Creating a Platform Game in Python

2024.05.01

For this project, I wanted to extend my knowledge of coding in Python. Since my last project was creating Tic Tac Toe, a simple game, I wanted something that involved more challenge towards me. Also, I tried Doodle Jump and it was fun I wondered if I could create this game in Python. So, I thought of a platform game, similar to Doodle Jump which is an adventure game where you jump from one platform to the next continuously while also avoiding obstacles (Doodle Games).

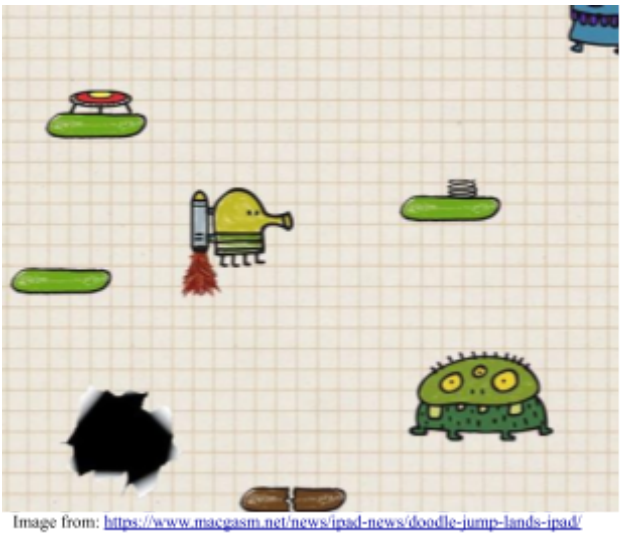
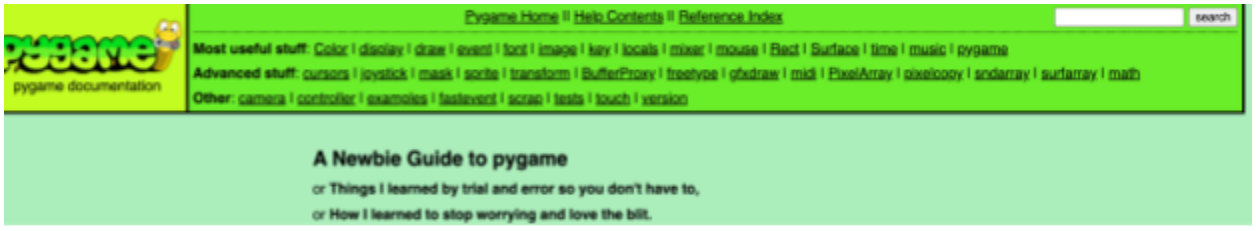


Image from: https://www.macgasm.net/news/ipad-news/doodle-jump-lands-ipad/

Instead of using Tkinter, I decided to use Pygame which is a set of Python modules designed for writing video games (Pygame). Making a platform game in Tkinter would mean that I would have to code more detail as the purpose of Tkinter is to create graphical user interfaces rather than handling what games would need like animation, collision detection, etc. Pygame however, is created to make games in Python more easier, already having functions that use a lot of codes into functions that can be used less complicatedly.



(Screenshot of https://www.pygame.org/docs/tut/newbieguide.html where it gives me many guides to learn the feature in pygame.)

Also, Pygame had many sources that gave a great introduction learning guide for those who haven't used Pygame before. By using this source, I was able to easily understand how to use Pygame and create my platform game.

How my game works:

The game I created has multiple stages, where a cat has to jump on different platforms to catch a mouse that remains still at the highest platform. In the beginning, I called it “Monkey Jump” because I wanted the character to be a monkey who jumps around different platforms to reach a banana, however, I liked the idea of the cat catching a mouse better. The twist is that the cat only has twenty seconds to reach the mouse and as the stage increases, the length and the number of platforms decreases, making it more challenging for the cat to reach the mouse.

The rules of this game consist of:

1. The game will have eight rounds in total so from round one to round eight
2. The platform amount will be 8, 8, 6, 6, 5, 5, 5, and 4 from round one to eight.
3. The platforms will be placed at random for each round.
4. The next round will begin as soon as the cat collects the mouse.
5. The cat will always start from the ground.
6. The platforms can be stood on however cannot go through it

For the player rules, the cat being the player:

1. The player has 20 seconds to reach the mouse for each round.
2. If the player does not get it in time, the game is over.
3. The player will use the arrow keys to move around the game.

