CSE 1321L: Programming and Problem Solving I Lab

Assignment 6

Module 4

What students will learn

- Incorporate the programming concepts learned throughout the semester.
- Develop a larger scale, graphical program.
- Incorporate multimedia elements into a pygame application.
- Incorporate custom events into a pygame application.

Content

Overview

In this assignment, you will incorporate all the programming concepts we have covered, leveraging iterative and decision structures, data structures like lists, dictionaries, and tuples, and leverage Object Oriented Programming.

Included with this assignment instructions we provided you with some free assets: images, sprites, and sounds. These assets were found online, and they are all licensed as Creative Commons CC 0, meaning that the author has dedicated these assets to the public domain.

BEFORE YOU START, MAKE SURE YOU HAVE DECOMPRESSED THE .ZIP FILE TO ACCESS THE ASSETS PROVIDED AND THE DEMO VIDEO.

Make sure that on your group project you understand the different types of licenses and make sure you follow them. If you are unsure if you have permission to use an asset for your project, please reach out to your instructor as soon as possible.

By the end of this assignment, if you follow the instructions, you should end up with a well-simple structured project that you can use as a reference for your own project.

Before we go on into the instructions, let's go over some of the new pygame elements you will incorporate in this game.

Audio

In pygame you can play sounds, you can do so by using the pygame.mixer module. This module handles all things audio, such as effects and background music.

To start, you must initialize the mixer, this can be done alongside the pygame initialization:

```
import pygame
pygame.init()
pygame.mixer.init()
```

Then, we need a Sound object variable. We can create one using the mixer.Sound() constructor function:

```
laser_sound = pygame.mixer.Sound("laser.wav")
```

Notice that the constructor expects a single string argument. This is going to be the directory or path to your sound file.

In the case of this example, the path is relative to the main folder of your project, meaning we do not have to include the whole path such as:

```
"c:/users/yourName/Desktop/myProject/laser.wav"
```

Instead, as long as laser.wav is in the same folder as your main Python file (or in a subdirectory you correctly reference, like "sounds/laser.wav"), this relative path will work just fine. This keeps your code cleaner, more portable, and easier to maintain if you move your project to a different machine or directory.

Now that we have our Sound object, it is ready to play. To play the sound, you must reference the object, in our example we have the variable "laser_sound", and call its .play() function.

Throughout this project, we will use this to play the spawn sound for the meteors. Every time they are spawned, we are going to play a sound.

Text

In pygame, we can also draw text. This is useful whenever we want to give instructions, show status, show dialogue, etc.

To show text into the display we must create a Font object using this constructor function.

```
font = pygame.font.Font(None, 24)
```

This creates a Font object that uses the default font (we can specify other fonts if you have a font file such as a .ttf or otf) and has a size of 24 points.

To specify the text of this font, you must create surface using the font.render() function:

```
text_surf = font.render("Hello World", True, (0, 0, 0))
```

The first parameter will be the actual text, while the second parameter is a Boolean argument that decides whether the text has enabled anti-aliasing which smooths the edges of the text.

Lastly, notice that the third argument is a tuple with three integers, as may suspect this is an RGB value therefore the color of the text surface.

We can also enclose this surface in a "rect" to manipulate its positioning:

```
text_rect = text_surf.get_rect(midbottom = (screen.get_width() / 2, screen.get_height() – 10))
```

In this example, we have positioned the text to appear centered along the bottom edge of the screen.

Lastly, to draw the image into the screen we just need to blit it:

```
screen.blit(text_surf, text_rect)
```

We are going to use text to display to the user when the game is over.

Images

Another asset type we can use in pygame are images. To use an image in pygame we first need to load it.

```
myImage = pygame.image.load("game_logo.png")
```

This will load an image surface. The load() function returns a surface object and expects a string argument. Notice how this argument works the same way as the sound. It is the path to the image file, and we can also use the relative path to it.

