Odd Semester (2021)



**BINUS UNIVERSITY**

**BINUS INTERNATIONAL**

**Assignment Cover Letter**

**(Individual Work****)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | |  | |
| **Student Information**: **Surname** | | | | | **Given Names**  **Nicander** | | **Student ID Number**  **2101693113** | |
| 1. | | **Alworth Chance** |  | |
|  |  |
| **Course Code** | **: COMP6502** |  |  | | **Course Name** | | **: Introduction to Programming** | |
| **Class** | **: L1AC** |  |  | | **Name of Lecturer(s)** | | **:** 1. Bagus Kerthyayana | |
|  |  |  |  | |  | | 2. Tri Asih Budiono | |
| **Major** | **: CS** |  |  | |  | |  | |
| **Title of Assignment**  (if any) | : Snowflake Invasion | |  |  | |  | |  | |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** |  |  | |  | |  | |
| **Due Date** | **: 7-11-2017** |  |  | | **Submission Date** | | **:** | |

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

# Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

# Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

1. Nicander Alworth Chance

**“Snowflake Invasion!”**

**Name :Nicander Alworth Chance**

**ID :2101693113**

1. **Description**

**The function of this program:**

This program is a game that trains people’s reaction time, as well as to entertain people, as what a game is supposed to be. This program consists of 12 stages with each stage getting harder and harder. Thus, this program will test the player’s ability in reacting to the incoming enemy as fast as possible

**II.a. Design/Plan**

**Project’s Hierarchy Chart**

Game initializes

Play

menu

Main

Menu

Quit

menu

If no

If yes

Play again or no

**II.b. Explanation of Each function**

**Main Menu: ( *menu.py* )**

1. **Outside def main ():**

* Holds the function import pygame. This allows the module to use functions from pygame, particularly initializing the window screen

