Mini project

DRUM MACHINE

RESEARCH:-

A drum machine is an electronic musical instrument designed to imitate the sound of drums, cymbals, and other percussion instruments. It is widely used in music production, live performances, and beat-making to create rhythmic patterns without needing a full drum set.

A drum machine often has pre-programmed beats and patterns for popular genres and styles, such as pop music, rock music, and dance music. Most modern drum machines made in the 2010s and 2020s also allow users to program their own rhythms and beats. Drum machines may create sounds using analog synthesis or play prerecorded samples. Drum machines have a range of capabilities, which go from playing a short beat pattern in a loop, to being able to program or record complex song arrangements with changes of meter and style.

Commonly used sounds include the hi-hat, snare, bass drum, crash cymbal, clap, and floor tom, each representing a key part of a typical drum kit.

Hi-hat: short metallic sound used to keep rhythm.
Snare: sharp and crisp sound, adds accents.
Bass drum (Kick): deep, low-pitched beat that gives the base rhythm.
Crash cymbal: loud sound used for transitions.
Clap: bright snapping sound that adds energy.
Floor tom: deep tone for fills and rolls.

This project shows how we can mix music and coding using Python. It helps us understand how drum machines work, how sounds are played in a pattern and how timing is managed. By using Pygame, we can see and hear how each drum sound like hi-hat, snare, and bass is played in rhythm.

In this Python-based project, Pygame, a library primarily used for game development is used to build an interactive drum sequencer.

The project begins with setting up the app environment, followed by drawing the grid-based board where each column represents a beat and each row represents a drum sound. The user can turn notes on and off, allowing them to compose rhythmic patterns. A moving beat tracker visually moves across the screen in real time, highlighting which beats are currently being played. The system then integrates audio playback functionality, where sound samples of each drum (hi-hat, snare, bass, crash, clap, and tom) are triggered at the appropriate steps.

Further functionality includes Play/Pause control, and the ability to adjust total beats and tempo, allowing flexible rhythm design ,users can add their brand new beats. Additionally, Save and Load features are implemented, enabling users to store their composed beats in files and reload them later. A Clear/Reset function helps in quickly restarting the composition process. Finally, the project ends with a cleanup and troubleshooting phase, refining the performance and ensuring smooth playback without lag or sound delay.

APPLICATIONS:

1. Music Production

2. Live Performance

3. Electronic Music and Sound Design

4. Practice Tool

5. Film, TV, and Game Audio

6. Education

7. Experimental and Avant-Garde Music

PYTHON 3.12

1. Python 3.12 executes the script as normal Python code

Python 3.12 runs this file exactly as previous versions would:

1. Imports (import pygame)
2. Variables and constants
3. Lists and list comprehensions
4. Loops (for, while)
5. Conditionals (if / elif / else)
6. Functions
7. Event loops
8. File I/O (open(), reading, writing)

2. Python 3.12 handles Pygame, which does the heavy lifting

Most of the program depends on Pygame, not Python 3.12 specifically.

Python 3.12 is simply:

1. Running the event loop.
2. Calling Pygame drawing functions.
3. Handling sound playback.
4. Managing lists and strings.

All drawing (pygame.draw), sound (pygame.mixer.Sound), and input handling (pygame.event.get()) are part of the Pygame library

3. Python 3.12 handles the timing logic

Python handles the arithmetic and control flow, while Pygame handles frame timing.

4. Python 3.12 reads & writes the save file

This is all core Python functionality that works unchanged in 3.12.

6. String manipulation

Python 3.12's string methods behave the same as earlier versions.

7. Event handling uses Python's basic control structure

Pygame generates eventsPython processes them.

What Part Is Python 3.12 Doing?

Python 3.12 is responsible for:

1. Running the script and executing Python syntax
2. Managing lists, strings, integers, loops
3. File input/output
4. Calling Pygame functions
5. Handling logical flow and updates

Pygame is responsible for:

1. Graphics
2. Sound playback
3. Event capture
4. Timing
5. Rectangles, drawing, fonts

ANALYSIS:-

This is a python program for a drum machine made using pygame. It allows the user to Create beats,Toggle instruments on/off,Adjust BPM,Change the number of beats in a loop, Play/pause, Save beats to a text file, Load beats from the saved file,Delete saved beats.It uses a grid where each row is an instrument, and each column is a beat step.

