Build a simple Geometry Dash–style game in Python

Geometry Dash taps that part of you that loves rhythm, precision, and the rush of a perfect run. Want to make your own version in Python? Here's a clean, minimal Pygame starter you can riff on.

Prerequisites

- **Python:** 3.9+ recommended
- **Pygame:** pip install pygame
- **Mindset:** Keep mechanics tight and visual feedback immediate

Core mechanics and starter code

- **Player:** A square with gravity and coyote-time jump
- **Obstacles:** Scrolling spikes; instant reset on collision
- **Camera:** Fake it by moving obstacles left
- **Difficulty:** Scales with obstacle speed and spawn rate

```
import pygame, sys, random WIDTH, HEIGHT = 800, 450 GROUND_Y = 360 BG_COLOR =
(20, 24, 31) PLAYER_COLOR = (80, 200, 120) SPIKE_COLOR = (240, 80, 80) pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT)) clock = pygame.time.Clock() font =
pygame.font.SysFont(None, 36) # Player state player = pygame.Rect(120, GROUND_Y - 40,
40, 40) vel_y = 0.0 gravity = 0.7 jump_strength = -12.0 on_ground = True coyote_frames = 6 #
small grace to jump right after leaving ground coyote_timer = 0 # Game state spikes = []
scroll_speed = 6 spawn_cooldown = 0 score = 0 alive = True def reset(): global player, vel_y,
on_ground, coyote_timer, spikes, scroll_speed, spawn_cooldown, score, alive
player.update(120, GROUND_Y - 40, 40, 40) vel_y = 0.0 on_ground = True coyote_timer = 0
spikes = [] scroll_speed = 6 spawn_cooldown = 0 score = 0 alive = True def spawn_spike(): #
Spikes as isosceles triangles on the ground base_w = random.choice([40, 50, 60]) height =
base_w x = WIDTH + random.randint(0, 100) # Represent spike by its triangle points points =
[(x, GROUND_Y), (x + base_w, GROUND_Y), (x + base_w // 2, GROUND_Y - height)] return
points def move_spikes(spikes_list, dx): return [[(px - dx, py) for (px, py) in tri] for tri in
spikes_list] def draw_ground(): pygame.draw.rect(screen, (35, 40, 52), (0, GROUND_Y, WIDTH,
HEIGHT - GROUND_Y)) def draw_spike(tri): pygame.draw.polygon(screen, SPIKE_COLOR, tri)
def tri_to_rect(tri): xs = [p[0] for p in tri] ys = [p[1] for p in tri] return pygame.Rect(min(xs),
min(ys), max(xs) - min(xs), max(ys) - min(ys)) def rect_triangle_collision(rect, tri): #
Broad-phase: AABB vs triangle bbox r = tri_to_rect(tri) if not rect.colliderect(r): return False #
Narrow-phase: check if any rect corner is inside triangle def sign(p1, p2, p3): return (p1[0] -
p3[0]) * (p2[1] - p3[1]) - (p2[0] - p3[0]) * (p1[1] - p3[1]) def point_in_tri(pt, a, b, c): d1 = sign(pt, a,
b) d2 = sign(pt, b, c) d3 = sign(pt, c, a) has_neg = (d1 < 0) or (d2 < 0) or (d3 < 0) has_pos = (d1
> 0) or (d2 > 0) or (d3 > 0) return not (has_neg and has_pos) corners = [ (rect.left, rect.top),
(rect.right, rect.top), (rect.left, rect.bottom), (rect.right, rect.bottom), ] a, b, c = tri[0], tri[1], tri[2] if
any(point_in_tri(pt, a, b, c) for pt in corners): return True # Edge overlap: simple check by
```

```
sampling points along rect bottom edge for x in range(rect.left, rect.right, 6): if point_in_tri((x,
rect.bottom), a, b, c): return True return False reset() 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_r: reset() if alive and (event.key ==
pygame.K_SPACE or event.key == pygame.K_UP): if on_ground or coyote_timer > 0: vel_y =
jump_strength on_ground = False coyote_timer = 0 # Physics if alive: vel_y += gravity player.y
+= int(vel_y) if player.bottom >= GROUND_Y: player.bottom = GROUND_Y vel_y = 0 if not
on_ground: on_ground = True coyote_timer = coyote_frames else: on_ground = False
coyote_timer = max(0, coyote_timer - 1) # Spawning spawn_cooldown -= 1 if spawn_cooldown
<= 0: spikes.append(spawn_spike()) spawn_cooldown = random.randint(30, 70) # Move spikes
spikes = move_spikes(spikes, scroll_speed) # Despawn off-screen spikes = [tri for tri in spikes if
tri_to_rect(tri).right > -5] # Collision for tri in spikes: if rect_triangle_collision(player, tri): alive =
False break # Score and difficulty score += 1 if score % 600 == 0: scroll_speed += 0.7 # Draw
screen.fill(BG_COLOR) draw_ground() for tri in spikes: draw_spike(tri)
pygame.draw.rect(screen, PLAYER_COLOR, player, border_radius=6) # UI s_text =
font.render(f"Score: {score}", True, (220, 220, 220)) tip_text = font.render("Space/Up to jump • R
to reset", True, (160, 160, 180)) status_text = font.render("Paused/Dead" if not alive else "",
True, (255, 120, 120)) screen.blit(s_text, (20, 20)) screen.blit(tip_text, (20, 54)) if not alive:
screen.blit(status_text, (20, 88)) pygame.display.flip() clock.tick(60)
```