1. **Inside def main ():**

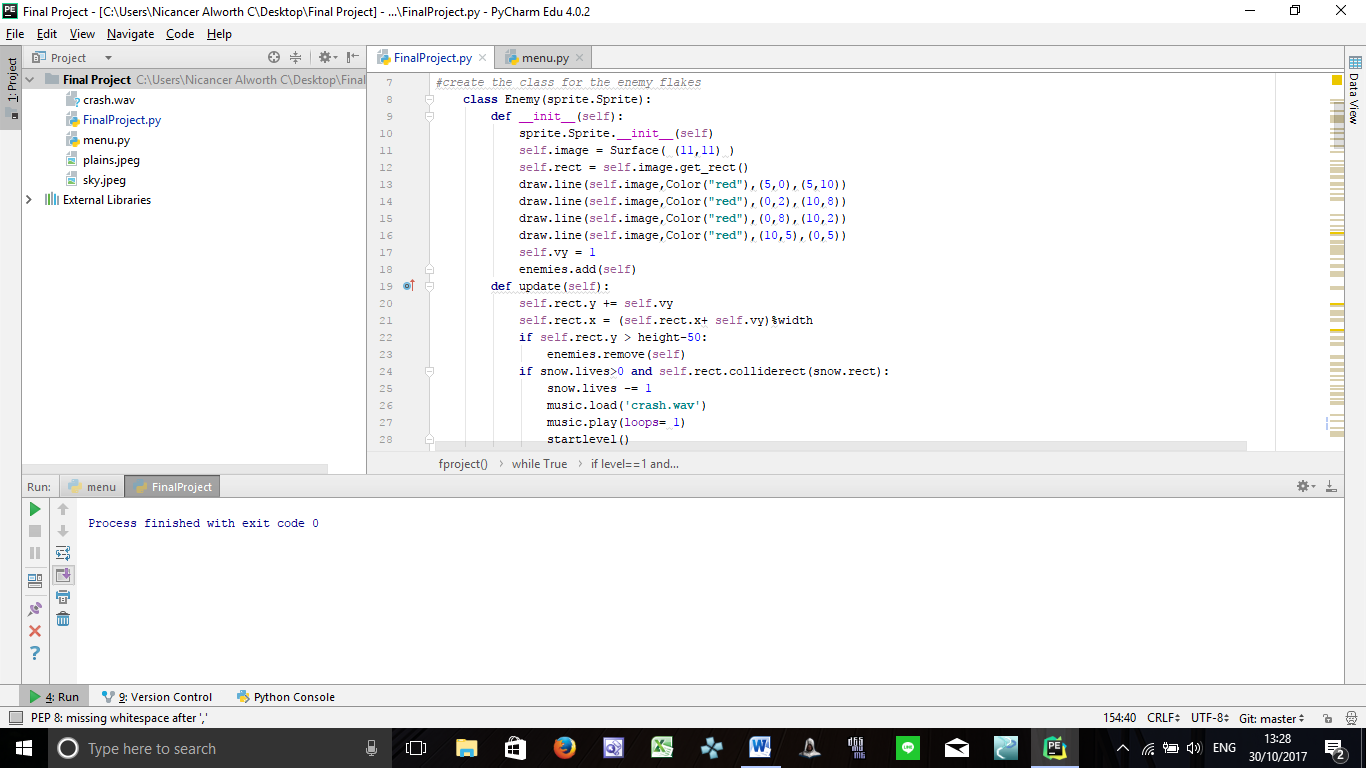
**Before the while True:**

* pygame.init() function initializes the pygame menu, which is to display the screen.
* Following that, pygame.display.set\_mode() sets the screen resolution for the game, and screen.fill() sets the background colour of the window
* Then, we set a caption for the window by using pygame.display.set\_caption(“Enter Caption here”)

