Drum Machine

PyGame Is Not Just For Games

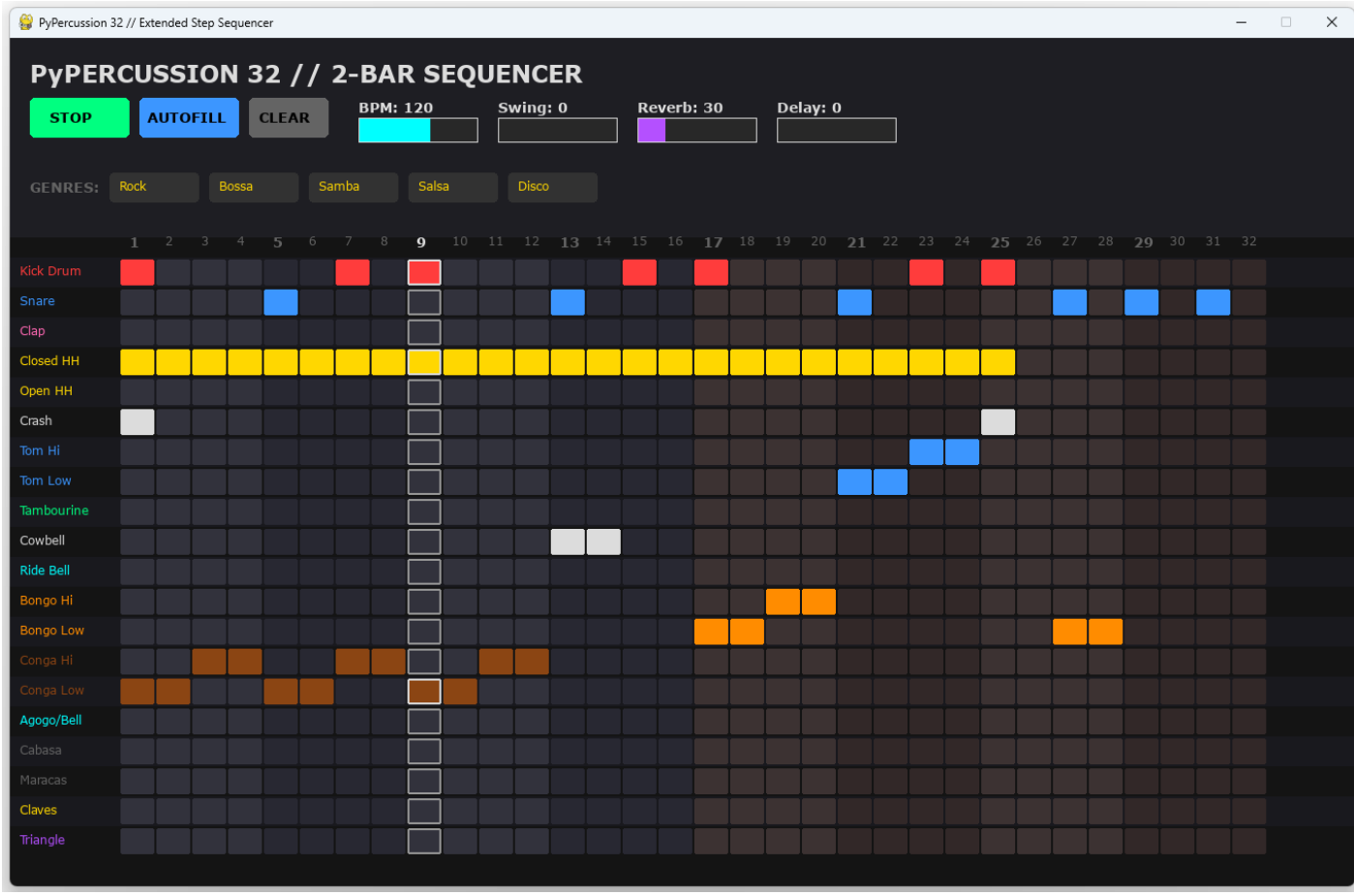
Simon Quellen Field
Follow

7 min read

.

3 days ago

Listen
Share
More
Press enter or click to view image in full size



The Musical Instrument Digital Interface (MIDI) allows electronic instruments to be controlled by a computer.