Level design and polish ideas

- **Tile editor:** Define ground, spikes, orbs, portals in a CSV or a Tiled map. Parse rows into entities at runtime for repeatable patterns and rhythm-aligned layouts. A Python remake called “Pydash” uses Pygame, includes two levels made in Tiled, and demonstrates orb interactions and jump timing.
- **Juice:** Add camera shake on death, particle sparks on jump, and beat-synced color pulses.
- **Audio:** Use short percussive cues for jump/collect, loop background music at 120–150 BPM for tight rhythm feel.

Open-source examples to study

- **Pydash (Pygame implementation):** Geometry Dash–style mechanics, orbs, and Tiled-based levels in Python.
- **NumWorks calculator ports:** Lightweight Geometry Dash–inspired scripts showing minimal loops and collision/score HUD on constrained hardware.
- **Another Pygame recreation:** A small project sketching the structure for graphics, sound, and logic in Python.

Sources:

Next steps

- **Tell me your vibe:** Classic square-only, or do you want ship/ball portals?

- **Difficulty curve:** Prefer spikes in sync with music or chaotic bursts?
- **Target platform:** Desktop only, or do you want a web build via Pyodide/pygbag?

```
import pygame, sys, random

WIDTH, HEIGHT = 800, 450
GROUND_Y = 360
BG_COLOR = (20, 24, 31)
PLAYER_COLOR = (80, 200, 120)
SPIKE_COLOR = (240, 80, 80)

pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
clock = pygame.time.Clock()
font = pygame.font.SysFont(None, 36)

# Player state
player = pygame.Rect(120, GROUND_Y - 40, 40, 40)
vel_y = 0.0
gravity = 0.7
jump_strength = -12.0
on_ground = True
coyote_frames = 6  # small grace to jump right after leaving ground
coyote_timer = 0

# Game state
spikes = []
scroll_speed = 6
spawn_cooldown = 0
score = 0
alive = True

def reset():
    global player, vel_y, on_ground, coyote_timer, spikes, scroll_speed, spawn_cooldown, score,
alive
    player.update(120, GROUND_Y - 40, 40, 40)
    vel_y = 0.0
    on_ground = True
    coyote_timer = 0
    spikes = []
    scroll_speed = 6
    spawn_cooldown = 0
    score = 0
    alive = True
```

```
def spawn_spike():
    # Spikes as isosceles triangles on the ground
    base_w = random.choice([40, 50, 60])
    height = base_w
    x = WIDTH + random.randint(0, 100)
    # Represent spike by its triangle points
    points = [(x, GROUND_Y), (x + base_w, GROUND_Y), (x + base_w // 2, GROUND_Y -
height)]
    return points

def move_spikes(spikes_list, dx):
    return [[(px - dx, py) for (px, py) in tri] for tri in spikes_list]

def draw_ground():
    pygame.draw.rect(screen, (35, 40, 52), (0, GROUND_Y, WIDTH, HEIGHT - GROUND_Y))

def draw_spike(tri):
    pygame.draw.polygon(screen, SPIKE_COLOR, tri)

def tri_to_rect(tri):
    xs = [p[0] for p in tri]
    ys = [p[1] for p in tri]
    return pygame.Rect(min(xs), min(ys), max(xs) - min(xs), max(ys) - min(ys))

def rect_triangle_collision(rect, tri):
    # Broad-phase: AABB vs triangle bbox
    r = tri_to_rect(tri)
    if not rect.colliderect(r):
        return False
    # Narrow-phase: check if any rect corner is inside triangle
    def sign(p1, p2, p3):
        return (p1[0] - p3[0]) * (p2[1] - p3[1]) - (p2[0] - p3[0]) * (p1[1] - p3[1])
    def point_in_tri(pt, a, b, c):
        d1 = sign(pt, a, b)
        d2 = sign(pt, b, c)
        d3 = sign(pt, c, a)
        has_neg = (d1 < 0) or (d2 < 0) or (d3 < 0)
        has_pos = (d1 > 0) or (d2 > 0) or (d3 > 0)