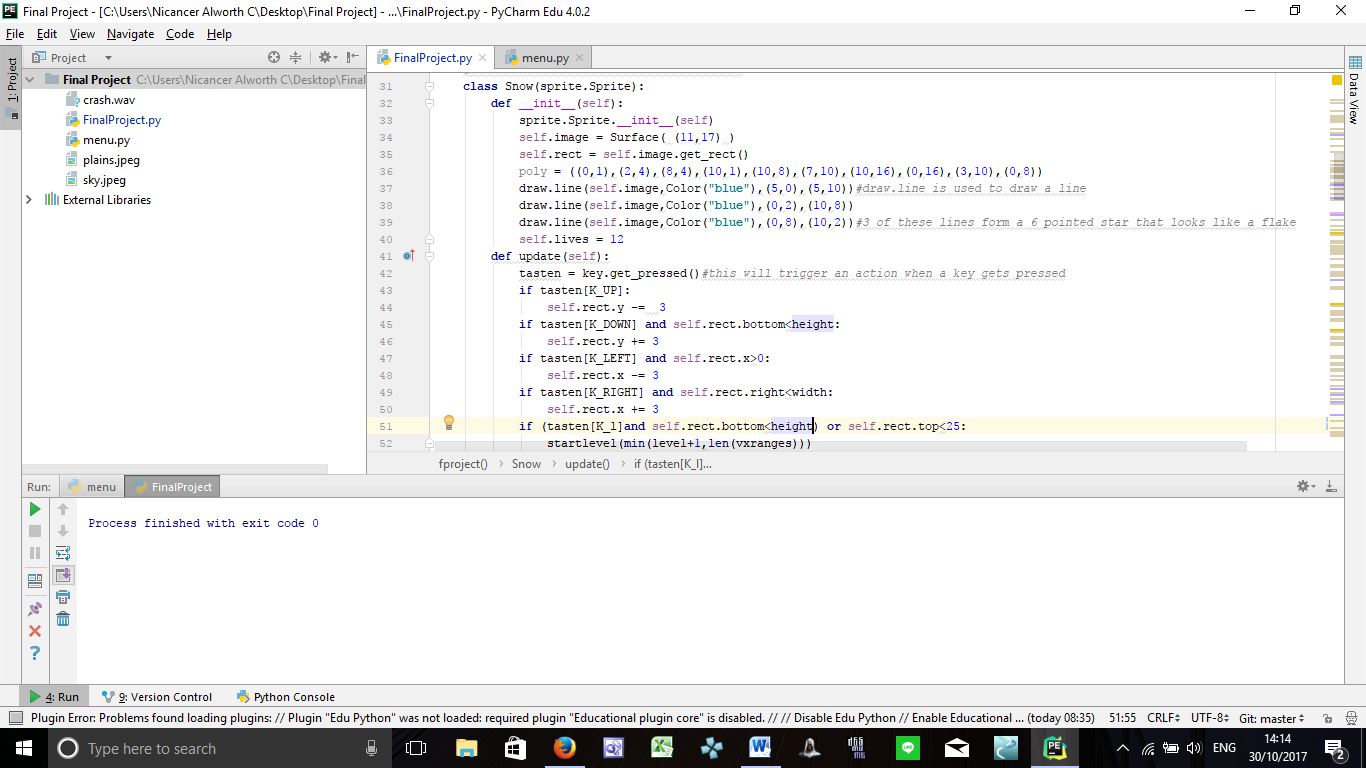
**In the while True:**

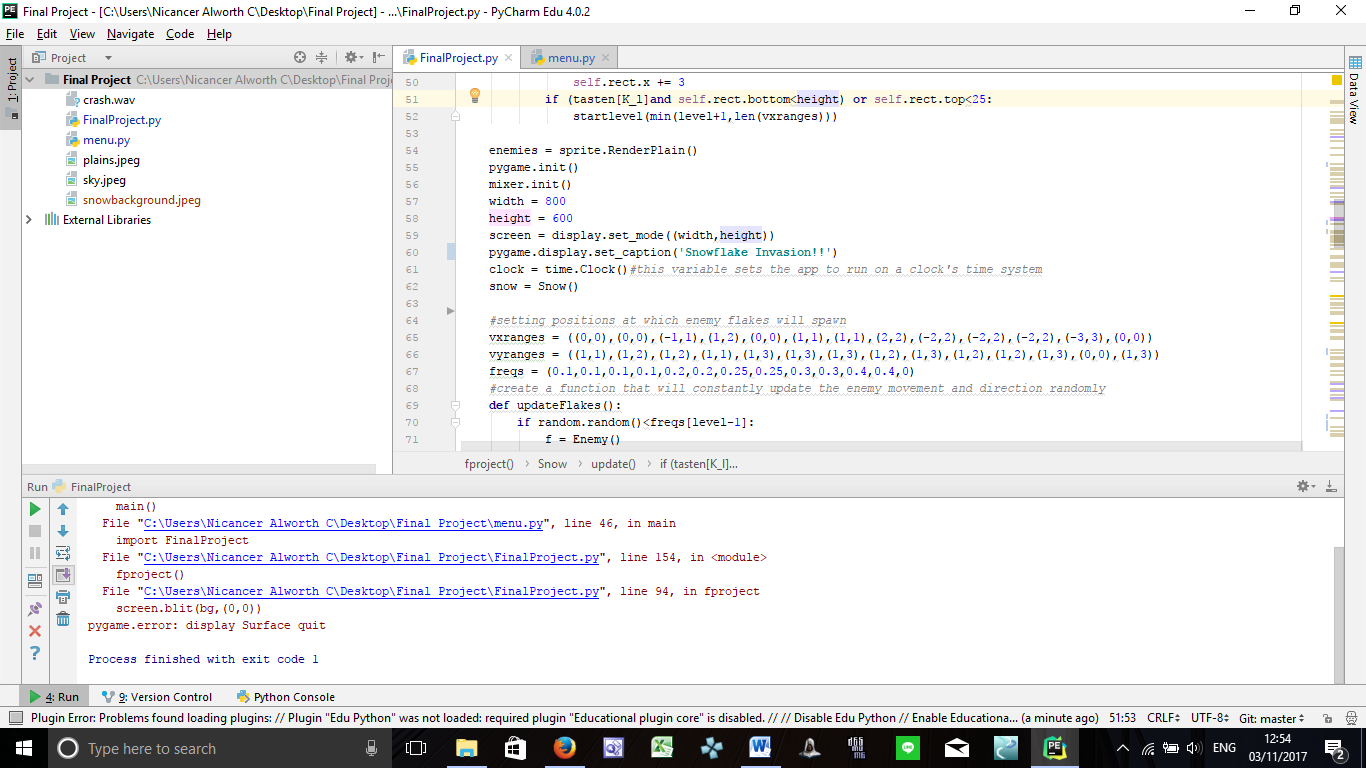
* various buttons are created for the user to interact with them. These buttons are the “Quit game” button, and the “Play game” button
* These buttons are created by using the pygame.draw.rect function to create a rectangle
* pygame.font.Sysfont initializes a font type so that we can write something using that font type by using the function myfont.render and screen.blit
* display.update() is used so that all changes made are applied
* for the quit game button, since the variable ‘Exit’ is used, we call exit.collidepoint function so that the button will react when the mouse cursor hovers around the button.
* Mouse.get\_pos() will return the position of the mouse cursor, therefore, exit.collidepoint(mouse.get\_pos()) will react when our mouse cursor hovers around the button
* Display.flip() acts like display.update() , that is to update any changes made. However, display.flip only updates one part of change, while display.update changes the whole document. Therefore, display.flip is used to update the change in color when the mouse cursor hovers around the button temporarily
* Finally, to activate the button , mouse.get\_pressed() is used and MOUSEBUTTONDOWN is declared so that the button will activate when the mouse clicks the button for both Exit game and play game