MIDI commands specify when to play a note or beat, which instrument should play it, and at what volume, among other details.

PyGame speaks MIDI. It can also draw a GUI on the screen and accept mouse and keyboard inputs. It is perfect for creating a digital musical instrument.

Let's Build a Drum Machine

Our digital musical instrument will keep a complex beat, using as many as 20 different instruments. By dividing the time into 32 steps, we can create rhythms that are complex enough to be interesting by themselves, even without melodic accompaniment.

You can easily expand upon this program, adding instruments, or increasing the steps to 64 or 128 or more, and adding more presets.

Presets are rhythms from different genres, such as rock, bossa nova, samba, salsa, or disco.

You can change the speed, add reverb, swing, and delay.

Here is what the code looks like:

```
import pygame
import pygame.midi
import time
import random

# --- CONFIGURATION & CONSTANTS ---
SCREEN_WIDTH = 1350
SCREEN_HEIGHT = 850
FPS = 60

# Colors
DARK_GRAY = (20, 20, 20)
BLACK = (0, 0, 0)
```

```
LIGHT_GRAY = (100, 100, 100)
GRID_GRAY = (40, 40, 40)
GREEN = (0, 255, 128)
RED = (255, 60, 60)
BLUE = (60, 150, 255)
GOLD = (255, 215, 0)
WHITE = (220, 220, 220)
ORANGE = (255, 140, 0)
PURPLE = (180, 80, 255)
PINK = (255, 105, 180)
CYAN = (0, 255, 255)
BROWN = (139, 69, 19)

INSTRUMENTS = [
    {"name": "Kick Drum",   "note": 36, "color": RED},
    {"name": "Snare",       "note": 38, "color": BLUE},
    {"name": "Clap",        "note": 39, "color": PINK},
    {"name": "Closed HH",   "note": 42, "color": GOLD},
    {"name": "Open HH",     "note": 46, "color": GOLD},
    {"name": "Crash",       "note": 49, "color": WHITE},
    {"name": "Tom Hi",      "note": 50, "color": BLUE},
    {"name": "Tom Low",     "note": 45, "color": BLUE},
    {"name": "Tambourine",  "note": 54, "color": GREEN},
    {"name": "Cowbell",     "note": 56, "color": WHITE},
    {"name": "Ride Bell",   "note": 53, "color": CYAN},
    {"name": "Bongo Hi",    "note": 60, "color": ORANGE},
    {"name": "Bongo Low",   "note": 61, "color": ORANGE},
    {"name": "Conga Hi",    "note": 63, "color": BROWN},
    {"name": "Conga Low",   "note": 64, "color": BROWN},
    {"name": "Agogo/Bell",  "note": 67, "color": CYAN},
    {"name": "Cabasa",      "note": 69, "color": LIGHT_GRAY},
    {"name": "Maracas",     "note": 70, "color": LIGHT_GRAY},
    {"name": "Claves",      "note": 75, "color": GOLD},
    {"name": "Triangle",    "note": 81, "color": PURPLE}
]

CC_REVERB = 91
CC_CHORUS = 93

# --- HELPER: GENERATE BLANK PATTERN (32 Steps) ---
def make_empty_pattern():
    return [[0]*32 for _ in range(len(INSTRUMENTS))]

# --- PRESET LIBRARY (32 Steps) ---
PRESETS = {}

# 1. ROCK - Includes a fill at the end
p_rock = make_empty_pattern()
p_rock[0] = [1,0,0,0, 0,0,1,0, 1,0,0,0, 0,0,1,0, 1,0,0,0, 0,0,1,0,
1,0,0,0, 0,0,0,0]
p_rock[1] = [0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0,
0,0,1,0, 1,0,1,0]
p_rock[3] = [1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1,
1,0,0,0, 0,0,0,0]
p_rock[5] = [1,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0,
1,0,0,0, 0,0,0,0]
PRESETS["Rock"] = p_rock
```

```
# 2. BOSSA NOVA
p_bossa = make_empty_pattern()
p_bossa[0]  = [1,0,0,1, 0,0,1,0, 0,0,1,0, 0,1,0,0] * 2
p_bossa[3]  = [1,1,1,1, 1,1,1,1, 1,1,1,1, 1,1,1,1] * 2
p_bossa[18] = [1,0,0,1, 0,0,1,0, 0,0,1,0, 0,1,0,0] * 2
p_bossa[16] = [0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0] * 2
PRESETS["Bossa"] = p_bossa

# 3. SAMBA
p_samba = make_empty_pattern()
p_samba[0]  = [0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0] * 2
p_samba[15] = [0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0] * 2
p_samba[1]  = [0,0,0,0, 0,0,0,1, 0,0,1,0, 0,0,0,1] * 2
p_samba[17] = [1,0,1,0, 1,0,1,0, 1,0,1,0, 1,0,1,0] * 2
p_samba[8]  = [1,0,0,1, 0,1,0,0, 1,0,0,1, 0,1,0,0] * 2
p_samba[13] = [0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,0,0, 0,0,1,0, 0,0,1,0,
0,0,1,0, 0,1,0,0]
PRESETS["Samba"] = p_samba

# 4. SALSA
p_salsa = make_empty_pattern()
p_salsa[0]  = [0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0] * 2
p_salsa[9]  = [1,0,0,1, 0,0,1,0, 0,0,0,0, 1,0,0,0] * 2
p_salsa[18] = [1,0,0,1, 0,0,1,0, 0,0,1,0, 0,1,0,0] * 2
PRESETS["Salsa"] = p_salsa

# 5. DISCO
p_disco = make_empty_pattern()
p_disco[0] = [1,0,0,0, 1,0,0,0, 1,0,0,0, 1,0,0,0] * 2
p_disco[1] = [0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0] * 2
p_disco[4] = [0,0,1,0, 0,0,1,0, 0,0,1,0, 0,0,1,0] * 2
p_disco[2] = [0,0,0,0, 1,0,0,0, 0,0,0,0, 1,0,0,0] * 2
PRESETS["Disco"] = p_disco

class DrumMachine:
    def __init__(self):
        pygame.init()
        pygame.midi.init()

        # Audio Setup
        try:
            self.midi_out =
pygame.midi.Output(pygame.midi.get_default_output_id())
            self.midi_out.set_instrument(0, channel=9)
        except:
            print("Visual Mode Only: No MIDI Output found.")
            self.midi_out = None

        self.screen = pygame.display.set_mode((SCREEN_WIDTH,
SCREEN_HEIGHT))
        pygame.display.set_caption("PyPercussion 32 // Extended Step
Sequencer")
        self.font = pygame.font.SysFont("Verdana", 12)
        self.title_font = pygame.font.SysFont("Verdana", 24, bold=True)
        self.bold_font = pygame.font.SysFont("Verdana", 14, bold=True)

        # Engine State
        self.bpm = 120
```

```
        self.playing = False
        self.current_step = 0
        self.last_tick = time.time()
        self.swing = 0.0

        # Audio Params
        self.reverb = 30
        self.delay = 0
        self.volume = 110

        self.active_pattern = make_empty_pattern()
        self.load_preset("Samba")

        # UI Dimensions
        self.grid_start_y = 220
        self.grid_start_x = 110
        self.step_w = 36
        self.step_h = 30

    def load_preset(self, name):
        if name in PRESETS:
            src = PRESETS[name]
            self.active_pattern = [list(row) for row in src]
            self.send_midi_cc(CC_REVERB, self.reverb)

    def clear_grid(self):
        self.active_pattern = make_empty_pattern()

    def autofill(self):
        for i, track in enumerate(self.active_pattern):
            inst = INSTRUMENTS[i]
            prob = 0.1
            if "Kick" in inst['name']: prob = 0.15
            elif "HH" in inst['name'] or "Shaker" in inst['name']: prob =
0.35

            # Fill 32 steps
            self.active_pattern[i] = [1 if random.random() < prob else 0
for _ in range(32)]

    def send_midi_note(self, note, velocity=100):
        if self.midi_out:
            self.midi_out.note_on(note, velocity, channel=9)

    def send_midi_cc(self, cc, value):
        if self.midi_out:
            self.midi_out.write_short(0xB9, cc, value)

    def play_step(self):
        for i, track in enumerate(self.active_pattern):
            if track[self.current_step] == 1:
                inst = INSTRUMENTS[i]
                self.send_midi_note(inst["note"], self.volume)

    def update(self):
        if self.playing:
            step_duration = (60.0 / self.bpm) / 4.0
```

```
            # Swing Logic (applied to even/odd steps)
            if self.current_step % 2 == 1:
                step_duration += step_duration * self.swing
            else:
                step_duration -= step_duration * self.swing

            if time.time() - self.last_tick >= step_duration:
                # Modulo 32 for the longer loop
                self.current_step = (self.current_step + 1) % 32
                self.play_step()
                self.last_tick = time.time()

    def draw_knob(self, x, y, label, value, max_val, color):