Before creating the game, I decided on the screen size and the color theme. I decided to go with a 500px by 700px with a sunset vibe theme, using colors like purple, pink, yellow, and orange. For the character and background, I thought it would be a fun idea to create pixel art for the background after looking at many of them like Terraria and Hollow Knight.

Before going fully into the codes to create my game, I started with the basics since I didn’t know how to use Pygame properly. On the Pygame website, there was a Newbie Guide for those who have never used it before. The newbie guide explained its key concepts, being selective about advice, structuring code effectively, optimizing where necessary, and maintaining a continuous learning mindset. Something that gave me a good start to creating my code was giving me a base template.

(Screenshot of how the code above displays. Code from: https://www.pygame.org/docs/tut/newbieguide.html)

By giving me a base template of the code to start creating a game, I was able to understand how some of the codes in Pygame work. After looking at the beginner guide, I figured out there were more guides to other

features in the game so to expand my understanding of Pygame I looked at them. One example of what I found out is that I learned that I need to use a feature called Rects to create rectangle shapes around the window.

This is what would need to be included to draw a rectangle into the window:

```
rect1 = pygame.Rect(100, 100, 50, 50)  # x, y, width, height
pygame.draw.rect(screen, color , rect1) #Place the rect1 into the screen
```

After exploring the different features that can be used in codes for python, I first began sketching how different areas in the game would look like:

There are these three main pages that I decided to add to my game:

1. The first page is the page that people will look at the beginning of the game which I called it the start page. I decided to make it plain with a pink background with a title and a label that said play. Instead of making the label a button, where the user would have to hover over their mouse to start the game, I used my character to move around the screen and make the user use the arrow keys to get the cat on the play button and press enter to play the game.

2. The second page is where all the game happens. As soon as the user presses enter to start the game, the cat will be teleported to stage 1, where there will be eight floating platforms that the cat would have to jump around to reach the mouse on the highest point. The stages will continue if the user can reach the mouse on time, getting harder and harder as the amount of platforms and size decreases.

3. When the user either passes all eight stages or fails to reach the mouse on time, the third screen will appear saying that the game is over. Just like the first page, the user would have to make the cat hover over the label they selected and press enter. If the user decides to select “no,” then the game will entirely shut down. However, if the user selects “yes” it will go to the beginning of the stage resetting everything.

After thought processing everything that I wanted to add to the game, I began coding right away. Instead of taking my time and understanding some of the features in Pygame, I decided that I would learn during the process of creating the game.

I started with the code that Pygame gave from the beginner guide:

```
import pygame

pygame.init()

screen = pygame.display.set_mode((1280,720))

clock = pygame.time.Clock()

while True:
   # Process player inputs.
   for event in pygame.event.get():
       if event.type == pygame.QUIT:
           pygame.quit()
           raise SystemExit

   # Do logical updates here.
   # ...

   screen.fill("purple")  # Fill the display with a solid color

   # Render the graphics here.
   # ...

   pygame.display.flip()  # Refresh on-screen display
   clock.tick(60)         # wait until next frame (at 60 FPS)
```

From the given code, I changed the screen size to 500 by 700 pixels and the background color to light blue.

```
import pygame

pygame.init()

screen = pygame.display.set_mode((500,700))#changed screen size

clock = pygame.time.Clock()

while True:
   # Process player inputs.
   for event in pygame.event.get():
       if event.type == pygame.QUIT:
```

```
            pygame.quit()
            raise SystemExit

    # Do logical updates here.
    # ...

    screen.fill("Light Blue")  #changed background color

    # Render the graphics here.
    # ...

    pygame.display.flip()  # Refresh on-screen display
    clock.tick(60)         # wait until next frame (at 60 FPS)
```

Then, I decided to create my first page without its functions first.

```
import pygame

# pygame setup
pygame.init()
screen = pygame.display.set_mode((500, 700))
clock = pygame.time.Clock()
#create a rectangle that has the dimentions 150x100 positioned in (175, 300)
called start_1.
start_1 = pygame.Rect(175, 300, 150, 100)

def START_SCREEN():

    screen.fill("pink") #set the background to pink

    pygame.draw.rect(screen, "white", start_1) #draw the rectangle created to
the screen with the color white
    #add the label "CAT JUMP!" in black centered in the x value with the height
150
    if pygame.font:
        font = pygame.font.Font(None, 64)
        text = font.render("CAT JUMP!", True, (10, 10, 10))
        textpos = text.get_rect(centerx=screen.get_width() / 2, y=150)
        screen.blit(text, textpos)

    #add another label that saids "START" with light grey positioned within the
rect_1
    if pygame.font:
        font = pygame.font.Font(None, 64)
        text = font.render("START", True, "Light Grey")
        textpos = text.get_rect(centerx=screen.get_width() / 2, y=325)
        screen.blit(text, textpos)

#display the screen created in START_SCREEN()
```