**The game: ( *FinalProject.py* )**



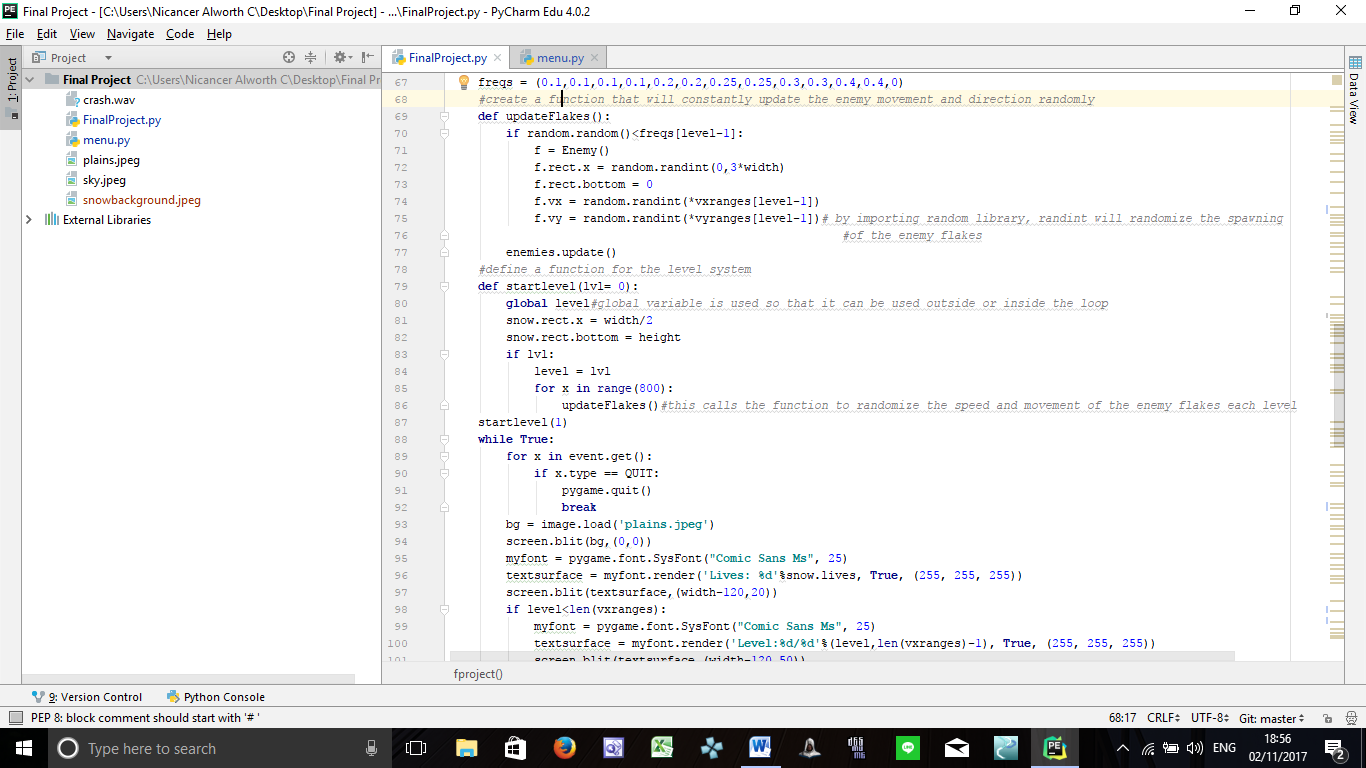
First, I created the class for the enemy sprites, by inheriting the sprite function from the pygame module. This of course, is possible by importing pygame at the start of the document. I used draw.line in order to draw a line from a point to another. These lines are drawn multiple times from different points in order to form some sort of star. The goal is to make these look like a snowflake. Inside the class I have two functions. The default init function to declare all the objects inside the class, and the update function which serves to allocate the movement of the enemy sprites. Self.rect.y and self.rect.x will represent the movement of the enemy sprites whether it moves along the x axis or it moves down the y axis. Inside it, I also added the condition that if the character collides with the enemy sprite, there would be a sound effect and the character will return to the bottom screen again, starting over the level.



This is the class for the character. Controls for the sprite are given using the function key.get\_pressed(). This results in user being able to control the direction of the sprite to the left, up, right , or down. I also made it so that if the player already masters a certain gap of levels and want to skip to a certain level, they can just press the key l as stated in the screenshot above( if (tasten[k\_l] e.t.c)

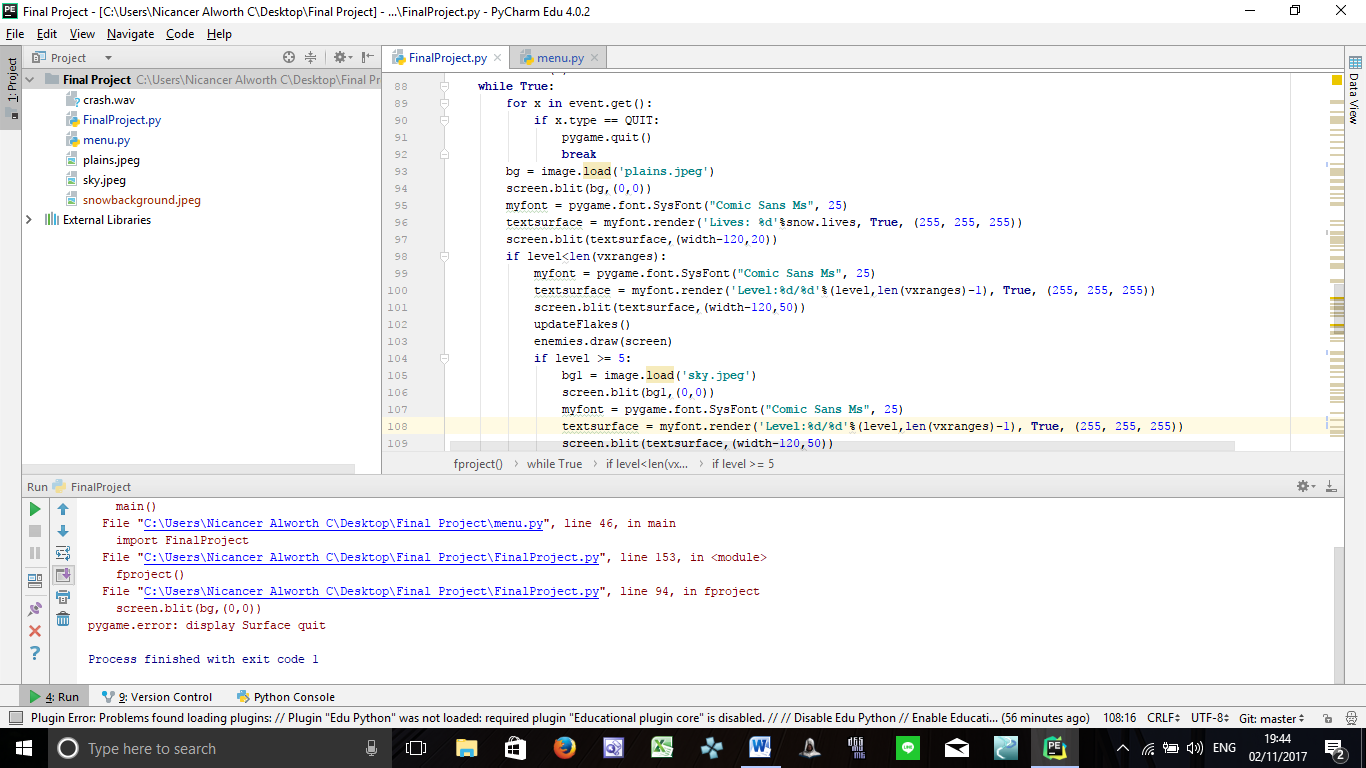
Next, sprite.RenderPlain() originates from the pygame module, to initialize the sprite on the screen. Pygame.init() is also used to create the screen in which the game will run on when the program is run. The height and width of the screen is set by creating the variable ‘width’ and ‘height’, which is then attached to the function “display.set\_mode((width,height))”. The module pygame.clock is also used to set the application to run on a clock’s system speed.

I also created the random locations in which the enemy sprite spawns initially, which is the vertical x ranges, and the vertical y ranges, followed by the freqs, which is the rate at which the enemy sprites spawn periodically.



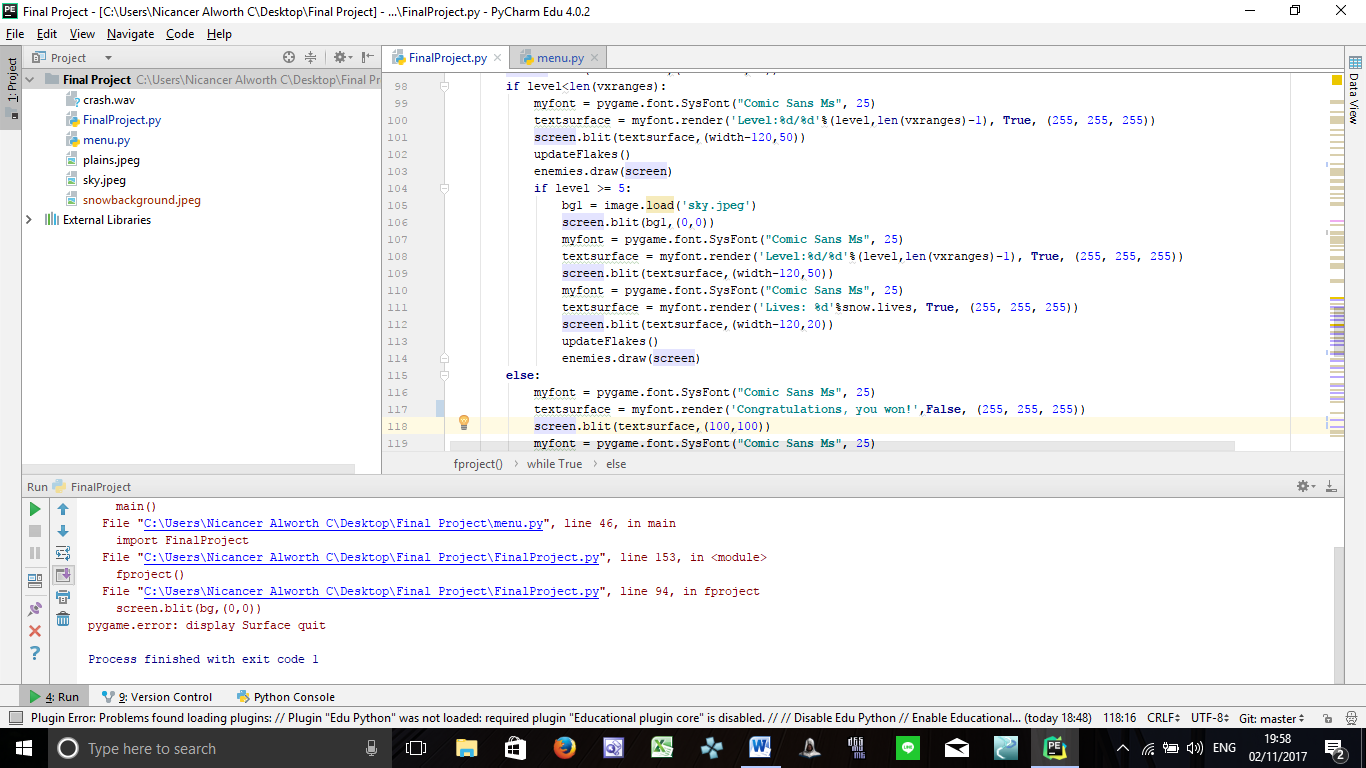
Next thing is the function updateFlakes(). This function’s purpose is to update the enemy flake’s movement and speed by using the random module imported at the start of the document. Therefore, the function startlevel() is also created so that the enemy flakes will be updated every level by calling upon the function updateFlakes(). Snow.rect.x and snow.rect.bottom initializes the position of the main character at the start of each level.

Therefore, as the level starts by one , we call the function startlevel with the parameter 1 inside it to initialize the first level.