        return not (has_neg and has_pos)

    corners = [
        (rect.left, rect.top),
        (rect.right, rect.top),
        (rect.left, rect.bottom),
        (rect.right, rect.bottom),
    ]
    a, b, c = tri[0], tri[1], tri[2]
    if any(point_in_tri(pt, a, b, c) for pt in corners):
        return True
    # Edge overlap: simple check by sampling points along rect bottom edge
    for x in range(rect.left, rect.right, 6):
        if point_in_tri((x, rect.bottom), a, b, c):
            return True
    return False

reset()
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_r:
                reset()
            if alive and (event.key == pygame.K_SPACE or event.key == pygame.K_UP):
                if on_ground or coyote_timer > 0:
                    vel_y = jump_strength
                    on_ground = False
                    coyote_timer = 0

    # Physics
    if alive:
        vel_y += gravity
        player.y += int(vel_y)

        if player.bottom >= GROUND_Y:
            player.bottom = GROUND_Y
            vel_y = 0
            if not on_ground:
                on_ground = True
            coyote_timer = coyote_frames
        else:
            on_ground = False
            coyote_timer = max(0, coyote_timer - 1)

        # Spawning
        spawn_cooldown -= 1
        if spawn_cooldown <= 0:
            spikes.append(spawn_spike())
            spawn_cooldown = random.randint(30, 70)
        # Move spikes
        spikes = move_spikes(spikes, scroll_speed)
        # Despawn off-screen
        spikes = [tri for tri in spikes if tri_to_rect(tri).right > -5]

        # Collision
        for tri in spikes:
            if rect_triangle_collision(player, tri):
                alive = False
                break

        # Score and difficulty
        score += 1
        if score % 600 == 0:
            scroll_speed += 0.7

    # Draw
    screen.fill(BG_COLOR)
    draw_ground()
    for tri in spikes:
        draw_spike(tri)
    pygame.draw.rect(screen, PLAYER_COLOR, player, border_radius=6)

    # UI
    s_text = font.render(f"Score: {score}", True, (220, 220, 220))
    tip_text = font.render("Space/Up to jump • R to reset", True, (160, 160, 180))
    status_text = font.render("Paused/Dead" if not alive else "", True, (255, 120, 120))
    screen.blit(s_text, (20, 20))
    screen.blit(tip_text, (20, 54))
    if not alive:
        screen.blit(status_text, (20, 88))

    pygame.display.flip()
    clock.tick(60)
```

Level design and polish ideas

- **Tile editor:** Define ground, spikes, orbs, portals in a CSV or a Tiled map. Parse rows into entities at runtime for repeatable patterns and rhythm-aligned layouts. A Python remake called “Pydash” uses Pygame, includes two levels made in Tiled, and demonstrates orb interactions and jump timing.
- **Juice:** Add camera shake on death, particle sparks on jump, and beat-synced color pulses.

- **Audio:** Use short percussive cues for jump/collect, loop background music at 120–150 BPM for tight rhythm feel.

Open-source examples to study

- **Pydash (Pygame implementation):** Geometry Dash–style mechanics, orbs, and Tiled-based levels in Python.
- **NumWorks calculator ports:** Lightweight Geometry Dash–inspired scripts showing minimal loops and collision/score HUD on constrained hardware.
- **Another Pygame recreation:** A small project sketching the structure for graphics, sound, and logic in Python.

Sources: