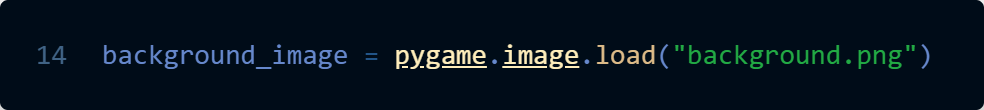
Understanding Image Rendering, Attachment, and Animation in Game Development

## 1. Code Explanation

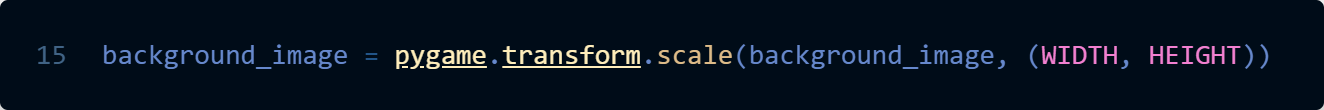
In this section, we will go through the code step by step and explain its functionality and purpose.

### 1.1. Rendering Images

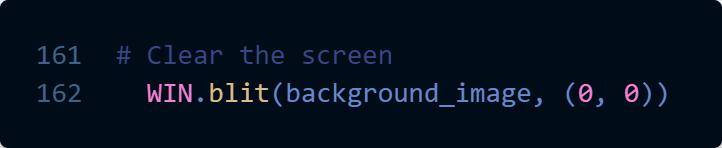
Static images such as the background are rendered this way. To render images in Pygame, you need to follow these steps:



We load the image file using pygame.image.load().



We scale the background image to match the dimensions of the window display using pygame.transform.scale(). I used the WIDTH and HEIGHT variable used to set the dimensions of the window display.



We display the image on the game window, using the blit() function of the display surface (WIN).

### 1.2 Attaching images to physical components

In this section, we will start from attaching static images to dynamic objects such as the ball and the paddle. Before moving onto attaching dynamic images to dynamic objects.

Static means non-changing and dynamic means changing

For Images:

Static images means the same image is being constantly displayed

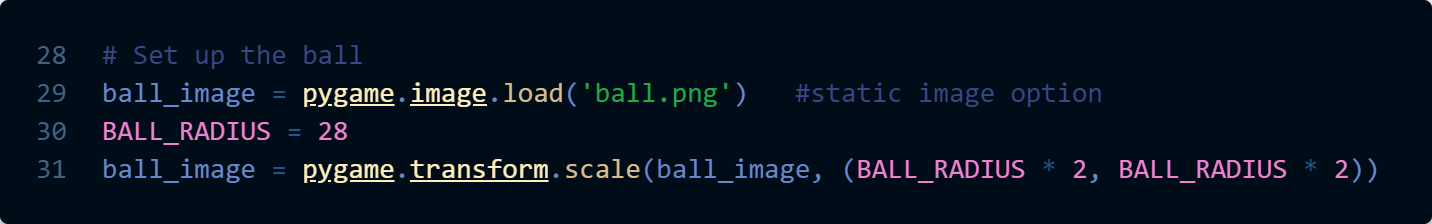
Dynamic images means images are being iterated through and changed

For Objects:

Static objects are objects that do not move

Dynamic objects are objects that have a change in size, shape, position etc.

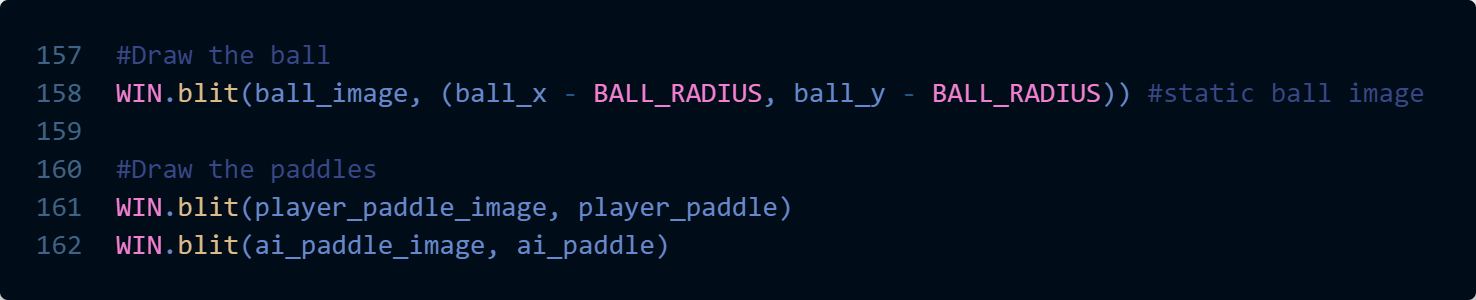
Ultimately, it is this dynamic property that defines animation





We load the images for the ball and the paddle. Ignoring lines 66-70. The images for both the ball and the paddle are scaled to be the size of the paddles that were previously drawn in BasepongAI.

So that when drawn, they match the size of the “actual” hitbox of the ball and paddle.



We display the image on the game window, using the blit() function to draw the images at locations where the “actual” ball and paddles are.

It's important to understand the following aspects regarding images used in the game:

**Aspect Ratio:** The ratio between the size of dimensions of an image and the container or frame that holds the image is commonly referred to as the aspect ratio.

When using images in the game, the scaling of the image is the scaling of the container/frame and not the image(content), typically resulting in the displayed images being smaller and not occupying the entire hitbox. This can lead to a discrepancy between what you see visually and the actual physics of the game, particularly when it comes to collisions or interactions with the image. The difference in size between the image and the hitbox can cause unexpected behaviour or inaccuracies.

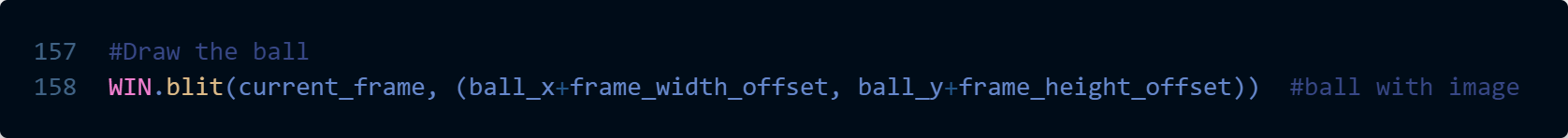
Fix: #1 Transform.Scale the image bigger

#2 Make the hitbox smaller, for the ball: that means decreasing the radius, for the paddle:making separate widths and heights between the displayed image and hitbox.

**Content Positioning:** The position of the content within the image file is crucial. If the image is not centered within the container, it can result in an off-center hitbox. This means that the hitbox used for collision detection or interaction may not align perfectly with the visual representation of the image. This misalignment can affect gameplay and lead to unintended outcomes.

Fix: Add offset to the displayed image position.

For example:



To summarize:

#1 The image within a PNG or JPG file may be smaller than the hitbox of the ball or paddle, causing a mismatch in size.

#2 The image may be off-center within the PNG or JPG file, leading to an off-center hitbox.

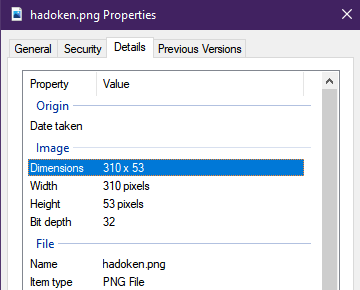
It's important to consider these factors when working with images in the game to ensure accurate and expected gameplay mechanics.