        pygame.draw.rect(self.screen, GRID_GRAY, (x, y, 120, 25))
        fill_w = (value / max_val) * 120
        pygame.draw.rect(self.screen, color, (x, y, fill_w, 25))

        lbl_surf = self.bold_font.render(f"{label}: {int(value)}", True,
WHITE)
        self.screen.blit(lbl_surf, (x, y - 20))
        pygame.draw.rect(self.screen, WHITE, (x, y, 120, 25), 1)

    def draw(self):
        self.screen.fill(DARK_GRAY)

        # --- HEADER ---
        pygame.draw.rect(self.screen, (30,30,35), (0,0,SCREEN_WIDTH, 200))
        title = self.title_font.render("PyPERCUSSION 32 // 2-BAR
SEQUENCER", True, WHITE)
        self.screen.blit(title, (20, 20))

        # --- CONTROLS ---
        play_col = GREEN if self.playing else RED
        play_txt = "STOP" if self.playing else "PLAY"
        pygame.draw.rect(self.screen, play_col, (20, 60, 100, 40),
border_radius=5)
        self.screen.blit(self.bold_font.render(play_txt, True, BLACK),
(40, 70))

        pygame.draw.rect(self.screen, BLUE, (130, 60, 100, 40),
border_radius=5)
        self.screen.blit(self.bold_font.render("AUTOFILL", True, BLACK),
(138, 70))

        pygame.draw.rect(self.screen, LIGHT_GRAY, (240, 60, 80, 40),
border_radius=5)
        self.screen.blit(self.bold_font.render("CLEAR", True, BLACK),
(250, 70))

        # Knobs
        self.draw_knob(350, 80, "BPM", self.bpm, 200, CYAN)
        self.draw_knob(490, 80, "Swing", self.swing*100, 50, GOLD)
        self.draw_knob(630, 80, "Reverb", self.reverb, 127, PURPLE)
        self.draw_knob(770, 80, "Delay", self.delay, 127, PINK)

        # --- PRESETS BAR ---
        p_x = 20
        p_y = 140
```

```
        self.screen.blit(self.bold_font.render("GENRES:", True,
LIGHT_GRAY), (p_x, p_y))
        p_x += 80
        for name in PRESETS.keys():
            col = GOLD if name in PRESETS else WHITE
            bg_col = (50,50,50)
            pygame.draw.rect(self.screen, bg_col, (p_x, p_y-5, 90, 30),
border_radius=4)
            txt = self.font.render(name, True, col)
            self.screen.blit(txt, (p_x + 10, p_y))
            p_x += 100

        # --- SEQUENCER GRID ---
        # Draw Beats Header (0 to 31)
        for i in range(32):
            col = WHITE if i == self.current_step else LIGHT_GRAY
            font = self.bold_font if i % 4 == 0 else self.font
            num_str = str(i+1)
            num = font.render(num_str, True, col)
            self.screen.blit(num, (self.grid_start_x + i * self.step_w +
10, self.grid_start_y - 25))

        # Draw Instruments & Grid
        for row_idx, track in enumerate(self.active_pattern):
            inst = INSTRUMENTS[row_idx]
            base_y = self.grid_start_y + row_idx * self.step_h

            if row_idx % 2 == 0:
                pygame.draw.rect(self.screen, (25,25,30), (0, base_y,
SCREEN_WIDTH, self.step_h))

            lbl = self.font.render(inst['name'], True, inst['color'])
            self.screen.blit(lbl, (10, base_y + 5))

            # 32 Grid Cells
            for col_idx, active in enumerate(track):
                rect_x = self.grid_start_x + col_idx * self.step_w
                rect_y = base_y

                rect = pygame.Rect(rect_x + 1, rect_y + 2, self.step_w -
2, self.step_h - 4)

                is_curr = (col_idx == self.current_step)

                if active:
                    pygame.draw.rect(self.screen, inst['color'], rect,
border_radius=2)
                    if is_curr:
                        pygame.draw.rect(self.screen, WHITE, rect, 2,
border_radius=2)
                else:
                    # Visual grouping: Bars are slightly different colors
                    if col_idx < 16:
                        base_col = (50, 50, 60) if (col_idx // 4) % 2 == 0
else (40, 40, 50)
                    else:
                        base_col = (60, 50, 50) if (col_idx // 4) % 2 == 0
