

Python Platformer Tutorial Using Pygame

This tutorial shows you how to build a basic side-scrolling platformer game using Pygame. You'll learn how to implement player movement, gravity, collision detection, collectible items, and camera scrolling.

Project Setup

Step 1: Install Pygame

Install Pygame using pip if you haven't already:

```
pip install pygame
```

Step 2: Folder Structure

Your project folder should look like this:

```
platformer/
├── platform1.py
├── assets/
│   ├── player.png      # the player character
│   ├── tile.png        # blocks the player walks on
│   └── coin.png         # collectible items
```

All image files should be PNG format with transparent backgrounds and roughly 64×64 pixels in size.

Initialize Pygame and Set Up the Screen

```
import pygame
import sys
import os
pygame.init()
WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Pygame Platformer")
clock = pygame.time.Clock()
FPS = 60
```

pygame.init() initializes all Pygame modules.

WIDTH and **HEIGHT** define the window resolution.

pygame.display.set_mode() creates the actual window.

pygame.display.set_caption() sets the window title.

clock and **FPS** control how fast the game loop runs (here: 60 frames per second).

Load Images and Define Tile Size

```
ASSETS = "assets"
TILE_SIZE = 64

player_img = pygame.image.load(os.path.join(ASSETS,
"player.png")).convert_alpha()
```

```
tile_img = pygame.image.load(os.path.join(ASSETS,
"tile.png")).convert_alpha()
coin_img = pygame.image.load(os.path.join(ASSETS,
"coin.png")).convert_alpha()
```

TILE_SIZE defines how big each tile is (64 pixels).

We load images using `pygame.image.load()` and convert them to include transparency with `convert_alpha()` for better performance.

Define the Level Layout

```
level = [
    "                                ",
    "                                ",
    "                                ",
    "                                ",
    "          C                    C",
    "      TTTT          TT      C",
    "                                ",
    "  C          C                    G",
    "TTTTT  TTTTTT  TTTT  TTTT"
]
```

The level is a 2D grid made of characters:

'T' = tile/platform

'C' = coin

' ' = empty space

'G' = goal (not yet implemented)

Each row is a line of tiles, starting from the top of the screen down.

Player Class (Movement, Gravity, Collision)

```
class Player(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = player_img
        self.rect = self.image.get_rect(topleft=(x, y))
        self.vel = pygame.Vector2(0, 0)
        self.on_ground = False
        self.score = 0
```

We define a `Player` class using Pygame's Sprite system, which makes it easy to update, draw, and manage game objects.

self.vel is a 2D vector representing the player's speed in both directions.

self.on_ground tracks whether the player can jump.

self.score keeps count of collected coins.

Player Update Method

Resets horizontal speed to zero each frame. Moves left/right with arrow keys. Only allows jumping if the player is on the ground.

```
def update(self, tiles, coins):
    keys = pygame.key.get_pressed()
    self.vel.x = 0
    if keys[pygame.K_LEFT]:
        self.vel.x = -5
    if keys[pygame.K_RIGHT]:
        self.vel.x = 5
    if keys[pygame.K_SPACE] and self.on_ground:
        self.vel.y = -18
```

Gravity increases downward velocity. Movement is split into two steps (horizontal then vertical) for clean collision handling.

```
self.vel.y += 0.8    # gravity
if self.vel.y > 20:
    self.vel.y = 20  # terminal fall speed

self.rect.x += self.vel.x
self.check_collision(tiles, 'horizontal')

self.rect.y += self.vel.y
self.on_ground = False
self.check_collision(tiles, 'vertical')
```

Collision Checking

```
def check_collision(self, tiles, direction):
    for tile in tiles:
        if self.rect.colliderect(tile.rect):
            if direction == 'horizontal':
                if self.vel.x > 0:
                    self.rect.right = tile.rect.left
                elif self.vel.x < 0:
                    self.rect.left = tile.rect.right
            elif direction == 'vertical':
                if self.vel.y > 0:
                    self.rect.bottom = tile.rect.top
                    self.on_ground = True
                    self.vel.y = 0
                elif self.vel.y < 0:
                    self.rect.top = tile.rect.bottom
                    self.vel.y = 0
```

Uses axis-aligned bounding box (AABB) collision. Correctly adjusts the player's position if they bump into a tile.

Coin Collection

Removes coins from the list when collected and increases the score.

```

for coin in coins:
    if self.rect.colliderect(coin.rect):
        coins.remove(coin)
        self.score += 1

```

Tile and Coin Classes

Both use Pygame's Sprite class to store position and image. Tile and Coin don't need update logic; they're just static objects.

```

class Tile(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = tile_img
        self.rect = self.image.get_rect(topleft=(x, y))

class Coin(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = coin_img
        self.rect = self.image.get_rect(topleft=(x, y))

```

Build the World

```

tiles = pygame.sprite.Group()
coins = pygame.sprite.Group()
player = Player(100, HEIGHT - 150)
player_group = pygame.sprite.GroupSingle(player)

# Build the world from the level layout
for y, row in enumerate(level):
    for x, cell in enumerate(row):
        if cell == "T":
            tiles.add(Tile(x * TILE_SIZE, y * TILE_SIZE))
        elif cell == "C":
            coins.add(Coin(x * TILE_SIZE + 16, y * TILE_SIZE + 16))

```

Multiplies x and y by TILE_SIZE to convert from grid coordinates to pixel coordinates.

Slight offset on coins centers them better visually.

Camera Scrolling and Game Loop

```

scroll_x = 0
font = pygame.font.SysFont(None, 36)

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

```

Basic event loop to close the game when the window is closed.

Update Logic

Scrolls the world based on the player's position. Negative **scroll_x** moves the world left as the player moves right.

```
player.update(tiles, coins)
scroll_x = -player.rect.centerx + WIDTH // 2
```

Drawing the Scene

```
screen.fill((135, 206, 235)) # sky blue background

for tile in tiles:
    screen.blit(tile.image, (tile.rect.x + scroll_x,
                             tile.rect.y))
for coin in coins:
    screen.blit(coin.image, (coin.rect.x + scroll_x,
                              coin.rect.y))
screen.blit(player.image, (player.rect.x + scroll_x,
                           player.rect.y))

# Draw HUD
text = font.render(f"Coins: {player.score}", True, (0, 0, 0))
screen.blit(text, (20, 20))

pygame.display.flip()
clock.tick(FPS)
```

Each object is drawn with the **scroll_x** offset to create the scrolling effect. **pygame.display.flip()** updates the entire screen. **clock.tick(FPS)** enforces a maximum of 60 frames per second.