The Pygame library is probably the most well known python library when it comes to making games. It’s not the most advanced or high level library, but it’s comparatively simple and easy to learn. Pygame serves as a great entry point into the world of graphics and game development, especially for beginners.

The Pygame framework includes several modules with functions for drawing graphics, playing sounds, handling mouse input, and other things that you’ll need while developing games in Python.

**Setting up our Game**

We’ll begin this Python tutorial by explaining several core concepts related to the Pygame library and about creating games in general. Also keep in mind, that many of these concepts are **transferable skills**. Should you switch to a more advanced game engine later many of these concepts will still hold true.

A picture containing text, font, white, graphics

Description automatically generated

In the above code we begin importing pygame and it’s modules into our python program. The second line allows us to use the functions and variables in the **pygame.locals**module without having to add the lengthy **pygame.locals** prefix.

A picture containing font, text, white, graphics

Description automatically generated

The init() function in pygame initializes the pygame engine. This line must be included before you begin writing any pygame code.

## The Game Loop

The Game Loop is where all the game events ***occur, update and get drawn*** to the screen. Once the initial setup and initialization of variables is out of the way, the Game Loop begins where the program keeps looping over and over until an event of type QUIT occurs.

Shown below is what a typical Game loop in Pygame looks like. It’s a simple “while” loop that runs infinitely.

A picture containing text, font, screenshot, white

Description automatically generated

Changes in the game are not implemented until the display.update() has been called. Since games are constantly changing values, the update function is in the game loop, constantly updating the window every iteration, with any changes that may have been made.

We place it at the very end so that all possible changes to the Sprites on the screen have already taken place. We could call this more than once, but that would be more performance intensive.

### Quitting the Game loop

Every game loop must have a end point, or some action that triggers the end point (such as clicking the quit button), else your game will run indefinitely.

A picture containing text, font, screenshot, line

Description automatically generated

We call both pygame.quit() and sys.exit() to close the pygame window and the python script respectively. Simply using sys.exit() can cause your IDE to hang due to a common bug. (Remember to [import the sys library](https://coderslegacy.com/python/libraries-in-python/python-sys/) to use this function).

Side note: If you didn’t import everything from *pygame.locals* as we did you would have to use *pygame.locals.QUIT* instead of *QUIT*.

**Event Objects in Pygame**

A Pygame “Event” occurs when the user performs a specific action, such as clicking his mouse or pressing a keyboard button. Pygame records each and every event that occurs. However it won’t really do anything with this information, because that part is up-to us.

We can find out which events have happened by calling the pygame.event.get() function (shown previously), which returns a list of pygame.event.Event objects (which we will just call Event objects for short).

One of the many attributes (or properties) held by event objects is type. The type attribute tells us what kind of event the object represents.

A black text on a white background

Description automatically generated with low confidence

If you take a look at the example above again, you’ll see we used event.type == QUIT to determine whether the game was to be closed or not. We can even create our own [custom events to signal](https://coderslegacy.com/python/pygame-userevents/) certain types of events (such as an enemy spawning).

**Creating a Display Screen**

For every game, we need to create a window of a fixed size by passing a tuple containing the width and height. This tuple is then passed into the display.set\_mode() function.



In pygame and other game libraries, we regard the (0, 0) coordinate as the top-left most corner. Similarly, the maximum x-point and maximum y-point is the bottom-right corner. Which in this case is (300, 300).

The X-values grow larger as you move left to right, and the Y-values grow larger from top to bottom.

A picture containing text, rectangle, screenshot, line

Description automatically generated

You can also customize this window later by changing it’s title and [the default icon](https://coderslegacy.com/python/how-to-change-the-pygame-icon/).

## Pygame Colors

Colors are going to be a big part of any game development framework or engine, so you should understand it well.

Pygame uses the typical RGB system of colors. To those who aren’t aware, this stand for Red, Green and Blue respectively. These three colors combined (in varying ratios) are used to create all the colors you see on computers, or any device that has a screen.

The values for each color range from 0 – 255, a total of 256 values. You can find the total number of possible color combinations by evaluating 256 x 256 x 256, which results in a value well over 16 million.

In order to use colors on Pygame, we first create **Color objects** using RGB values. RGB values must be in a tuple format, with three values, each corresponding to a respective color.