1. Purpose of the Program

1. A Pygame drum machine / step sequencer.
2. Allows users to create rhythmic patterns, play them, adjust BPM, save/load patterns.

2. Grid System

1. Sequencer grid drawn using draw_grid().
2. Rows indicate instruments (hi-hat, snare, kick, etc.).
3. Columns represent beats.
4. Each square indicates whether a note will play.
5. Colors indicate:

   Active/inactive instrument,Selected/unselected beat,Current playback beat (blue highlight).

3. Buttons & Controls

1. Play/Pause button

   The user can play or pause the beats with the help of this button.

2. BPM +5 / –5

   The user can increase the beats per minute by 5 or decrease the beats per minute by 5.

3. Loop beats +1 / –1

   The user can increase or decrease the loop beats by 1.

4. Clear board button

   This clears the board

5. Save beat button
   This saves the beats you have created.
6. Load beat button

   Loads the beats.

7. Instrument enable/disable controls
   Enables or disables the controls.

4. Menus

Save Menu:

1. Text input for naming a beat

2. Save button

3. Close button

Load Menu:

1. List of saved beats

2. Highlight selected beat

3. Load beat button

4. Delete beat button

5. Close button

5. Instruments Loaded

1. hi-hat, snare, kick, crash, clap, tom

2. Each beat step checks clicked[i][active_beat] to determine if sound plays.

6. Timing

1. bpm controls playback speed.
2. Beat timing uses:
   beat_length = 3600
3. When active_length exceeds this, the beat advances.

7. The clicked matrix

1. Shape: [instruments][beats]
2. Values:

    a. 1 = step active

    b. -1 = step inactive

8. The saved_beats List

1. Stores every line from "saved_beats.txt" as a raw string.
2. Beats are manually parsed using string slicing.

9. Mouse Handling

1. Clicking the grid toggles a beat.

2. Clicking the instrument name toggles the instrument on/off.

3. Clicking buttons changes BPM/beat count.

10. Keyboard Handling

1. TEXTINPUT captures typed names in the save menu.

2. BACKSPACE deletes characters..

Input:

1. Drum sound files: hi-hat, snare, bass drum, crash, clap, and floor tom.
2. Mouse clicks on grid cells to turn beats on or off.
3. Keyboard inputs to start/stop playback.
4. Tempo (BPM) value to control speed.
5. Number of beats or steps in the sequence.
6. Save and load options for storing and retrieving beat patterns.
7. Clear/reset option to start a new pattern.

Output:

1. Playback of drum sounds in rhythm according to the selected pattern.

2. Visual grid display showing active beats and moving playhead.

3. Real-time changes in tempo and pattern playback.

4. Saved pattern files that can be loaded again later.

5. Complete drum rhythm created by combining all selected instruments.

6. User can create their own fun beats and can save them.

ALGORITHM:

1. Start the Program
   Initialize Pygame, mixer, screen, fonts, and colors.

2. Load Sounds
   Load all drum sound files (hi-hat, snare, kick, etc.).

3. Set Default Values
    a. Beats = 8
    b. BPM = 120
    c. Instruments = 6
    d. clicked[][] = all OFF (-1)
    e. active_list[] = all active (1)

4. Load Saved Beats

    If the file exists, read all previously saved beats.

5. Main Game Loop

    a. Clear screen and draw the beat grid.
    b. Draw all control buttons (Play, BPM, Beats, Save, Load, Clear).
    c. If playing:
        i. Count frames until the next beat.
        ii. Move to the next beat.
        iii. Play active instrument sounds for that beat.

    d. Handle mouse clicks:
        i. Toggle grid cells ON/OFF.
        ii. Change BPM and beat count.
        iii. Activate/deactivate instruments.
        iv. Open Save or Load menu.

    e. Handle Save Menu:
        i. Type beat name.
        ii. Save data (name, bpm, beats, clicked[][]) into JSON.

    f. Handle Load Menu:
        i. Select beat.

ii. Load or delete it.

6. Update Display

   Show the updated grid, menus, and animations.

7. Exit
   Save all beats to JSON before closing.

BUILD:

```
import pygame

from pygame import mixer

pygame.init()


black = (0, 0, 0)

white = (255, 255, 255)

gray = (128, 128, 128)

dark_gray = (50, 50, 50)

light_gray = (170, 170, 170)

blue = (0, 255, 255)

red = (255, 0, 0)

green = (0, 255, 0)

gold = (212, 175, 55)

WIDTH = 1400

HEIGHT = 800
```

```
active_length = 0
active_beat = 0

# sounds
'''
hi_hat = mixer.Sound('sounds\kit2\hi hat.wav')
snare = mixer.Sound('sounds\kit2\snare.wav')
kick = mixer.Sound('sounds\kit2\kick.wav')
crash = mixer.Sound('sounds\kit2\crash.wav')
clap = mixer.Sound('sounds\kit2\clap.wav')
tom = mixer.Sound("sounds\kit2\\tom.wav")
'''
hi_hat = mixer.Sound('sounds\hi hat.wav')
snare = mixer.Sound('sounds\snare.wav')
kick = mixer.Sound('sounds\kick.wav')
crash = mixer.Sound('sounds\crash.wav')
clap = mixer.Sound('sounds\clap.wav')
tom = mixer.Sound("sounds\\tom.wav")

screen = pygame.display.set_mode([WIDTH, HEIGHT])
pygame.display.set_caption('The Beat Maker')
```

```
label_font = pygame.font.Font('Roboto-Bold.ttf', 32)
medium_font = pygame.font.Font('Roboto-Bold.ttf', 24)
beat_changed = True
timer = pygame.time.Clock()
fps = 60
beats = 8
bpm = 240
instruments = 6
playing = True
clicked = [[-1 for _ in range(beats)] for _ in range(instruments)]
active_list = [1 for _ in range(instruments)]
pygame.mixer.set_num_channels(instruments * 3)
save_menu = False
load_menu = False
saved_beats = []
file = open('saved_beats.txt', 'r')
for line in file:
    saved_beats.append(line)
beat_name = ''
typing = False
index = 100
```

```
def draw_grid(clicks, beat, actives):
    boxes = []
    left_box = pygame.draw.rect(screen, gray, [0, 0, 200, HEIGHT - 200], 5)
    bottom_box = pygame.draw.rect(screen, gray, [0, HEIGHT - 200, WIDTH, 200],
5)
    for i in range(instruments + 1):
        pygame.draw.line(screen, gray, (0, i * 100), (200, i * 100), 3)
    colors = [gray, white, gray]
    hi_hat_text = label_font.render('Hi Hat', True, colors[actives[0]])
    screen.blit(hi_hat_text, (30, 30))
    snare_text = label_font.render('Snare', True, colors[actives[1]])
    screen.blit(snare_text, (30, 130))
    kick_text = label_font.render('Bass Drum', True, colors[actives[2]])
    screen.blit(kick_text, (30, 230))
    crash_text = label_font.render('Crash', True, colors[actives[3]])
    screen.blit(crash_text, (30, 330))
    clap_text = label_font.render('Clap', True, colors[actives[4]])
    screen.blit(clap_text, (30, 430))
    tom_text = label_font.render('Floor Tom', True, colors[actives[5]])
    screen.blit(tom_text, (30, 530))
```

```
    for i in range(beats):
        for j in range(instruments):
            if clicks[j][i] == -1:
                color = gray
            else:
                if actives[j] == 1:
                    color = green
                else:
                    color = dark_gray
            rect = pygame.draw.rect(screen, color,
                                    [i * ((WIDTH - 200) // beats) + 205, (j * 100) + 5, ((WIDTH
- 200) // beats) - 10,
                                     90], 0, 3)
            pygame.draw.rect(screen, gold, [i * ((WIDTH - 200) // beats) + 200, j *
100, ((WIDTH - 200) // beats), 100],
                             5, 5)
            pygame.draw.rect(screen, black,
                             [i * ((WIDTH - 200) // beats) + 200, j * 100, ((WIDTH - 200) //
beats), 100],
                             2, 5)
            boxes.append((rect, (i, j)))
    active = pygame.draw.rect(screen, blue,
```

```
                        [beat * ((WIDTH - 200) // beats) + 200, 0, ((WIDTH - 200) //
beats), instruments * 100],
                        5, 3)
    return boxes



def play_notes():
    for i in range(len(clicked)):
        if clicked[i][active_beat] == 1 and active_list[i] == 1:
            if i == 0:
                hi_hat.play()
            if i == 1:
                snare.play()
            if i == 2:
                kick.play()
            if i == 3:
                crash.play()
            if i == 4:
                clap.play()
            if i == 5:
                tom.play()
```

```
def draw_save_menu(beat_name, typing):
    pygame.draw.rect(screen, black, [0, 0, WIDTH, HEIGHT])
    menu_text = label_font.render('SAVE MENU: Enter a Name for this beat', True, white)
    screen.blit(menu_text, (400, 40))
    exit_btn = pygame.draw.rect(screen, gray, [WIDTH - 200, HEIGHT - 100, 180, 90], 0, 5)
    exit_text = label_font.render('Close', True, white)
    screen.blit(exit_text, (WIDTH - 160, HEIGHT - 70))
    saving_btn = pygame.draw.rect(screen, gray, [WIDTH // 2 - 100, HEIGHT * 0.75, 200, 100], 0, 5)
    saving_text = label_font.render('Save Beat', True, white)
    screen.blit(saving_text, (WIDTH // 2 - 70, HEIGHT * 0.75 + 30))
    if typing:
        pygame.draw.rect(screen, dark_gray, [400, 200, 600, 200], 0, 5)
    entry_rect = pygame.draw.rect(screen, gray, [400, 200, 600, 200], 5, 5)
    entry_text = label_font.render(f'{beat_name}', True, white)
    screen.blit(entry_text, (430, 250))
    return exit_btn, saving_btn, beat_name, entry_rect
```

```
def draw_load_menu(index):
    loaded_clicked = []
    loaded_beats = 0
    loaded_bpm = 0
    pygame.draw.rect(screen, black, [0, 0, WIDTH, HEIGHT])
    menu_text = label_font.render('LOAD MENU: Select a beat to load in', True, white)
    screen.blit(menu_text, (400, 40))
    exit_btn = pygame.draw.rect(screen, gray, [WIDTH - 200, HEIGHT - 100, 180, 90], 0, 5)
    exit_text = label_font.render('Close', True, white)
    screen.blit(exit_text, (WIDTH - 160, HEIGHT - 70))
    loading_btn = pygame.draw.rect(screen, gray, [WIDTH // 2 - 100, HEIGHT * 0.87, 200, 100], 0, 5)
    loading_text = label_font.render('Load Beat', True, white)
    screen.blit(loading_text, (WIDTH // 2 - 70, HEIGHT * 0.87 + 30))
    delete_btn = pygame.draw.rect(screen, gray, [WIDTH // 2 - 400, HEIGHT * 0.87, 200, 100], 0, 5)
    delete_text = label_font.render('Delete Beat', True, white)
    screen.blit(delete_text, (WIDTH // 2 - 385, HEIGHT * 0.87 + 30))
    if 0 <= index < len(saved_beats):
        pygame.draw.rect(screen, light_gray, [190, 100 + index*50, 1000, 50])
    for beat in range(len(saved_beats)):
```

```
        if beat < 10:
            beat_clicked = []
            row_text = medium_font.render(f'{beat + 1}', True, white)
            screen.blit(row_text, (200, 100 + beat * 50))
            name_index_start = saved_beats[beat].index('name: ') + 6
            name_index_end = saved_beats[beat].index(', beats:')
            name_text =
medium_font.render(saved_beats[beat][name_index_start:name_index_end], True,
white)
            screen.blit(name_text, (240, 100 + beat * 50))
        if 0 <= index < len(saved_beats) and beat == index:
            beats_index_end = saved_beats[beat].index(', bpm:')
            loaded_beats = int(saved_beats[beat][name_index_end +
8:beats_index_end])
            bpm_index_end = saved_beats[beat].index(', selected:')
            loaded_bpm = int(saved_beats[beat][beats_index_end +
6:bpm_index_end])
            loaded_clicks_string = saved_beats[beat][bpm_index_end + 14: -3]
            loaded_clicks_rows = list(loaded_clicks_string.split("], ["))
            for row in range(len(loaded_clicks_rows)):
                loaded_clicks_row = (loaded_clicks_rows[row].split(', '))
                for item in range(len(loaded_clicks_row)):
                    if loaded_clicks_row[item] == '1' or loaded_clicks_row[item] == '-1':
```

```
                loaded_clicks_row[item] = int(loaded_clicks_row[item])
            beat_clicked.append(loaded_clicks_row)
            loaded_clicked = beat_clicked
    loaded_info = [loaded_beats, loaded_bpm, loaded_clicked]
    entry_rect = pygame.draw.rect(screen, gray, [190, 90, 1000, 600], 5, 5)
    return exit_btn, loading_btn, entry_rect, delete_btn, loaded_info



run = True
while run:
    timer.tick(fps)
    screen.fill(black)
    boxes = draw_grid(clicked, active_beat, active_list)
    # drawing lower menu
    play_pause = pygame.draw.rect(screen, gray, [50, HEIGHT - 150, 200, 100], 0,
5)
    play_text = label_font.render('Play/Pause', True, white)
    screen.blit(play_text, (70, HEIGHT - 130))
    if playing:
        play_text2 = medium_font.render('Playing', True, dark_gray)
    else:
        play_text2 = medium_font.render('Paused', True, dark_gray)
```

```
    screen.blit(play_text2, (70, HEIGHT - 100))

    # beats per minute buttons

    bpm_rect = pygame.draw.rect(screen, gray, [300, HEIGHT - 150, 200, 100], 5,
5)

    bpm_text = medium_font.render('Beats Per Minute', True, white)

    screen.blit(bpm_text, (308, HEIGHT - 130))

    bpm_text2 = label_font.render(f'{bpm}', True, white)

    screen.blit(bpm_text2, (370, HEIGHT - 100))

    bpm_add_rect = pygame.draw.rect(screen, gray, [510, HEIGHT - 150, 48, 48],
0, 5)

    bpm_sub_rect = pygame.draw.rect(screen, gray, [510, HEIGHT - 100, 48, 48], 0,
5)

    add_text = medium_font.render('+5', True, white)

    screen.blit(add_text, (520, HEIGHT - 140))

    sub_text = medium_font.render('-5', True, white)

    screen.blit(sub_text, (520, HEIGHT - 90))

    # beats per loop buttons

    beats_rect = pygame.draw.rect(screen, gray, [600, HEIGHT - 150, 200, 100], 5,
5)

    beats_text = medium_font.render('Beats In Loop', True, white)

    screen.blit(beats_text, (612, HEIGHT - 130))

    beats_text2 = label_font.render(f'{beats}', True, white)

    screen.blit(beats_text2, (670, HEIGHT - 100))
```

```
    beats_add_rect = pygame.draw.rect(screen, gray, [810, HEIGHT - 150, 48, 48],
0, 5)

    beats_sub_rect = pygame.draw.rect(screen, gray, [810, HEIGHT - 100, 48, 48],
0, 5)

    add_text2 = medium_font.render('+1', True, white)

    screen.blit(add_text2, (820, HEIGHT - 140))

    sub_text2 = medium_font.render('-1', True, white)

    screen.blit(sub_text2, (820, HEIGHT - 90))

    # clear board button

    clear = pygame.draw.rect(screen, gray, [1150, HEIGHT - 150, 200, 100], 0, 5)

    play_text = label_font.render('Clear Board', True, white)

    screen.blit(play_text, (1160, HEIGHT - 130))

    # save and load buttons

    save_button = pygame.draw.rect(screen, gray, [900, HEIGHT - 150, 200, 48], 0,
5)

    save_text = label_font.render('Save Beat', True, white)

    screen.blit(save_text, (920, HEIGHT - 140))

    load_button = pygame.draw.rect(screen, gray, [900, HEIGHT - 98, 200, 48], 0,
5)

    load_text = label_font.render('Load Beat', True, white)

    screen.blit(load_text, (920, HEIGHT - 90))

    # instrument rectangles

    instrument_rects = []
```

```
    for i in range(instruments):
        rect = pygame.rect.Rect((0, i * 100), (200, 100))
        instrument_rects.append(rect)
    if beat_changed:
        play_notes()
        beat_changed = False
    if save_menu:
        exit_button, saving_button, beat_name, entry_rect = draw_save_menu(beat_name, typing)
    elif load_menu:
        exit_button, loading_button, entry_rect, delete_button, loaded_information = draw_load_menu(index)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False
        if event.type == pygame.MOUSEBUTTONDOWN and not save_menu and not load_menu:
            for i in range(len(boxes)):
                if boxes[i][0].collidepoint(event.pos):
                    coords = boxes[i][1]
                    clicked[coords[1]][coords[0]] *= -1
        if event.type == pygame.MOUSEBUTTONUP and not save_menu and not load_menu:
```

```
if play_pause.collidepoint(event.pos) and playing:
    playing = False
elif play_pause.collidepoint(event.pos) and not playing:
    playing = True
    active_beat = 0
    active_length = 0
if beats_add_rect.collidepoint(event.pos):
    beats += 1
    for i in range(len(clicked)):
        clicked[i].append(-1)
elif beats_sub_rect.collidepoint(event.pos):
    beats -= 1
    for i in range(len(clicked)):
        clicked[i].pop(-1)
if bpm_add_rect.collidepoint(event.pos):
    bpm += 5
elif bpm_sub_rect.collidepoint(event.pos):
    bpm -= 5
if clear.collidepoint(event.pos):
    clicked = [[-1 for _ in range(beats)] for _ in range(instruments)]
for i in range(len(instrument_rects)):
```

```
            if instrument_rects[i].collidepoint(event.pos):
                active_list[i] *= -1
        if save_button.collidepoint(event.pos):
            save_menu = True
        if load_button.collidepoint(event.pos):
            load_menu = True
            playing = False
    elif event.type == pygame.MOUSEBUTTONUP:
        if exit_button.collidepoint(event.pos):
            save_menu = False
            load_menu = False
            playing = True
            typing = False
            beat_name = ''
        if entry_rect.collidepoint(event.pos):
            if save_menu:
                if typing:
                    typing = False
                else:
                    typing = True
            if load_menu:
```

```
                index = (event.pos[1] - 100) // 50
        if save_menu:
            if saving_button.collidepoint(event.pos):
                file = open('saved_beats.txt', 'w')
                saved_beats.append(f'\nname: {beat_name}, beats: {beats}, bpm: {bpm}, selected: {clicked}')
                for i in range(len(saved_beats)):
                    file.write(str(saved_beats[i]))
                file.close()
                save_menu = False
                load_menu = False
                playing = True
                typing = False
                beat_name = ''
        if load_menu:
            if delete_button.collidepoint(event.pos):
                if 0 <= index < len(saved_beats):
                    saved_beats.pop(index)
            if loading_button.collidepoint(event.pos):
                if 0 <= index < len(saved_beats):
                    beats = loaded_information[0]
                    bpm = loaded_information[1]
```

```
                clicked = loaded_information[2]
                index = 100
                save_menu = False
                load_menu = False
                playing = True
                typing = False
    if event.type == pygame.TEXTINPUT and typing:
        beat_name += event.text
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_BACKSPACE and len(beat_name) > 0:
            beat_name = beat_name[:-1]

beat_length = 3600 // bpm

if playing:
    if active_length < beat_length:
        active_length += 1
    else:
        active_length = 0
        if active_beat < beats - 1:
            active_beat += 1
```

```
            beat_changed = True
        else:
            active_beat = 0
            beat_changed = True

    pygame.display.flip()

file = open('saved_beats.txt', 'w')
for i in range(len(saved_beats)):
    file.write(str(saved_beats[i]))
file.close()
pygame.quit()
```

TEST:

Beat maker.mp4

IMPLEMENTATION:

https://github.com/riddhihk05-create/Beat-maker-drum-system/tree/main

https://github.com/Shir0311/Beatmaker