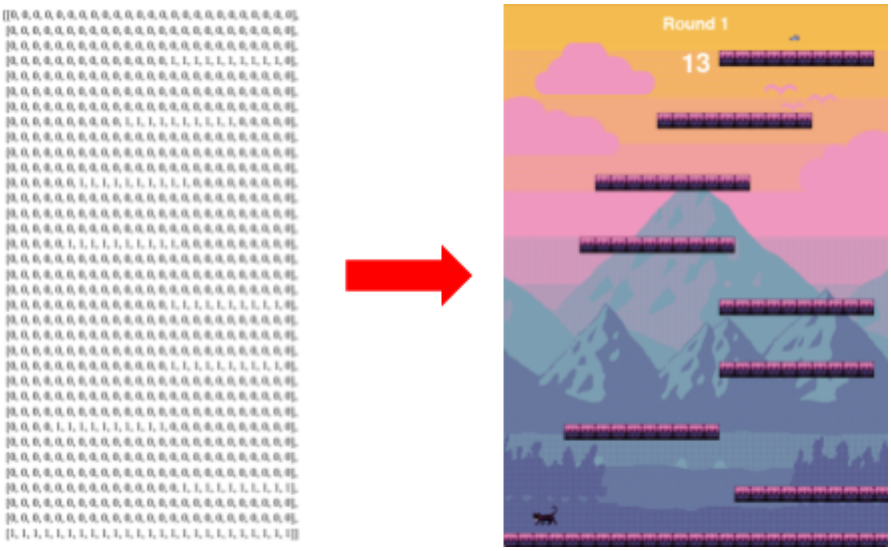
```
START_SCREEN()

while True:
  for event in pygame.event.get():
      if event.type == pygame.QUIT:
          pygame.quit()
          raise SystemExit

  pygame.display.flip()   # Refresh on-screen display
  clock.tick(60)          # wait until next frame (at 60 FPS)
```

This is the code for the beginning of the page, which looks like this:

After creating the first page, I continued to the second page. To create platforms in a game, I first began planning on the coordinates of where I wanted the platforms to be. Before mapping out the platforms, I first determined the size of each platform which was 20 x 20 pixels. Since the game has dimensions 500 by 700, 25 by 35 were going to be my dimensions for mapping my data. In my code, I created a variable called mapData which is where there will be the list making a list made up of zeros and ones. Zero means a blank space, and one means a sprite, which is an image used in games to represent visual elements (Viswanadham). The image below represents what mapping data looks like.