A picture containing text, font, white, handwriting

Description automatically generated

Shown above are examples of how we can create Colors. We will use these later on in our Pygame tutorial when we begin creating backgrounds or [shapes](https://coderslegacy.com/python/pygame-draw-shapes/).

### Frames per second

Computer’s are extremely fast and can complete millions of loop cycles in under a second. Now obviously, this is a little fast for us humans. As reference, movies are run at 24 frames per second. Anything less than that will have an obvious stutter to it, whereas values over 100 may cause the things to move too fast for us to see.

By default, if we do not create a limitation the computer will execute the game loop as many times as in can within a second. This is actually a major problem, because without a “limiter” the frame rate will fluctuate greatly throughout the game depending on what’s currently happening (number of objects on screen, player moving or not, etc.)

To limit it we use the tick(fps)method where fps is an integer. The tick() method belongs to the pygame.time.Clock class and must be used with an object of this class.

A picture containing text, font, white, graphics

Description automatically generated

This can vary from game to game, depending on how it was designed but you should aim for a value between 30 – 60. Keep in mind, that if you create a rather complex and heavy game the computer might not be able to run it well at higher frames.

**Rects & Collision Detection in Pygame**

In every game, each object has fixed boundaries that define the space that it currently occupies. These fixed boundaries are essential when the object interacts or “collides” with other objects.

By defining these boundaries, the game is able to detect when two or more boundaries overlap or touch. This allows it to then handle the interact based on which objects are touching. Such as the Player picking up an item, or attacking another entity.

Pygame rect example

Shown in the image above is a typical “rect” object (colored) around a Car. It’s not 100% accurate, as it does not full take on the shape of the Car but it is accurate enough for most purposes.

To check for collisions, we have [various methods and functions](https://coderslegacy.com/python/pygame-rect-tutorial/), each used for a slightly different purpose. For example the below code is used to check for collisions between two Rects. (Can you guess whether it will return True or False?)

A picture containing text, font, white

Description automatically generated

We can also check for a collision between a Rect and a pair of coordinates.A picture containing text, font, white

Description automatically generated

There is another trick we can use to automatically create a Rect based off an image’s dimensions. We will explore this later on in this Pygame tutorial.

With this, we are now done with basic theory in our Pygame Tutorial. We will now move on to actually utilizing and applying these concepts to create a proper game.

**Game Creation – Part#1**

Time to begin the second half our Python Pygame Tutorial. We’ve discussed some basic concepts and code, now lets explore how we can use this information to make a proper game.

Whether it’s GUI, Pygame or any other large application, the Classes approach (OOP) is almost always the best idea (Unless you have a really simple and small program). Using Classes, we’ll be using methods to store blocks of code that are to be repeated several times throughout the game. An object of each Class will represent an entity in our game (such as an Enemy, or the Player).

Although our game will only have one Enemy and one Player, this may not always be the case. We may expand this game later by adding more enemies, in which case all we would have to do, is create a new enemy object from our Enemy Class.

Below is the initial version of our game. It’s not yet complete, but the foundation has been set.

A screenshot of a computer program

Description automatically generated with medium confidence

### Code Explanation

A picture containing text, font, screenshot, line

Description automatically generated

Above you can see the Code for the Player Class. The benefit of using classes here is that we can spawn multiple entities from the same block of code. Now, this doesn’t really apply to the Player Class, since most games will only have one player but it does apply to the Enemy Class as most games will have multiple enemies.

Passing pygame.sprite.Sprite into the parameters,makes the Player Class it’s child class. Passing super().init() then calls the init() function of the Sprite class. super().\_\_init\_\_() is a whole different concept related to Classes in Python. You can look it up if you’re interested, else just include it the way we’ve shown above.

Next is the **image.load()** function to which we pass the file path of our image. Note, this does not define the borders for our Player Sprite. This is instead done using the get\_rect() function. This function is able to automatically create a rectangle of the same size as the image. We will be using this in [Collision Detection](https://coderslegacy.com/python/pygame-rpg-collision-detection/) later on.

The last line, self.rect.center, defines a starting position for the Rect. Later we’ll use the Rect’s coordinates to draw the image to the exact same location. If you aren’t careful, you might end up with the Rect and the Image in two different places.