If an image has transparency, which may be the case for this logo image, we must use the convert_alpha() function or else instead of the transparent portion of the image will show as black:

```
myImage = pygame.image.load("game_logo.png").convert_alpha()
```

Then, we can enclose this surface in a rect:

```
Image_rect = myImage.get_rect(topleft = (0, 0))
```

Lastly, we must draw it onto the screen:

```
screen.blit(myImage, image_rect)
```

For this assignment, we are going to use images to draw the meteors sprites (there are different types of meteors), the player sprite, and to draw a background image.

Movement – Key press handling

There are two ways to handle key presses in pygame, through event-based key handling and state-based key handling. We are going to use the latter since it allows the user to continuously move the character by just holding a key instead of repeatedly having to press it.

To start, we need to call the key module get_pressed() function. This function checks the current state of all keys, it returns a sequence of Boolean values mapped to all the keys in the keyboard.

```
keys = pygame.key.get_pressed()
```

After getting this sequence, we can then use if statements to perform actions when a key is being pressed:

```
if keys[pygame.K_a]:
      // Move left
elif keys[pygame.K_d]:
      // Move right
```

We will leave the movement part up to you, remember that we have covered this in Rect movements.

Custom Events

For this game, we need to be able to spawn meteors given a random amount of time. There are lots of ways to handle this such as having a timer, but this is sometimes cumbersome and hard to implement correctly. Instead, the better way to handle timed events is more accurately by creating our own user event:

```
MY_CUSTOM_EVENT = pygame.USEREVENT + 1
```

Pygame reserves event types from pygame.USEREVENT (typically value 24) to pygame.USEREVENT + 7 for custom user events, giving you 8 slots to define your own. Some of the events pygame includes are QUIT, KEYDOWN, and KEYUP to mention some you may have seen before.

Since we want to add a new user event, we must initialize it as number 25, hence the + 1.

Since we want to fire up our event based on a timer, we must schedule it like this:

```
pygame.time.set_timer(MY_CUSTOM_EVENT, 1000)
```

This sample code schedules event number 25 every 1 second, the second parameter is an integer value representing milliseconds.

We have scheduled the event, now we must be able to read or catch the event and trigger something:

```
for event in pygame.event.get():
    …
    if event.type == MY_CUSTOM_EVENT:
        print(“Hello World”)
```

Notice that the for loop we are using is the same as the one we use to set the pygame.QUIT event. This is because if you remember QUIT is also another user event.

Regarding our own event, every 1 second our event will get triggered and it will print into the terminal “Hello World”

In the assignment, we are going to instead spawn a meteor. We will discuss the time interval and the location of the meteor in the instructions.

Attached Files

We have attached 4 audio files and 12 image files. The game must be used in all 16 files, and you must exclusively use the assets we have provided along with the instructions.

You must store these files inside your project file. In your project file you should have two sub-folder named “img” and “audio” where you will save the files in their respective folders.

This is so your project is well structured and organized, instead of having everything spread in the project folder.

Use this structure as a base reference on how your group project file should look like.

Regarding the Configuration.py file

We will discuss the details on what this file should contain, but we whole purpose of this file is so we can have one file that handles all the “hardcoded” values as well as the “settings” for the game.

This will be useful whenever you want to change some of the parameters in the game without having it to change it throughout the different files.

For your group project you should try to have a configuration file so you can easily change parameters in the future or while you develop the game.

Regarding the Assets

If you liked the assets we provided, they are all sourced from kenney.nl/assets. Most of the assets published are licensed as CC 0 so you can use them in your project.

Do not rename the assets. Your submission will not include the assets; therefore, your program should work completely fine on our computers as long as you use the exact filenames and folder structure provided. Make sure your code references the assets using relative paths like "img/player_sprite.png" or "audio/spawn_sound_1.ogg" so everything loads correctly on our end.

Assignment 6: Meteor Dodge

For this assignment you are going to create a simple game where the player needs to dodge falling meteors by moving side to side.

Included along with the instructions are some assets provided for you that you must implement in your deliverable.

Remember to review the Demo Video included on how the game should look and behave.

To develop this game, we are going to leverage all the concepts you have learned so far in the semester, especially importing our own modules/scripts and object-oriented programming.

Let's start with the configuration file.

Config.py

We are going to use this file to set up some important constant variables that we are going to use throughout the game. The names of all variables, lists, tuples, and dictionaries here must be ALL UPPERCASE.

General settings:

- We need to use the pygame module in here so start by importing pygame.
- Create a variable from the width and height of the game which should be 480 x 800.
- Create a variable for the frame rate of the game which should be 60.
- Create a custom pygame event to handle the meteor spawning event (refer to the assignment Overview section).
- Create two tuples called BLACK and WHITE and give them their appropriate RGB color value.

Player settings:

- Create a variable to store the player's speed initialized as 12
- Create a variable to store the player's image sprite file path, make sure you handle the correct path to this image.
- Create a variable to store the player's death sound file path, make sure you handle the correct path to this sound file.

Meteor settings:

- Create a dictionary that handles 4 different speeds based on the meteor's size. Make sure you use the size of the meteor (str) as key:
- Big meteors have a speed of 10
- Medium meteors have a speed of 11
- Small meteors have a speed of 12
- Tiny meteors have a speed of 13
- Create a dictionary that stores the different paths to the various meteor images and sized. Use the size of the meteor (str) as the key, the value of these keys should be a list containing the paths to the corresponding images. Use the following code as reference:

```
PATHS = {
            "big": ["img1.png", "img2.png"],
            "med: [], # and so on...
}
```

- Create a list that stores the meteor span sounds paths. Make sure you handle the correct path to all the meteor spawn sound files.

Since the game will consist of a player and several meteors, we will create two classes to encapsulate them.

Meteor.py

- Create a class called "Meteor"

Import Modules

- In this class you must import the config.py file you have just created.
- This class should also import pygame and the random module.

Constructor

- Define the class constructor function, it will not expect any parameters (except **self**).
- Define these class attributes and objects:
    - **type:**
        - It is going to store the type of meteor (Big, Medium, Small, Tiny).
        - The code must randomly choose the type of meteor.
    - **speed:**
        - This attribute stores the speed of the meteor.

        - This is based on the meteor speed dictionary initialized in the configuration file.
        - Use the “type” attribute to retrieve the corresponding speed based on the meteor size.
    - **sprite:**
        - This is a pygame.image object.
        - Your program should randomly choose an asteroid based on the “type” attribute.
        - Remember, we have 4 different sprites for Big asteroids and 2 different sprites for each medium, small, and tiny asteroid.
    - **rect:**
        - This attribute stores a pygame.Rect object.
        - Construct this Rect using the “sprite” attribute.
    - **spawn_sound:**
        - This is a pygame.mixer.Sound object.
        - The program should randomly choose one sound file out of the three spawning sound provided.
    - **rect.topleft:**
        - This is not an attribute, but to specify the “spawn” location of the meteor in the screen.
        - Meteors will spawn from the top of the screen (y = 0).
        - The program should choose a random x coordinate so it does not spawn on the same spot.
        - Make sure the entire sprite is inside the screen/display.

Functions

Finally, the Meteor class will contain the following functions (do not forget to add **self** as a parameter):

- **fall():**
    - This function does not take any parameters, and it should just move the meteor downwards. This should be achieved using the object’s rect and speed attributes and the rect.move() function.
- **draw():**
    - This function should take in a surface (the target surface) as parameter. The function should use the parameter surface (target surface) to blit the object’s sprite and rect attributes.

Player.py

- Create a class called Player

Import Modules

- In this class you must import the config.py file you have just created.
- This class should also import pygame.

Constructor

- Define a constructor function that also does not take in any parameters (except **self**).
- Define these class attributes and objects:
    - **speed:**
        - This variable stores the speed of the player.
        - Its value is based on the player speed defined in the configuration file.
    - **sprite:**
        - This is a pygame.image object that loads the player image file specified in the configuration file.
    - **rect:**
        - This is a pygame.Rect object. Use the sprite attribute to initialize it using the method sprite.get_rect().
        - Also, define the X and Y coordinate of this rect attribute so the player model is centered in the x-axis and a 20-pixel margin from the bottom edge of the screen.
    - **alive:**
        - This is a Boolean variable to keeps whether the player is alive or dead (hit by a meteor)
    - **deadSound:**
        - This is a pygame.mixer.Sound object.
        - Use the specified path in the configuration file to load the corresponding sound file.