else (50, 40, 40)
```

```
                    pygame.draw.rect(self.screen, base_col, rect,
border_radius=2)
                    if is_curr:
                        pygame.draw.rect(self.screen, (150,150,150), rect,
2, border_radius=2)

        pygame.display.flip()

    def handle_input(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return False

            if event.type == pygame.MOUSEBUTTONDOWN:
                mx, my = pygame.mouse.get_pos()
                m_btn = event.button

                # 1. Grid Interaction (0 to 31)
                if mx > self.grid_start_x and my > self.grid_start_y:
                    col = (mx - self.grid_start_x) // self.step_w
                    row = (my - self.grid_start_y) // self.step_h

                    if 0 <= col < 32 and 0 <= row < len(INSTRUMENTS):
                        self.active_pattern[row][col] = 1 -
self.active_pattern[row][col]
                        if self.active_pattern[row][col] == 1:
                            self.send_midi_note(INSTRUMENTS[row]['note'],
90)

                # 2. Controls
                if 20 <= mx <= 120 and 60 <= my <= 100:
                    self.playing = not self.playing
                    self.current_step = 0

                if 130 <= mx <= 230 and 60 <= my <= 100:
                    self.autofill()

                if 240 <= mx <= 320 and 60 <= my <= 100:
                    self.clear_grid()

                # Knobs
                if 350 <= mx <= 470 and 80 <= my <= 105:
                    self.bpm += 5 if m_btn == 1 else -5
                if 490 <= mx <= 610 and 80 <= my <= 105:
                    self.swing = min(0.5, self.swing + 0.05) if m_btn == 1
else max(0.0, self.swing - 0.05)
                if 630 <= mx <= 750 and 80 <= my <= 105:
                    self.reverb = (self.reverb + 10) % 127
                    self.send_midi_cc(CC_REVERB, self.reverb)
                if 770 <= mx <= 890 and 80 <= my <= 105:
                    self.delay = (self.delay + 10) % 127
                    self.send_midi_cc(CC_CHORUS, self.delay)

                # 3. Preset Buttons
                if 135 <= my <= 170:
                    px = 100
                    for name in PRESETS.keys():
```

```
                        if px <= mx <= px + 90:
                            self.load_preset(name)
                        px += 100

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    self.playing = not self.playing
                    self.current_step = 0

        return True

    def run(self):
        clock = pygame.time.Clock()
        running = True
        while running:
            running = self.handle_input()
            self.update()
            self.draw()
            clock.tick(FPS)

        if self.midi_out:
            del self.midi_out
            pygame.midi.quit()
        pygame.quit()

if __name__ == "__main__":
    app = DrumMachine()
    app.run()
```

You can see the 20 instruments up at the top of the program, and how easy it would be to add as many as your MIDI software has.

Next come the presets. The beats are indexed by the instrument (the MIDI note). Again, easy to expand to your own rhythms, or to other recognized musical genres.

The DrumMachine class encapsulates the behavior, drawing knobs and controls, playing a beat, and handling mouse input. The Autofill method is bound to be disappointing, as random drumming seldom sounds like anything other than a two-year-old with a new drum. Feel free to improve on that.

Here is a short clip of the result:
Listen to the result...

What to Learn From Reading This Code

- How to use MIDI sound from Python.
- How to use PyGame for non-game user interfaces.
- How to code rhythms and play them in sequence.
- How to control timing.

Python

Programming

Software Development

Electronic Music

Music