A picture containing text, font, screenshot

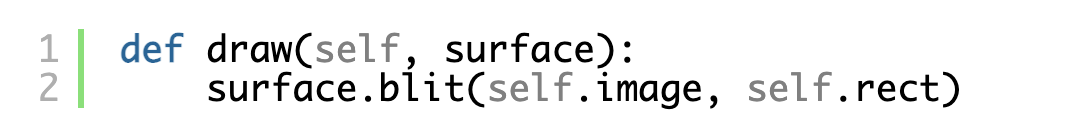
Description automatically generated

This is a method from the Player class that controls the movement of the player. When this function is called, the checks to see if any keys are pressed down or not.

The if statements we’ve included after this, check for 4 keys, UP, DOWN, LEFT and RIGHT. If the if statement proves true, then the move\_ip() method is called on Player.rect moving it in a certain direction. The move\_ip() takes two parameters, the first representing the distance to be moved in the X direction and second, the distance to be moved in the Y direction.

The two if statements, if self.rect.left > 0: and if self.rect.left > 0: ensure that the player isn’t able to move off screen.

Two of the **if statements** are commented out because this is a side scroller game. We don’t need up and down movement here. We only included them to show you how it would be done.



The blit() method takes two inputs, the first the surface to be drawn to and second, the object which we want to draw. Normally we would write surface.blit(self.surf, self.rect) since we’re drawing the rectangle to the surface we’ve defined. But since we’re using an image, we pass self.image instead of self.surf. (An image is in fact, a surface in Pygame)

[Surfaces](https://coderslegacy.com/python/pygame-surface/) play an important role in Pygame, and we can’t hope to cover it all here, so we’ve given it’s own article. Read it if you’ve had any difficulty understanding surfaces, the **blit()** function or anything related to it. The enemy class is setup very similarly, so we don’t need to discuss it too much.

A picture containing text, font, screenshot, white

Description automatically generated

The only change is with the last line, where we included randomized starting points. (It would be pretty boring if the Enemy appeared from the same location each time)

A picture containing text, font, screenshot, line

Description automatically generated

This method is part of the Enemy Class. It first calls the move\_ip() function, moving the Enemy object down by 10 pixels. Next it checks to see if the top of the Enemy has reached the end of the screen. If True, it resets it back to the top of screen and at a random location on the X axis.

A picture containing text, font, screenshot, white

Description automatically generated

The commands shown above are all in the game loop, so they are repeating continuously. First the update and move functions for both the Enemy and Player class are called.

Next we refresh the screen using the DISPLAY.fill(WHITE) function, finally we call the draw functions for both the Player and Enemy objects, drawing them to the screen.

Finally, the pygame.display.update() command updates the screen with all the commands that have occurred up-till this point, and the tick() makes sure it repeats only 60 times per second.

## Game Creation – Part 2

Our game is still pretty incomplete. There’s no fun in playing a game with the same thing happening over and over again. There is no end point, no variation in the game difficulty and most importantly, there are no consequences of colliding with the enemy.

In this section we’re going to cover Sprite Grouping, [Collision Detection](https://coderslegacy.com/python/pygame-sprite-collision-detection/), [User events](https://coderslegacy.com/python/pygame-userevents/) and some other minor features.

A screenshot of a computer code

Description automatically generated with low confidence

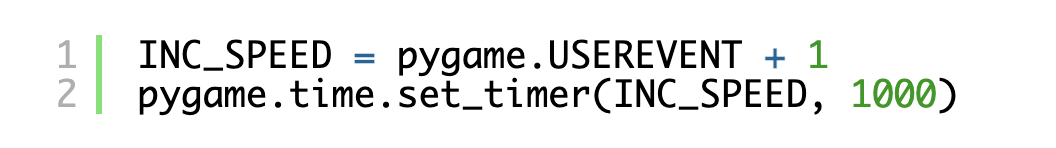
### Explanation

A picture containing text, font, screenshot, white

Description automatically generated

In this section, we’ve created “groups” for our sprites. A Sprite group is sort of like a classification. It’s much easier to deal with 2 or 3 groups, rather than having to deal with dozens or even hundreds of sprites. Keeping them in one group allows us to easily access every sprite in that group.

In the example above, we’ve created two groups one called enemy and the other called all\_sprites. (This code doesn’t have more than one enemy, but since multiple enemies could easily be added here, we’ve created a separate group for it). To add a Sprite to a group, you just have to use the add() function.