Functions

Finally, the Player class will have the following functions (do not forget to add **self** as a parameter):

- **move():**
    - This function should take in a parameter called keys.
    - This parameter will be of type pygame.key.get_pressed()

        - Use this parameter to check if a player pressed the key “a”. If so, move the rect using the rect.move() function so the player rect+sprite move towards the left.
        - Use this parameter to check if a player pressed the key “d”. If so, move the rect using the rect.move() function so the player rect+sprite move towards the right.
        - The player sprite should not move outside of the bounds of the screen. If the player sprite gets to the right edge of the screen, it should just stay there even if the user keeps pressing the “d” button. Same with the left edge of the screen.
    - **draw():**
        - This function works the same as with the Meteor draw() function.
        - Take is a surface as a parameter and blit the user sprite into the target surface.

Assignment6.py

This is the main file of the game.

Import Modules

- Make sure you import the pygame and random module, the Meteor and Player class, and the Configuration File.

Setup (Before the main loop)

- Initialize pygame and pygame.mixer.
- Set the custom meteor spawn event.
- The first meteor spawn event should happen 1 second after the game starts.
- Set the Display surface using the size specified in the Configuration File
- Create a pygame.time.Clock object.
- Create a font and text surface object.
    - This font should use the default font style and have a size of 48
    - The text of this font should be “You Lost!”
    - The text of this font should be colored white (use the configuration file).
    - The placement of this text should be in the center of the display.
- Create a surface object
    - This surface will be used to add the darker shade to the display when the player loses the game.
    - The size of this surface should be the same as the size of the screen (use the configuration file).
    - Set the opacity of this surface to ‘100’
    - Set the color of this surface to black (use the configuration file).
- Create an image object and load the background image.

- Create and initialize a list called meteors.
- Create and initialize a Player object.

Main Loop

- Implement a for loop that traverses the pygame.event.get() events.
    - Handle the termination of the game when the user clicks on the close button in the windows title bar.
    - Handle the SPAWN_METEOR_EVENT from the configuration file. When triggered it should:
        - Create a Meteor object.
        - Append the meteor object into the meteors list.
        - Play the meteor object's spawn sound attribute.
        - Reset the spawn meteor event by calling the pygame.time.set_timer() method again. Now, instead of the timer being set to 1 second, choose a random number from 800 to 1500.
- Blit the background image.
- Traverse the meteors list and:
    - Draw the meteor into the screen using the meteor's draw() function.
    - Call the meteor's fall() function.
    - Since we do not want to end up with a list that grows overtime, if the meteor is at the bottom edge of the screen, delete it from the list.
    - Check if the meteor has collided with the player. If it has and the player is alive:
        - Play the player's dead sound (use the player's attribute)
        - Set the player's alive attribute to False.
        - Remove the meteor from the list.
- Create and initialize pygame.key.get_pressed().
    - We are going to pass this object as an argument for the player's move() function.
- If the player is alive, the player should be able to move (use the player's move() function) and draw the player into the screen.
- If the player is not alive (lost the game), blit the shade surface and blit the font we created at the beginning of this file.
- Lastly, call the clock.tick() function and pass the FPS value from the configuration file and flip the display.

Submission Instructions:

- Programs must follow the output format provided in the Demo Videos.
- Programs must be working correctly.
- Programs must be written in Python.
- Programs must be implemented exclusively using the Pygame module.
- Programs must be submitted with the correct **.py** format.
- Programs must be saved in files with the correct file name:
    - Assignment6.py
    - Config.py
    - Meteor.py
    - Player.py
- **Do not include the assets in your submission.**
- Programs (source code files) must be uploaded to Gradescope by the due date.