1. **Auto-scrolling world**
    - The player doesn't move forward; instead, the level scrolls leftward.
    - Implementation: Move obstacles/platforms leftward each frame at a constant speed.
2. **Jump physics**
    - Single button jump (space/arrow key).
    - Gravity pulls the player down continuously.
    - Jump is a velocity impulse upward.
3. **Collision detection**
    - Player collides with obstacles (spikes, blocks).
    - If collision occurs → game over.
4. **Level design**
    - Obstacles are spawned at fixed intervals or loaded from a level file.
    - Could be randomized for endless mode.
5. **Restart mechanic**

- On death, reset player position and obstacles.

```
import pygame, sys

pygame.init()

# Screen setup
WIDTH, HEIGHT = 800, 400
screen = pygame.display.set_mode((WIDTH, HEIGHT))
clock = pygame.time.Clock()

# Player setup
player = pygame.Rect(100, HEIGHT-50, 40, 40)
gravity = 0
jump_strength = -12

# Obstacles
obstacles = [pygame.Rect(600, HEIGHT-50, 40, 40)]
speed = 5

def reset_game():
    global obstacles, player, gravity
    player.y = HEIGHT-50
    gravity = 0
    obstacles = [pygame.Rect(600, HEIGHT-50, 40, 40)]

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE and player.bottom >= HEIGHT:
                gravity = jump_strength

    # Gravity
    gravity += 0.5
    player.y += gravity
    if player.bottom >= HEIGHT:
        player.bottom = HEIGHT
        gravity = 0

    # Move obstacles
    for obs in obstacles:
        obs.x -= speed
        if obs.right < 0:
            obstacles.remove(obs)
            obstacles.append(pygame.Rect(WIDTH, HEIGHT-50, 40, 40))

    # Collision
    for obs in obstacles:
        if player.colliderect(obs):
            reset_game()

    # Draw
    screen.fill((30,30,30))
    pygame.draw.rect(screen, (0,200,255), player)
    for obs in obstacles:
        pygame.draw.rect(screen, (200,50,50), obs)
    pygame.display.flip()
    clock.tick(60)
```

- **Spike obstacles**: Draw triangles instead of rectangles. **Multiple obstacle types**: Store obstacle type and render accordingly.
- **Level scripting**: Load obstacle positions from a file (JSON or text).
- **Music sync**: Tie obstacle timing to beats for Geometry Dash–style rhythm.

👉 Since you've already been experimenting with scrolling mechanics, you could extend this by adding **parallax backgrounds** or **different jump states (like double jump or gravity flip)**.

Would you like me to show you how to implement **gravity flip (upside-down mode)** next? That's one of Geometry Dash's signature mechanics.