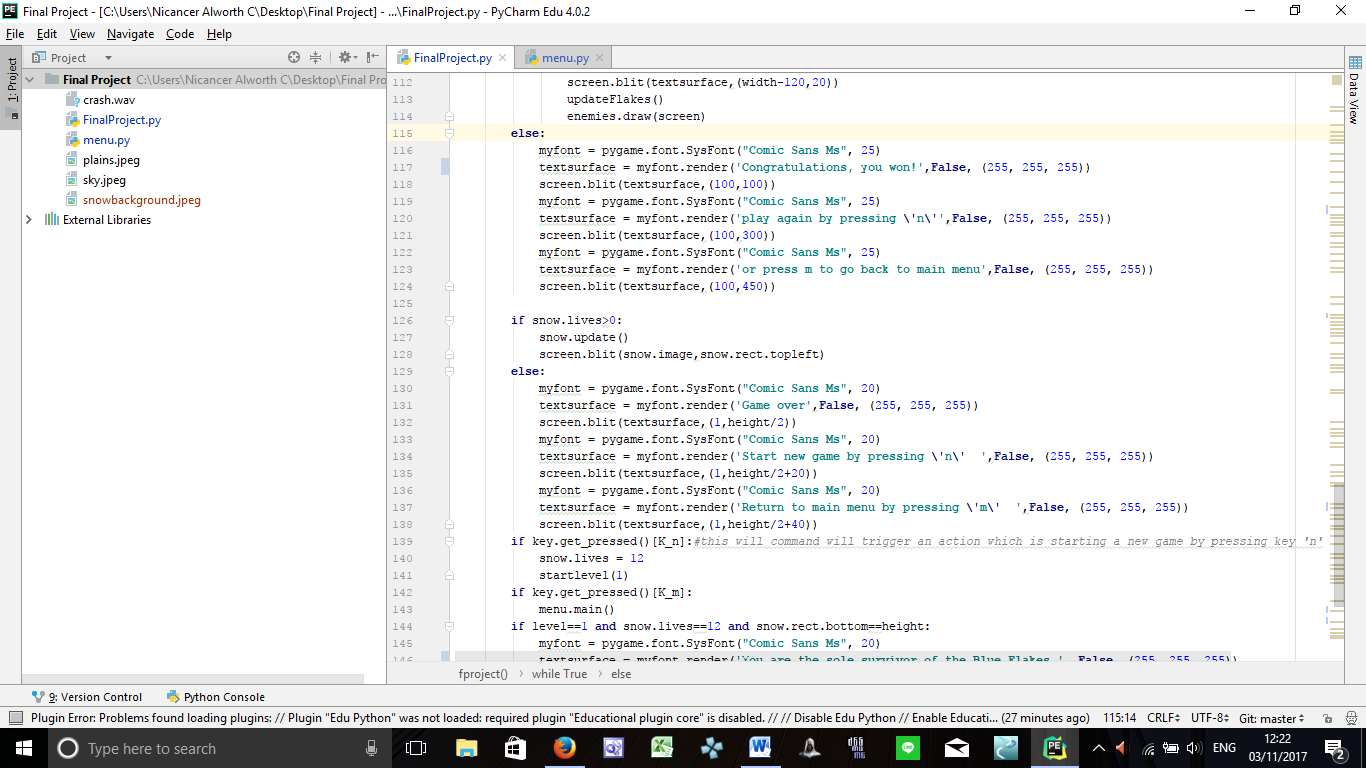
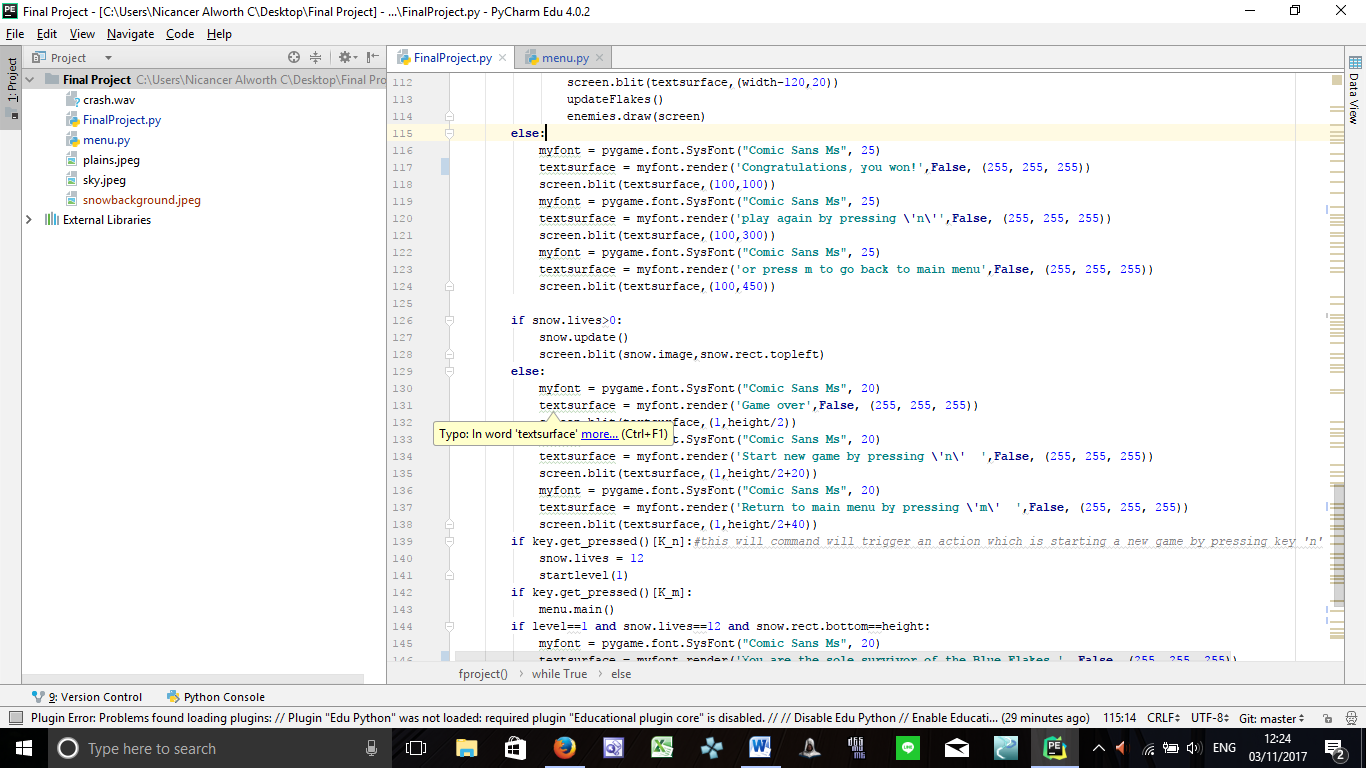
Next is the while True loop. First, I set the game to be able to exit when user clicks on the x button at the top right of the screen by using the command if x.type == Quit, pygame.quit()

After that, I set a background image for the game by loading the file plains.jpeg which is put at the same folder as the main python file. The function pygame.font.SysFont is used to set the font type and size, therefore, in the instance above, myfont has a comic sans ms font type and 25 font size. By calling myfont, using myfont.render will set the caption and colour of the text, followed by screen.blit which updates the text on to the screen.

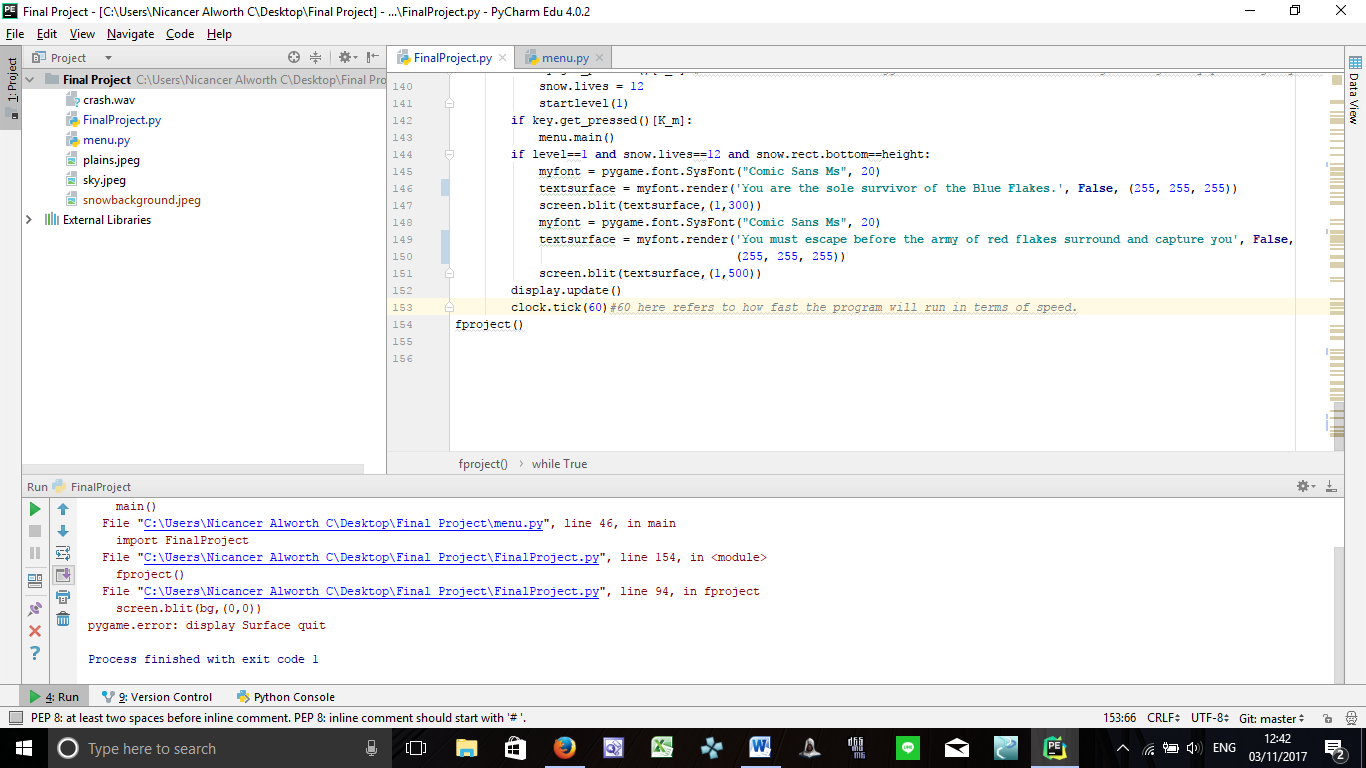


In the first condition, I made it so that if level is less than the total number of stages, which is 12, the enemy flake will still run, and it will constantly update the movement and speed of the enemy flake per level by calling the function updateFlakes().

The game will run with an additional condition, such as that if the player proceeds to level 5 and beyond, the screen background will change, according to the story.

Next, in order to enable the game to end, I created another condition, which is so that when the player reached the end of stage 12, a message containing ‘ Congratulations, plpay again by pressing n or press m to go back to main menu’ will show up.

Using the snow class which defines the main character, I called upon snow.lives to make a condition wherein if the character dies/ live = 0 , there should be a message showing ‘Game over, start a new game by pressing n or return to main menu by pressing m. In order to make this command applicable, I added a condition if key.get\_pressed()[K\_n] and key.get\_pressed()[K\_m]. Therefore, if user presses the key m, it will return to level 1 and has their lives back again, by calling the function startlevel(1) with 1 as its parameter to start the level, and if the user presses the key m, the user will be directed to the main menu



To make the game a little bit interesting, I added a little backstory at the start of the game by popping the message along the screen when level is equal to 1 and lives are still at full health. I used display.update() to update all changes made in the whole document instead of using display.flip() since it only updates one part of the change.

Clock.tick is used to set the particle movement and character movement, a.k.a the program run speed. The more the number inside the parameter, the faster the program will run.

Lastly, as I put everything in a function, I called the function name at the end so that the program will work.