Department of Computer Science
and Applications
Panjab University, Chandigarh

C++ PROJECT:
“BEATBYTE”

Submitted By (Group members) :
Amisha Sharma (56)
Parneet Kaur (81)
Shifali Gautam (95)

Course : MCA 1 (E)

Submitted To : Dr Balwinder Kaur

BEATBYTE – A Piano Game in C++ using SFML

Introduction:

BeatByte is an interactive **piano simulation game** developed using **C++** and **SFML.** It allows users to play musical notes virtually using their keyboard.

Purpose & Objectives:

The main goal of **BeatByte** is to create a virtual piano that provides **realistic sound feedback** while being intuitive and easy to use. It aims to make music learning accessible and enjoyable for everyone.

Features of This Code:

✔ Plays piano sounds when keys are pressed.
✔ Displays a visual piano keyboard.
✔ Uses SFML for handling graphics and audio.

Technology Used:

- **Programming Language:** C++
- **Graphics & Audio Library:** SFML
- **Development Environment:** Visual Studio

CODE

```cpp
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>
#include <unordered_map>
#include <vector>
#include <string>
#include <iostream>
```

```cpp
int main() {
    sf::RenderWindow window(sf::VideoMode(650, 300), "SFML Piano");

    // Load font
    sf::Font font;
    if (!font.loadFromFile("../Fonts/arial.ttf")) {
        std::cerr << "Error: Could not load font!" << std::endl;
        return -1;
    }
```

```cpp
// Load sound buffers
std::unordered_map<sf::Keyboard::Key, sf::Sound> sounds;
std::vector<std::string> notes = {
    "../Music/C_s.wav", "../Music/D_s.wav", "../Music/F_s.wav",
    "../Music/G_s.wav","../Music/Bb.wav","../Music/C_s1.wav",
    "../Music/D_s1.wav", "../Music/C.wav","../Music/D.wav",
    "../Music/E.wav","../Music/F.wav", "../Music/G.wav",
    "../Music/A.wav", "../Music/B.wav", "../Music/C1.wav",
    "../Music/D1.wav", "../Music/E1.wav", "../Music/F1.wav"
};
```

```cpp
std::vector<sf::Keyboard::Key> keys = {
    sf::Keyboard::W, sf::Keyboard::E, sf::Keyboard::T,
    sf::Keyboard::Y, sf::Keyboard::U, sf::Keyboard::O,
    sf::Keyboard::P, sf::Keyboard::A, sf::Keyboard::S,
    sf::Keyboard::D, sf::Keyboard::F, sf::Keyboard::G,
    sf::Keyboard::H, sf::Keyboard::J, sf::Keyboard::K,
    sf::Keyboard::L, sf::Keyboard::V, sf::Keyboard::B
};
```

```cpp
std::vector<std::string> keyLabels = { "W", "E", "T", "Y",
    "U", "O", "P", "A", "S", "D", "F", "G", "H", "J", "K",
    "L", "V", "B" };

std::vector<sf::SoundBuffer>buffers(notes.size());
std::unordered_map<sf::Keyboard::Key,sf::RectangleShape>keyShapes;
std::unordered_map<sf::Keyboard::Key,sf::Text>keyTexts;

int keyWidth=40,keyHeight=200,blackKeyWidth=30,blackKeyHeight=120;
```

```cpp
// Black key positions
std::unordered_map<sf::Keyboard::Key, float> blackKeyPositions = {
  {sf::Keyboard::W,75},{sf::Keyboard::E,125},{sf::Keyboard::T,225},
  {sf::Keyboard::Y,275},{sf::Keyboard::U,325},{sf::Keyboard::O,425},
  {sf::Keyboard::P,475}
};
```

```cpp
// White key positions
std::unordered_map<sf::Keyboard::Key, float> keyPositions = {
 {sf::Keyboard::A,50},{sf::Keyboard::S,100},{sf::Keyboard::D,150},
 {sf::Keyboard::F,200},{sf::Keyboard::G,250},{sf::Keyboard::H,300},
 {sf::Keyboard::J,350},{sf::Keyboard::K,400},{sf::Keyboard::L,450},
 {sf::Keyboard::V,500},{sf::Keyboard::B,550}
};
```

```cpp
// Load sounds and create keys
for (size_t i = 0; i < notes.size(); ++i) {
 if (!buffers[i].loadFromFile(notes[i])){
   std::cerr<<"Error loading: "<<notes[i]<<std::endl;
   return -1;
 }
  sounds[keys[i]].setBuffer(buffers[i]);

  sf::RectangleShape key;
  bool isBlackKey=blackKeyPositions.find(keys[i])!=blackKeyPositions.end();
```

```
if (isBlackKey) {
  key = sf::RectangleShape(sf::Vector2f(blackKeyWidth, blackKeyHeight));
  key.setPosition(blackKeyPositions[keys[i]],50);
  key.setFillColor(sf::Color::Black);
  }
else {
  key = sf::RectangleShape(sf::Vector2f(keyWidth, keyHeight));
  key.setPosition(keyPositions[keys[i]], 50);
  key.setFillColor(sf::Color::White);
  }
  key.setOutlineThickness(3);
  key.setOutlineColor(sf::Color::Black);
  keyShapes[keys[i]] = key;
// Add key labels
sf::Text text;
text.setFont(font);
text.setString(keyLabels[i]);
text.setCharacterSize(20);
text.setFillColor(isBlackKey ? sf::Color::White : sf::Color::Black);
text.setStyle(sf::Text::Bold);
text.setPosition(key.getPosition().x+(key.getSize().x - text.getLocalBounds().width)/2,
    key.getPosition().y+key.getSize().y - 30);
keyTexts[keys[i]] = text;
}
```

```
while (window.isOpen()) {
  sf::Event event;
  while (window.pollEvent(event)) {
   if (event.type == sf::Event::Closed)
   window.close();

   if(event.type==sf::Event::KeyPressed && sounds.find(event.key.code) != sounds.end())
   {
   if(sounds[event.key.code].getStatus() != sf::Sound::Playing) {
   sounds[event.key.code].play();
   }
   keyShapes[event.key.code].setFillColor(sf::Color::Cyan);
   }
   if(event.type == sf::Event::KeyReleased && keyShapes.find(event.key.code) != keyShapes.end())
   {
   if(blackKeyPositions.find(event.key.code) != blackKeyPositions.end())
   {
    keyShapes[event.key.code].setFillColor(sf::Color::Black);
     }
   else {
   keyShapes[event.key.code].setFillColor(sf::Color::White);
           }
       }
    }
```

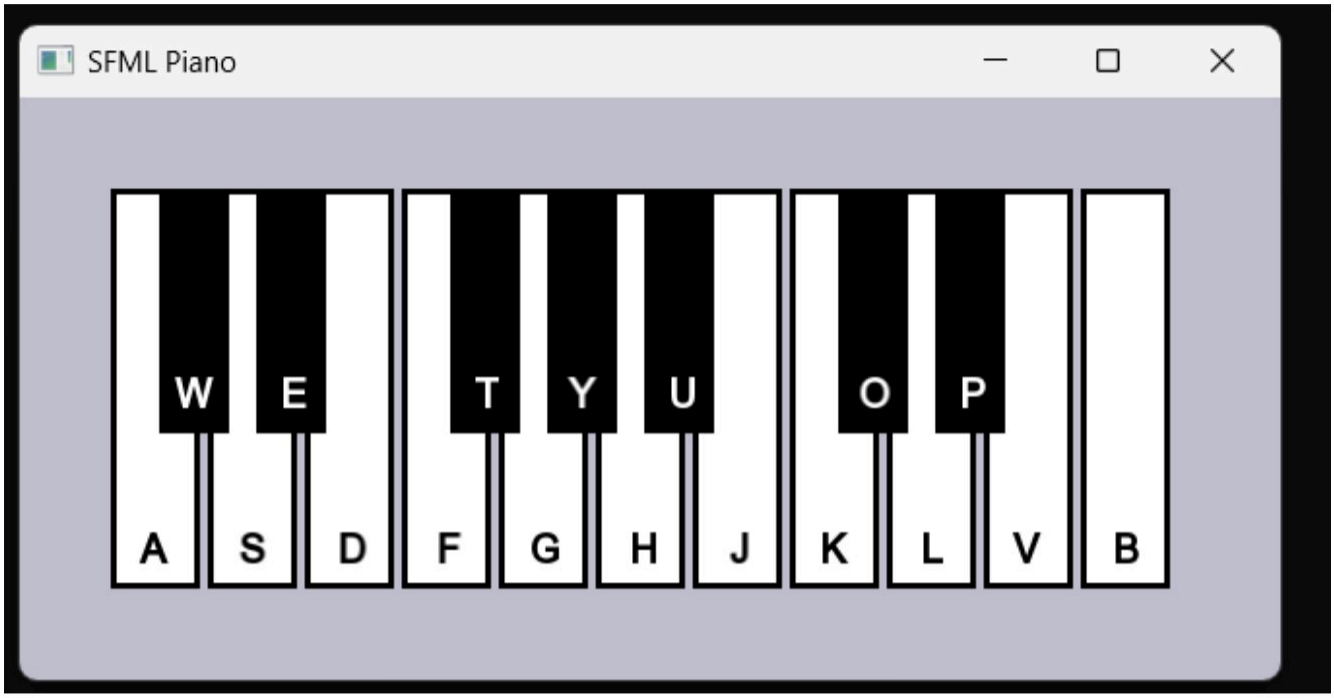
```
window.clear(sf::Color(191, 190, 207));

// Draw white keys first
for (const auto& pair : keyShapes) {
    if (blackKeyPositions.find(pair.first) == blackKeyPositions.end()) {
        window.draw(pair.second);
    }
}
// Draw black keys on top
for (const auto& pair : keyShapes) {
    if (blackKeyPositions.find(pair.first) != blackKeyPositions.end()) {
        window.draw(pair.second);
    }
}
// Draw labels
for (const auto& pair : keyTexts) {
    window.draw(pair.second);
}

        window.display();
    }

    return 0;
}
```

OUTPUT



EXPLANATION OF CODE

1. Include Necessary Libraries

- **SFML/Graphics.hpp:** Used for rendering the graphical window and drawing shapes (piano keys).
- **SFML/Audio.hpp:** Used for loading and playing sound files (piano notes).
- **unordered_map:** Used for efficient key-value lookups (mapping keys to sounds and key shapes).
- **vector:** Used for storing lists of notes, keys, and other elements.
- **string:** Used for handling file paths and labels.
- **iostream:** Used for error logging.

2. Create the Main Window

```
sf::RenderWindow window(sf::VideoMode(650, 300), "SFML Piano");
```

Creates a 650×300 pixel window titled "SFML Piano";

3. Load Font for Key Labels

```
// Load font
sf::Font font;
if (!font.loadFromFile("../Fonts/arial.ttf")) {
    std::cerr << "Error: Could not load font!" << std::endl;
    return -1;
}
```

- Loads a font (arial.ttf) for displaying key labels.
- If the font file is missing, the program prints an error and exits.

4. Define Notes and Corresponding Keyboard Keys

Define the Sound File Paths

```
// Load sound buffers
std::unordered_map<sf::Keyboard::Key, sf::Sound> sounds;
std::vector<std::string> notes = {
    "../Music/C_s.wav", "../Music/D_s.wav", "../Music/F_s.wav",
    "../Music/G_s.wav","../Music/Bb.wav","../Music/C_s1.wav",
    "../Music/D_s1.wav", "../Music/C.wav","../Music/D.wav",
    "../Music/E.wav","../Music/F.wav", "../Music/G.wav",
    "../Music/A.wav", "../Music/B.wav", "../Music/C1.wav",
    "../Music/D1.wav", "../Music/E1.wav", "../Music/F1.wav"
};
```

Map Notes to Keyboard Keys

```
std::vector<sf::Keyboard::Key> keys = {
    sf::Keyboard::W, sf::Keyboard::E, sf::Keyboard::T,
    sf::Keyboard::Y, sf::Keyboard::U, sf::Keyboard::O,
    sf::Keyboard::P, sf::Keyboard::A, sf::Keyboard::S,
    sf::Keyboard::D, sf::Keyboard::F, sf::Keyboard::G,
    sf::Keyboard::H, sf::Keyboard::J, sf::Keyboard::K,
    sf::Keyboard::L, sf::Keyboard::V, sf::Keyboard::B
};
```

Define Key Labels

```
std::vector<std::string> keyLabels = { "W", "E", "T", "Y",
    "U", "O", "P", "A", "S", "D", "F", "G", "H", "J", "K",
    "L", "V", "B" };
```

5. Create Piano Keys

```
int keyWidth=40,keyHeight=200,blackKeyWidth=30,blackKeyHeight=120;

// Black key positions
std::unordered_map<sf::Keyboard::Key, float> blackKeyPositions = {
  {sf::Keyboard::W,75},{sf::Keyboard::E,125},{sf::Keyboard::T,225},
  {sf::Keyboard::Y,275},{sf::Keyboard::U,325},{sf::Keyboard::O,425},
  {sf::Keyboard::P,475}
};

// White key positions
std::unordered_map<sf::Keyboard::Key, float> keyPositions = {
 {sf::Keyboard::A,50},{sf::Keyboard::S,100},{sf::Keyboard::D,150},
 {sf::Keyboard::F,200},{sf::Keyboard::G,250},{sf::Keyboard::H,300},
 {sf::Keyboard::J,350},{sf::Keyboard::K,400},{sf::Keyboard::L,450},
 {sf::Keyboard::V,500},{sf::Keyboard::B,550}
};
```

6. Load Sounds and Create Key Shapes

- **Loads sounds** into **sf::SoundBuffer** and associates them with **sf::Sound**.
- **Creates white and black keys** and positions them correctly.
- **Assigns labels** to keys.

7. Main Game Loop

```
while (window.isOpen()) {
  sf::Event event;
  while (window.pollEvent(event)) {
   if (event.type == sf::Event::Closed)
   window.close();

   if(event.type==sf::Event::KeyPressed && sounds.find(event.key.code) != sounds.end())
   {
   if(sounds[event.key.code].getStatus() != sf::Sound::Playing) {
   sounds[event.key.code].play();
   }
   keyShapes[event.key.code].setFillColor(sf::Color::Cyan);
   }
   if(event.type == sf::Event::KeyReleased && keyShapes.find(event.key.code) != keyShapes.end())
   {
   if(blackKeyPositions.find(event.key.code) != blackKeyPositions.end())
   {
    keyShapes[event.key.code].setFillColor(sf::Color::Black);
     }
   else {
   keyShapes[event.key.code].setFillColor(sf::Color::White);
            }
        }
    }

    window.clear(sf::Color(191, 190, 207));
```

Summary

- This **SFML piano program** uses **graphical rendering** and **audio playback**.
- It **maps keyboard keys** to corresponding piano notes.
- It provides **visual and audio feedback** when keys are pressed.
- **White and black keys** are drawn correctly.