The function I created is called gameScene. It will be the area where it will load the platforms, labels, and backgrounds on the second page. This means that another function will have to be created to generate platforms for the game. Each platform is going to be one 10 by 10 sprite.

```
import random
import pygame
import time

# pygame setup
pygame.init()
screen = pygame.display.set_mode((500, 700)) # Create a window of size 500x700 pixels
clock = pygame.time.Clock() #A clock to control the frame rate
roundnumber = 1 #variable to store the current round number

def gameScene():
   global timer #making the timer global so the same variable can be used in other fucntions

   #Creating the visuals, text, etc
   screen.blit(background,(0,0))  # Draw the background image at position (0,0) --> the image is loaded in the code below
   Round_Number = pygame.Rect(0, 0, 500, 50) #Draw a new rectangle that represents a bar at the top
   pygame.draw.rect(screen,(255, 185, 50), Round_Number) #Write the round number on the center top

   #create a label at the top center "Round" to show the user what round they are at
   if pygame.font:
       font = pygame.font.Font(None, 32)
       text = font.render(("Round " + str(roundnumber)), True, (255, 255, 255))
       textpos = text.get_rect(centerx=screen.get_width() / 2, y=15)
       screen.blit(text, textpos)


   #Render map for mapData
   #cell size: 20 x 20
   #cell code: 1 -> plat
   #cell code: 0 -> air


   #for ever number in mapData --> if the number is 0, continue (leave it blank), else if the number is 1, insert a platform
   for i in range(35):
       for j in range(25):
           if mapData[i][j] == 0:
               continue
           elif mapData[i][j] == 1:
```

```
                screen.blit(pygame.transform.scale(tile['grass_plat'],(20,20)),
(20*j, 20*i))

    #place the mouse (that the user has to get) at the highest platform
    screen.blit(mouse_touch, pygame.Rect((goal[0] * 20 + 10, goal[1] * 20),
(20,20)))

    #Timer
    timer = 20 - int(time.time() - startTime)
    #inserting the time as a label while it is counting to zero
    if pygame.font:
        font = pygame.font.Font(None, 50)
        text = font.render(str(timer), True, (255, 255, 255))
        textpos = text.get_rect(centerx=screen.get_width() / 2, y=60)
        screen.blit(text, textpos)



mapData = [[0]*25 for _ in range(35)]  #Create a 2D array representing the map
platforms = []  # List to store platform rectangles

tile = dict() #Create a dictionary to store tile images
tile['grass_plat'] = pygame.image.load('./Tiles/gblock.png').convert_alpha()
#Load a grass platform tile image
mouse_touch = pygame.image.load('./Tiles/mouse.png') #Load a image for the
mouse
player_image = pygame.image.load('./Tiles/Character.png').convert_alpha() #Load
a image for the player (cat)
background = pygame.image.load('./Tiles/Wallpaper.png').convert_alpha() #Load a
image for the background


dt = 0.001 #time between each frame
onTheGround = False

timer = 20
startTime = time.time()

goal = pygame.Vector2(0,0)

gameScene() #render the gameScene


while True:
  for event in pygame.event.get():
      if event.type == pygame.QUIT:
          pygame.quit()
          raise SystemExit
```

```
  pygame.display.flip()   # Refresh on-screen display
  clock.tick(60)          # wait until next frame (at 60 FPS)
```

With the code above only, it will display:

With only the gameScene function, it is just a display of the background, round number, mouse, and timer. The platforms had to be coded separately in a different function. For every round, the number of platforms and the length of the platforms will be different. Instead of creating one function that changes the platform for every round, I created one function for each round because it was something I could do without struggling so much.

```
def stage1(): #Creating a function for round 1
   global mapData, platforms, goal, roundnumber # Making the mapData,
platforms, goal, and roundnumber global so they can be used in other functions

   #10 * 1 size, 10 platforms
   ##25 x 35 cells

   platforms.clear() # Clear the list of platforms

   #creating platforms
   for i in range(8): #Make eight platforms
       rx = random.randint(0,25-10)# Select the vertical position of the
platform randomly
       ry = (35-3) // 8 * i + 3 # Select the certain horizontal position of the
platform so they are evenly spaced out
       if i == 0:
           goal = pygame.Vector2(rx+4,ry-1) #Select the highest platform that
the mouse will be placed at
       platforms.append(pygame.Rect((rx * 20, ry * 20), (20 * 10, 20))) #Create
a platform rectangle and add it to the list
```

```
        for i in range(10): #Create a platform with the length being 10
            mapData[ry][rx + i] = 1 #Label the platforms with 1s in the mapData
    x, y = 0, 34 #Initialize x and y coordinates for the ground
    platforms.append(pygame.Rect(((0, 680)), (500, 20))) # Add a ground platform
    for i in range(25): #Create the platform length for the ground
        mapData[y][x + i] = 1 #Label the platforms with 1s in the mapData
```

The code above is the location of where the platform will be placed. It is written before the gameScene so that the mapData can receive the information to place the platforms. Since the function is labeled as Stage 1, the platforms will be in the game's first stage. The x coordinates of the platforms were made random however, the y coordinates were spaced out evenly because if the platforms are randomly chosen and decided to combine all the platforms, there will be no space for the user to jump and reach the mouse. Also, there is a possibility that some of the platforms are difficult to reach, making it impossible to beat the game. Since I have chosen to make the game challenging if the user beats a round by shortening the platform and decreasing the number of platforms, the yellow highlighted areas are placed where the number decreases every round.

(Screenshot of the different rounds in the platform game I created. Comparison of the two rounds)

The image above shows the difference of the two rounds. Since I have set the amount of platform the same as round 1 for round 2, it doesn't change however, the length of the platform shortens from 10 to 8.

After creating the map, I continued with the movement of the actual character. From the beginner guide, another code that it provides people who are new to Pygame is a code that moves a circle around by using the key, WASD.

```
# Example file showing a circle moving on screen
import pygame
```

```
# pygame setup
pygame.init()
screen = pygame.display.set_mode((1280, 720))
clock = pygame.time.Clock()
running = True
dt = 0

player_pos = pygame.Vector2(screen.get_width() / 2, screen.get_height() / 2)

while running:
    # poll for events
    # pygame.QUIT event means the user clicked X to close your window
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # fill the screen with a color to wipe away anything from last frame
    screen.fill("purple")

    pygame.draw.circle(screen, "red", player_pos, 40)

    keys = pygame.key.get_pressed()
    if keys[pygame.K_w]:
        player_pos.y -= 300 * dt
    if keys[pygame.K_s]:
        player_pos.y += 300 * dt
    if keys[pygame.K_a]:
        player_pos.x -= 300 * dt
    if keys[pygame.K_d]:
        player_pos.x += 300 * dt

    # flip() the display to put your work on screen
    pygame.display.flip()

    # limits FPS to 60
    # dt is delta time in seconds since last frame, used for framerate-
    # independent physics.
    dt = clock.tick(60) / 1000

pygame.quit()
```

Using this code and incorporating some basic physics, I was able to create a cat character that jumps around the game environment. The first step was to replace the red circle with a sprite representing the cat. To achieve this, I created a sprite variable that holds the PNG file of the cat image. Instead of using the pygame.draw.circle function to draw the circle and combine it with player_pos, I used the blit function to draw the sprite on the screen. This function allows for drawing images in Pygame, effectively placing the cat sprite into a rectangle that can be moved around using player_pos.

Additionally, I changed the movement controls from the WASD keys to the arrow keys. This involved updating the key detection section of the code to listen for arrow key presses (pygame.K_UP, pygame.K_DOWN, pygame.K_LEFT, and pygame.K_RIGHT) instead of the WASD keys. This adjustment was made because arrow keys are a common choice for movement in many games.

Next, I implemented basic physics to handle jumping and gravity. This required adding variables for the cat's velocity and acceleration. By adjusting these variables, I could simulate gravity pulling the cat down and the cat jumping when the player pressed the jump key. The image below shows a flow chart of how the jumping function works in the game.

(Flow chart diagram of how the jumping function works in the game)

In the image above, there is a drawing demonstration of how the jump works. The yVelo is the upward force of the cat. So when the player presses the upwards arrow key, yVelo which is 0 becomes -400 boosting the cat to jump up, as it goes upwards the yVelo loses its boost exponentially, causing it to boost at a quick rate at the beginning and reach its maximum slowly which is when yVelo becomes 0 then, since there is no more boost, it continues to increase making it create a curve as the character goes down to a platform. Now the jump feature was made, however, there was another problem that occurred: the cat was able to keep jumping in the air, making it an advantage for them not to use the platforms. The variable OnTheGround was made to prevent this advantage, so the cat wouldn't allow the player to jump when the cat wasn’t on the ground.

The code of how the player moved around:

```
    else:
```

```
        # Call the gameScene function (context not provided, assuming it
sets up or updates the game scene).
        gameScene()

        # Draw the player image on the screen at the player's current
position.
        screen.blit(player_image, pygame.Rect(player_pos, (30, 30)))

        # Get the current state of all keyboard keys.
        keys = pygame.key.get_pressed()

        # If the UP key is pressed and the player is on the ground, make
the player jump.
        if keys[pygame.K_UP] and onTheGround:
            yVelo = -400  # Set the vertical velocity for the jump.
            player_pos.y += yVelo * dt  # Update the player's y position
based on the jump velocity and delta time.
            onTheGround = False  # Set onTheGround to False because the
player is now in the air.

        # If the LEFT key is pressed, move the player to the left.
        if keys[pygame.K_LEFT]:
            player_pos.x -= 300 * dt  # Move the player left by a speed
of 300 pixels per second, scaled by delta time.
            if collisionCheck():  # Check for collisions after moving
left.
                player_pos.x += 300 * dt  # If a collision is detected,
move the player back to the right.

        # If the RIGHT key is pressed, move the player to the right.
        if keys[pygame.K_RIGHT]:
            player_pos.x += 300 * dt  # Move the player right by a speed
of 300 pixels per second, scaled by delta time.
            if collisionCheck():  # Check for collisions after moving
right.
                player_pos.x -= 300 * dt  # If a collision is detected,
move the player back to the left.

        # Apply gravity to the player.

        # Increase the vertical velocity by the acceleration due to
gravity, scaled by delta time.
        yVelo += yAcc * dt
        # Update the player's y position based on the current vertical
velocity and delta time.
        player_pos.y += yVelo * dt
```

```
        if collisionCheck():  # Check for collisions after applying
gravity.
            player_pos.y -= yVelo * dt  # If a collision is detected,
move the player back to the previous y position.

            if yVelo > 0:  # If the player was falling (positive y
velocity),
                onTheGround = True  # Set onTheGround to True because the
player has landed.
            yVelo = 0  # Reset the vertical velocity to 0 because the
player is now on the ground.
```