We talked about event objects earlier, such as QUIT. Python Pygame, gives us the option to create custom events called “User events”. Here we’ve created an event called INC\_SPEED. To do so, we called the pygame.USEREVENT and added one into it (to ensure that it will have a unique ID). More about [events in a separate tutorial](https://coderslegacy.com/python/pygame-userevents/).

Next we use the Pygame time module’s set\_timer() function to call the INC\_SPEED event object every 1000 milliseconds, or 1 second.

A picture containing text, font, screenshot, line

Description automatically generated

The next piece of code if about the Game loop. In the for loop where we iterate over every event that occurs, we insert an if statement to check for the INC\_SPEED event occuring. If it does, we increase the SPEED variable by 2. This variable us used by the Enemy class to determine it’s speed.

All in all, the purpose of this code is to make the game more challenging as time passes.

A picture containing text, screenshot, font

Description automatically generated

This section of code is related to collision detection in Python pygame. Remember how we created groups earlier? You’re about to see a massive benefit that we get from having meaningful groups.

The spritecollideany() function takes two parameters, the first must be a regular Sprite, like P1 or E1. The second must be a Sprite group, such as **Enemies**or **all\_sprites**. This function compares the sprite passed in the first parameter, to see if it’s touching any of the sprites in the group passed in parameter two.

In our case, it checks to see whether our Player has collided with any of the sprites in the enemies group. The benefit of this function is that even if there are 1000 enemy sprites, we don’t have to check collisions with them individually, and instead just use this function.

Finally, the collision holds True, we kill all the sprites using the kill() function, fill the screen with red, wait two seconds and close the entire program.

(Calling kill() will remove the sprite from the group, hence it will no longer be drawn to the screen. If you don’t use groups, and try kill() on a sprite, you might not get the desired effect)

A black text on a white background

Description automatically generated with medium confidence

Yet another benefit of the grouping system, we can now call the “move” functions for all the sprites and redraw them in just 3 lines of code. If you’ve noticed, we’ve removed the two draw() functions from both the Player and Enemy class in our code.

## Game Creation Tutorial in Pygame – Part 3

This is a bit of an additional, and also optional section in our Python Pygame tutorial. In this section, we’re going to cover **backgrounds**, **sound**, **fonts**and a **scoring**system. These are all important features one needs in a game, to make it a complete product.

As usual, take a good look at the code below before proceeding to the block by block explanations.

A screen shot of a computer code

Description automatically generated with low confidence

**Explanation**

**A black text on a white background

Description automatically generated with low confidence**

First up is the fonts. In the code above, we’re setting the fonts to be used later on in our program. We create two different fonts, font and font\_small which both have the same font family, but different font sizes.

Next we use the render() function to actually create the graphics for the Font of our choice. We also have to pass in the text we wish to be displayed and the color we want it to be in. In this case, “Game Over” and BLACK respectively.

A screenshot of a computer code

Description automatically generated with medium confidence

This is the second part of our fonts. You can see us rendering another font called **scores**. We didn’t do this earlier because this font is meant to be rendered inside Game loop as it has a continuously changing value. We moved the **game\_over** font rendering out of the loop to avoid unnecessary performance loss.

Finally, we display both fonts using the blit() function. In it’s first parameter we pass the rendered font and in the second we pass a pair of co-ordinates which mark the origin point of the font.

A screenshot of a computer program

Description automatically generated with medium confidence

There are two steps to creating a background. First you load the image (outside the game loop for performance) and then proceed to draw the image using the blit() function. The blit() function must be in the game loop as it needs to re draw itself as the other objects move.

What’s also important, is that the background is the first thing that’s drawn. Pygame has a layer system, where if you aren’t careful, you will draw the background over all the other sprites. Basically it’s like painting. If you draw the Player on the street, it’s ok. But if you draw the street over the player, it’s a problem.

A picture containing text, font, screenshot

Description automatically generated

Here we are using a simple one line function from the [Pygame Mixer library](https://coderslegacy.com/python/pygame-mixer/) to play a crash sound once collision has occurred. Pygame.mixer.Sound() is used to load the sound into Pygame, while play() is used to actually “play” it.

The images that have been used in this code can be found in GOL in the PyGame Tutorial folder in ‘Introduction to Event Driven Programming’.