### 1.3. Cutting Sprite Sheet into Frames using Subsurface

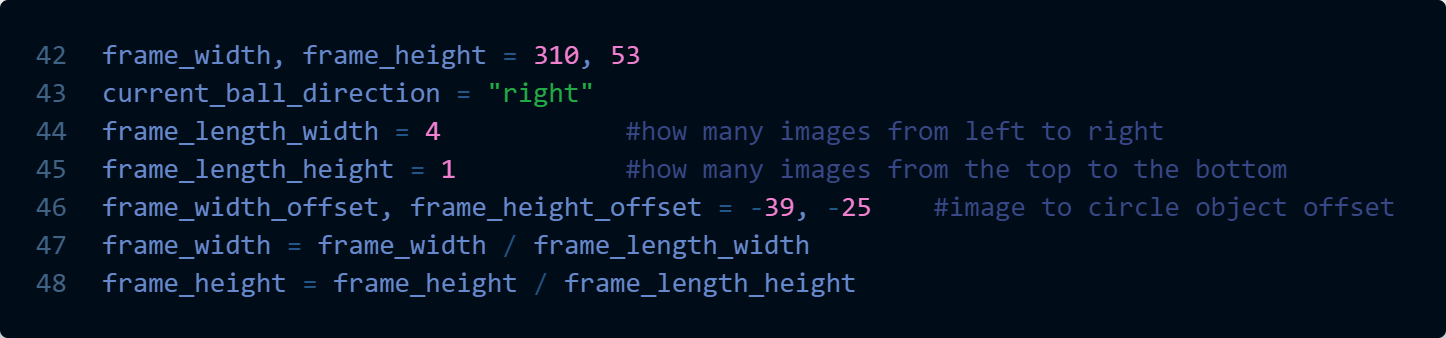
### To cut an image into frames using subsurface(), you can follow these steps:

### 

We load the sprite sheet image containing multiple frames.

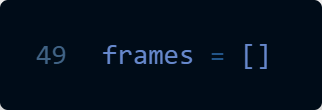


Determine the dimensions of each frame within the sprite sheet.



Adjust the dimensions based on how many frames there are in the image.

Adjust for Content Positioning, mentioned above in Section 1.2.



Create an empty list to hold the individual frames.

If you have a collection of images instead of a sprite sheet, you would still put them into the empty list “frames”. Or you can place images together in one file by using online tools, search for “sprite sheet generator”, this is especially useful if you draw or you want to conserve storage space.

### 

A nested loop is used to iterate through both the top and bottom frames. The outer loop controls the number of rows (top and bottom), and the inner loop iterates through the number of columns (left to right). The frame\_rect is updated to include the appropriate offset for both the width and height, based on the current row and column values. Subsurface() is used to extract each frame. By doing this, the code will generate a set of images that include frames from both the top and bottom as well as left to right of the original image.

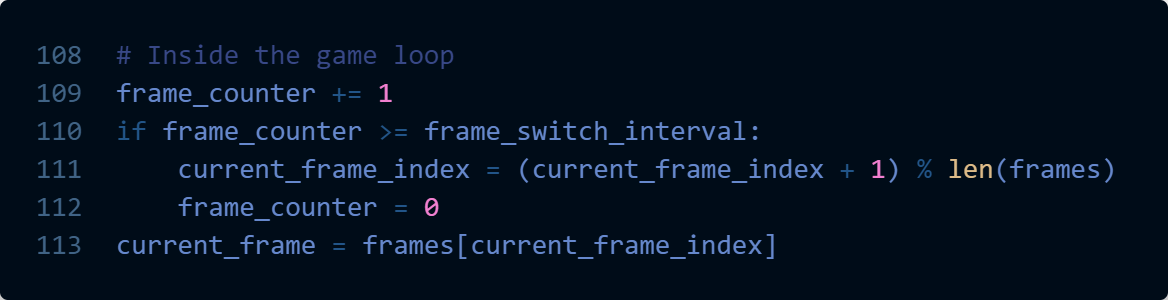
### 1.4 Controlling the frame rate of the displayed images

### 

FRAMES is synonymous with frames displayed per second(FPS).

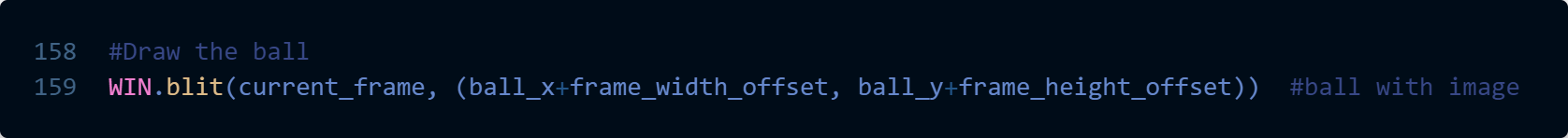
The FRAMES variable decides how many times the game loop is iterated through per second. Everything gets updated and replaced/redrawn during each loop. Hence an iteration of a loop is a frame, how many times a loop is executed determines the frames displayed/updated per second.

If I were to cycle through my animation of my ball\_animation, it would run at the speed of the game. My animation would run at/through 60 frames per second. I have 4 frames, each with drastic changes, so I have a frame\_switch\_interval to have it run slower, to make the animation smoother.



I want to iterate through 6 frames of my animation per second, so I have a frame\_switch\_interval to wait for 10(60/6) frames/loops of the game before moving onto the next frame. The frame\_counter hits 10, resets to 0 and starts recounting to 10.

%(modulo operator) takes the remainder of a number after a division.

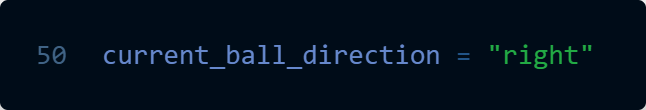


We control the frame to be displayed through WIN.blit(“current\_frame”)

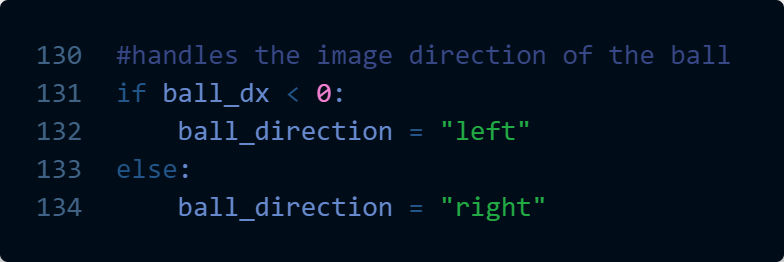
### 1.5. Transforming the Direction of Animated Frames

Changing the state of an object, and changing the corresponding animation. Typically, you would render a whole different picture cut into frames for other types of games.

To transform the direction of the animated frames, you can use the transform.flip() function.

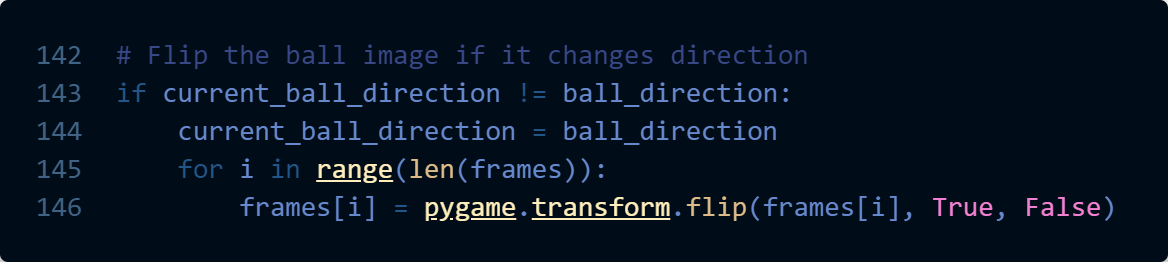


We create a variable to keep track of the current direction of the image the ball is facing towards.



We Implement the desired logic to switch the direction of the ball.

The direction the image of the ball is facing will change. A cleaner way to code this would be if dx<0, ball\_direction=”left”



Whenever the direction changes, iterate through the list of frames and flip each frame horizontally using transform.flip().

This will ensure that the frames are flipped horizontally when the ball changes its direction.

### 