**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](https://en.wikipedia.org/wiki/Rhythm) and beats. Drum machines may create sounds using [analog synthesis](https://en.wikipedia.org/wiki/Analog_synthesizer) or play prerecorded [samples](https://en.wikipedia.org/wiki/Sampling_(music)). 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](https://en.wikipedia.org/wiki/Arrangement_(music)) 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 totalbeats 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.

1. BPM +5 / –5

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

1. Loop beats +1 / –1

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

1. Clear board button

This clears the board

1. Save beat button  
   This saves the beats you have created.
2. Load beat button

Loads the beats.

1. 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:  
   1. 1 = step active
   2. -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 SoundsLoad all drum sound files (hi-hat, snare, kick, etc.).
3. Set Default Values
   1. Beats = 8
   2. BPM = 120
   3. Instruments = 6
   4. clicked[][] = all OFF (-1)
   5. active\_list[] = all active (1)
4. Load Saved Beats

If the file exists, read all previously saved beats.

1. Main Game Loop 
   1. Clear screen and draw the beat grid.
   2. Draw all control buttons (Play, BPM, Beats, Save, Load, Clear).
   3. If playing:
      1. Count frames until the next beat.
      2. Move to the next beat.
      3. Play active instrument sounds for that beat.
   4. Handle mouse clicks:
      1. Toggle grid cells ON/OFF.
      2. Change BPM and beat count.
      3. Activate/deactivate instruments.
      4. Open Save or Load menu.
   5. Handle Save Menu:
      1. Type beat name.
      2. Save data (name, bpm, beats, clicked[][]) into JSON.
   6. Handle Load Menu:
      1. Select beat.
      2. Load or delete it.
2. Update Display

Show the updated grid, menus, and animations.

1. 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:



IMPLEMENTATION: