**Day 1**

Using the information from <https://www.pygame.org/docs/> it helped me first understand how to use pygame. Using the example provided on the website it creates a screen with a purple background with a red circle that can be moved with the wasd keys.

A red circle on a purple background

Description automatically generated

While this is a helpful start the vison for my game is the background moves instead of the main character unless near the edge of the map. So, first thing I had to change was the background being an image rather than just a solid colour. Using the code pygame.image.load() I could temporarily store the image in a variable, then using screen.blit() I could update the background of the game to be the image I chose.

A video game screen shot of a house

Description automatically generated

The image above is the result of updating the image as the background, an issues arose from this, this being due to the image having the resolution 1000x1000 while the game screen had 1280x720 part of it was cut off while on the right side there’s black due to the image not being big enough.. To fix the issue I first tried using the code pygame.transform.scale() to zoom into the background image however while this did fix the issue of the size being too small it created unbearable amounts of lag that would ruin the users experience. So instead as a temporary fix I increased the resolution of the image, this fixed the issue making the image bigger without creating large amounts of lag.

A screenshot of a video game

Description automatically generated

**Day 2**

Main goal of today was to get the movement of the character/background the way I wanted it to work. This started by making the main character (the red circle) appear in the middle of the screen. This can be done by having y coordinate of the circle be half the screen height and the x coordinate be half of the screen length.

In the example used on the pygame website it uses the code keys = pygame.key.get\_pressed(), this checks if any keys are being pressed in that moment. Then the code also uses if keys[pygame.K\_w]:, this checks if the button mentioned is being pressed and if so it completes the code from the if statement. This is repeated 4 times each with different keys (w, a, s, d), this allows the circle to move as it changes the circles position depending on what is pressed. I removed the ability to move the circle and instead replaced it with an updated coordinate for the map to move to.



A screen shot of a computer program

Description automatically generated

I made it so if the character wants to move right, the background scrolls to the left. This gives it the illusion that the character is moving while in fact it’s only the background. While this is how I want the character to move the user is still able to walk outside of the area and creates a distorted background if done so.

**Day 3**

Next, I decided to make sure when the character reaches the edge of the map instead of the camera panning into nothingness the character moves in whichever direction the user intends to.

A black screen with white text

Description automatically generated

To do this I changed the code that moves the map when a key is pressed, this makes it check if the character is at the edge at the map, if so, it moves the character but if not, the map moves. While this worked it created two issues, first being while it worked for both moving up and to the left as the map coordinates for the edge would always be 0,0 no matter the background used it was different for down and right as different backgrounds may have different resolutions. And secondly the player position wouldn’t return to the middle if they tried to move the opposite direction.

To fix the first issue I searched up there was a way to get the pixel resolution for an image and store it in a variable, using the website <https://www.codingninjas.com/studio/library/get-the-width-and-height-of-the-image-in-pygame> it taught me get\_width() and get\_height() could be used to get the resolution.

So first I tried to if mapy == background.get\_height(): , however this didn’t work since the height was 4000 while the top of the map was 0. To fix this I had to first get the image height then multiply it by -1. This fixed the issue of the character never reaching the bottom edge however due to the maps position being based on the top left position of the character would still be able to go under the map. I fixed this by adding the height of the screen to the background height X -1.

A screen shot of a computer program

Description automatically generated

**Day 4**

To resolve the issue of the character not returning to the middle after moving position I changed the code to check the position of the character before moving the background.

A black screen with white text

Description automatically generated

This checks whether the player is under half the screen, if so the player will move back up to the half point and only then will start actually moving upwards.

**Day 5**

Using pygame.mixer.music.load() I can load a audio file and with pygame.mixer.music.play() I can play the audio file, this lets me create background music for the game.



The -1 argument in the .play code will indefinitely play the music on loop after it finishes.

**Day 6**

As I didn’t attend for the main character to stay as a red circle forever my next challenge was for the main character to be an image of a person. To start I created a value called character and used pygame.image.load to load a character image into it. Then i got rid of the pygame.draw.circle(screen, "red", player\_pos, 40) and reused the same method for the allowing the background to appear by using screen.blit(character, player\_pos). The allows the png stored in variable character to always follow the player positions (player\_pos) when moving.

A screenshot of a video game

Description automatically generated

While this felt like an easy solution walking around with the character made me realise how lifeless the game felt as the character remained the same while walking. This ruins the immersion of the game as making the character slide around instead of having different walking frames made the game look incomplete.

To create the illusion of walking I created a list which contained of images for each different walking animations. Then also created four variables called (right/left/up/down)\_current, this is used to choose the appropriate image within the list. Then to make the correct images appear when walking in the chosen direction when the key is pressed the image within the directed list is shown. To make the image change between the different images in the animation the direction\_current values are used to choose the correct image. This is done by first being at value 0 so it choses the first image in the list is chosen, then the current value is compared to the length of the list, so if its bigger the value goes back to 0 but if it isn’t the value is increased by 1. This allows it to go through each image and reset the animation once all images are done.

A computer screen shot of a program

Description automatically generated

However while walking right the idle picture of the character is still shown, this is due to it being in the while running: loop. However deleting screen.blit(character, player\_pos) from the running section causes the character to not appear while no buttons are pressed.

A drawing of a person

Description automatically generated

To fix this issue the program needs to check when no buttons are being pressed. To do this I created a new variable called no, this variable would only be changed if a button is pressed. This means while no buttons are being pressed I could use screen.blit(character, player\_pos) to show the character. This resolved the issue allowing only the correct animations to be shown.

A cartoon of a person standing in a bush

Description automatically generated 

**Day 7**

While last time I made animations for walking , the speed of the iamge changes wasn’t limited this means every frame the image would change. As the game runs at 60 frames per second the images would be changing way too fast.

To fix this I had to use import time, using this allows me to use code that is able to keep track of time within the program. I used this to create code that took the time when the image first changed then would subtract that time to the current time to see if enough time had past. If enough time had past the starting timer would reset.

A screen shot of a computer program

Description automatically generated

**Day 8**

One of the essential aspects of my RPG will be collision detection, as the game is played by the main character going to an interacting with objects and enemies in the world. To do this I had to first create something the character could interact with, so I created the first enemy.

A cartoon of a person and a dog

Description automatically generated

First enemy was a cat, to create this cat I had to create the class enemies, with the cat being an object of this class.

A computer screen shot of a program

Description automatically generated

The enemies class takes many variables whch are used to identify each enemy (the amount of variables is likely to increase later). The method appear allows for enemies to appear on the screen with them being at the location of “loc”.

When searching through <https://www.pygame.org/docs/> I was unable to find a way to detect a collision of two images, unfortunatly as I was unable to think of a way to solve this issue I searached it up. Using <https://stackoverflow.com/questions/59551594/how-to-detect-collision-of-two-images-in-pygame> I learnt that you are able to create an invicible rectangle the size of an image, then using .colliderect() it can check wether two images have collieded.

**A screen shot of a computer code

Description automatically generatedA black background with white text

Description automatically generated**

As shown I created a new method for enemies to create a rectangle for themselves, with the main character also creating a rectangle for themselves. This allows for .colliderect to see if theyre colliding (print yes was just used to check if it worked).

**Day 9**

After creating the enemy collision system I realised that it was unsless as the current way the map is coded is bad. The current map is just a picture of a background which moves depending on the main character. This mean that nothing in the map is currently able to be interacted with, my idea is to create a map by having it be two maps instead. The first one is the viewable image that the user will see, while underneath is a separate map that determines if anything at that spot can be interacted with. To make this I will separate the map int