The code above displays how the character can move around the screen. I also included collision detection to ensure the cat could land on platforms and not fall through them. This was done by checking if the cat's rectangle intersects with any platform rectangles. For the platforms, I applied a code function on them called “colliderect.”

```
    def collisionCheck():
        for plat in platforms:
            #when the platform is against the player
            if plat.colliderect(pygame.Rect(player_pos,(20,40))):
                #say that the player is touching a platform
                return True

        #outside check... (to prevent it from getting out of frame)
        if player_pos[0] <= -5 or player_pos[0] >= 480:
            return True

        return False
```

As shown in the code above, it checks the relationship between the player and the platform. This is used so that if a collision is detected, for example when the cat is standing on the platform, it forces the same force on the cat preventing it from going through the platform. So when the player moves to the left and attempts to go to the area where it can’t be seen, the wall applies the same amount of force to the right, forcing the cat on the screen. So now the character can jump around different platforms without going through them. However, the mouse that is placed at the center of the highest platform does not have the function applied; the character can go right through the mouse. I have done this on purpose so that it makes the effect of the cat look like they have caught the mouse and eaten it.

After creating the map of the game for each stage and the character movement, it was now time to combine the two codes so that the game would work. During the combinations, I created a 20-second timer that would occur in each round, I made sure that the variable that held the time number would always be 20 at the beginning of the game, losing each second as time passed by. When the timer reaches 0 seconds and the player doesn’t get the cat to the mouse, the game will be over, and ask the player if they want to do it again or not. If they have said no, the game will automatically shut down the player, if they say yes, the game will go to the front page again, restarting.

In the running section, where the functions are input in that area the visual elements will show starting with the function, Gamestart, and after that, the different functions will continue, depending on the player's choices.

After creating the platform game, there are a few things that I can do to make this game a more fun and interesting experience:

1. The graphics of the game.

I thought that the artistic features of the game were great in the area where the gameplay happens, however, I think that I can create better graphics for the beginning of the page and the end of the page. The background and the sprite were drawn using the art style pixel art and I used this because I thought game-wise it is a very aesthetically pleasing style to use for a game as many other games use this feature as well. I think that the start of the page and the end of the page is a bit boring so I think if I were to continue on this project, I would work on the design of how the games more to make it more fun to play.

2. Learning more about Pygame.

For this game, instead of using Tkinter, I used Pygame, and since it was something new for me because I had never heard about the game module before, I had a hard time adjusting to how it worked. If I wanted to work more on this project, I would stop trying to add more and try delving into the features used in Pygame first. If I had done this in the beginning, I would have understood how to make a game more and would create more advanced features in the game such as moving platforms and obstacles.

Overall, this project was a very fun learning adventure because I was able to learn something new about how to use Pygame to create games. Even though it was challenging for me because some of the coding parts were a little bit complicated, I was able to learn more and expand on my Python coding skills. I also was interested in how platform games were created for a long time and this process has made me understand how hard it is to program and create a game; so I look up to the game creators that make amazing games online for people to play and entertain themselves with.

Works Cited

Doodle Games. “Doodle Jump.” *Doodle Games*, https://doodlegames.io/doodle-jump. Accessed 1 May 2024.

Oxford. “Oxford Languages and Google - English | Oxford Languages.” *Oxford Languages*, https://languages.oup.com/google-dictionary-en/. Accessed 22 May 2024.

Pygame. “About - pygame wiki.” *Pygame*, https://www.pygame.org/wiki/about. Accessed 1 May 2024.

Viswanadham, Vamsi. “Pygame - blit() function.” *Naukri.com*, 26 March 2024, https://www.naukri.com/code360/library/pygame---blit-function. Accessed 9 May 2024.