles are spawned.

I also changed how the player saves the game. The player now must manually save, vs autosaving, which gives the player more agency, say if they had a bad run, they can go back in time.

Testing carried out	**Next steps**
Peer testing was carried out. No bugs were found this time, and the game was working as intended.	Continue along the project path, and create the final component — the high score table.

Version 6 - High score table

The highscore table works as intended. The game, when concluded, saves to the high score table as intended. The highscore can sort entries, and navigate between pages.

Testing carried out	**Next steps**
Peer testing was carried out. No bugs were found in the version either, and the game was working as intended.	

Version 7 - PEP8 Code and Fully commented and score per second The final component of the game - the score per second is implemented and it works. The code is fully commented and adjusted so it suits PEP8. This is to ensure the code is future proofed and scalable if future changes need to be made.	
Testing carried out Peer testing was carried out. One minor bug was found in the page switching component of the program. It was quickly fixed.	**Next steps** None.

Final conclusion

The 2048 game is a simple yet addictive game. I have brought a twist to this classic game by coding it up in tkinter, a Python framework. The program I have made performs the specified task, and even adds more to the traditional experience by extending it so there are additional stats (predictions and score per second) and local highscores. I have ensured the user experience is at the highest standard — offering clear instructions, the ability for the user to change their controls and the possibility of saving.

Regarding the development of the game, it was developed iteratively and was thoroughly peer tested for errors throughout the app cycle. The game code has been fully commented and is up to standard with standard python formatting (PEP8) — future proofing the code, in case it needs to be changed again later. The code is organised in an Object Oriented manner, allowing the code to be logical, well structured and highly reusable/modifiable. Supporting documentation (this document) has included graphical explanations of the mathematics and logic behind the main program — making the program even more clearer.

Assessment Schedule/Mahere Aromatawai

Achieved	*Merit*	*Excellence*
Your program should at the very least: ☐ Perform the specified task ☐ Use two complex programming techniques (see Appendix 1) ☐ Be flexible and robust (see Appendix 1) ☐ Be coded clearly and include comments to explain the program (versioning is strongly recommended)	☐ documenting the program with appropriate variable/module names and organised comments that describe code function and behaviour. The purpose and use of the program should be clear from the GUI as on screen statements or Help function/s. ☐ following conventions for the chosen programming language	☐ documenting the program ☐ ensuring that the program is a well-structured, logical response to the task ☐ making the program flexible and robust (See Appendix 1) ☐ comprehensively testing and debugging the program (with supporting documentation). Program should not be able to be broken.

☐ Tested and debugged to ensure it works on a sample of expected cases (with supporting documentation)	(PEP). Validation is required via a PEP checking site. ☐ testing and debugging the program in an organised way to ensure that it works on a sample of both expected cases and relevant boundary cases. (with supporting documentation)	

Appendix 1

Examples of complex programming techniques include:

- programming or writing code for a graphical user interface (GUI)
- reading from, or writing to, files or other persistent storage
- object-oriented programming using class(es) and objects defined by the student
- using types defined by the student
- using third party or non-core API, library or framework
- using complex data structures (e.g. stacks, queues, trees).

Example of ways of making a program flexible and robust include:

- using actions, conditions, control structures and, methods, functions or procedures effectively
- checking input data for validity
- correctly handling expected, boundary and invalid cases
- using constants, variables and derived values in place of literals.