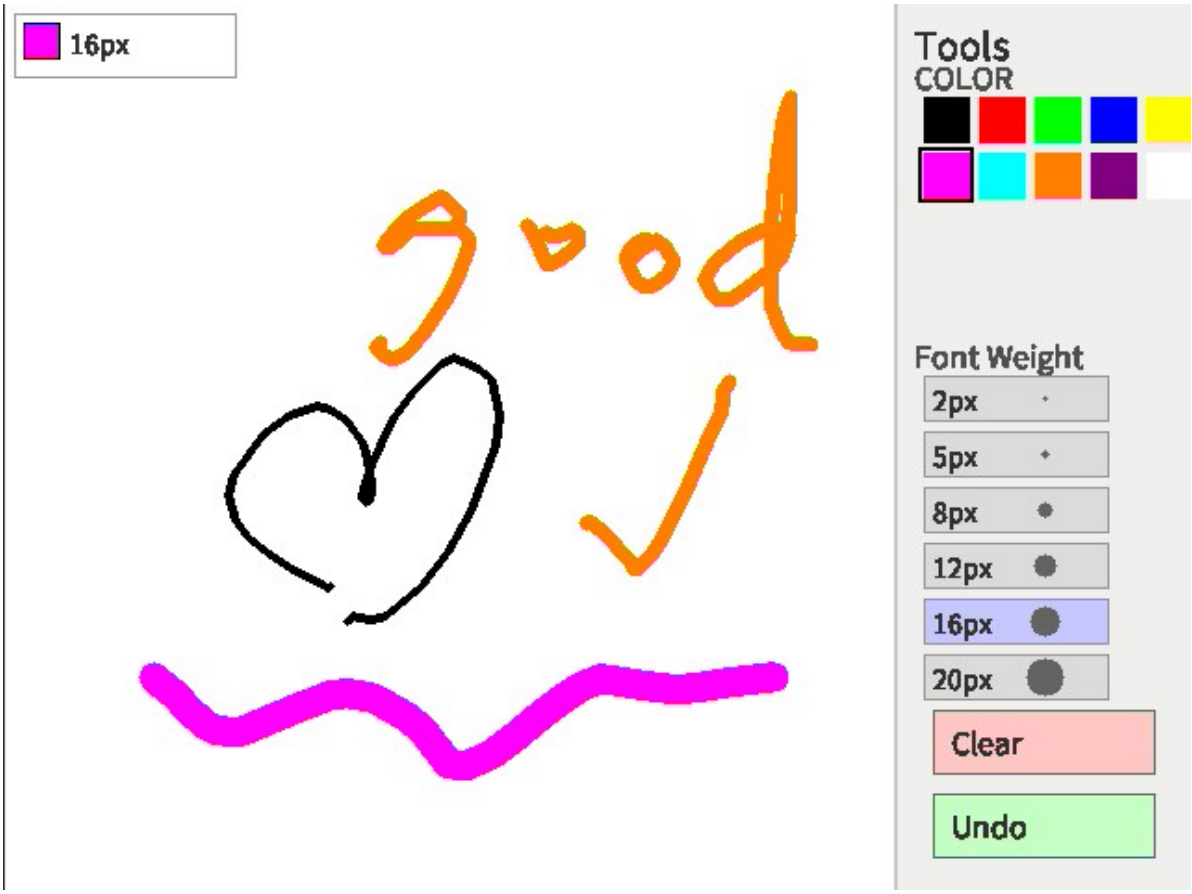
Enhanced Touch Painting

Example Results

The source code file for this example is named `enhanced_touch_drawing.py`.

Connect to the IDE and run the code. You will see a drawing application interface with a canvas and toolbar. You can draw on the canvas by touching the screen, select different colors and brush sizes, and clear the canvas. The toolbar contains color selection, brush size adjustment, and control buttons (such as clearing the screen).



Code Explanation

1. Import Necessary Modules

```
import time, os, sys
from media.display import *
from media.media import *
from machine import TOUCH
```

- `time`, `os`, `sys`: System-related modules for time control, operating system functions, and system operations. - `media.display` and `media.media`: used for screen display and media resource management.
- `machine.TOUCH`: used to initialize and operate the touch sensor.

2. Initialize the touch sensor and screen constants

```
tp = TOUCH(0)
DISPLAY_WIDTH = 640
DISPLAY_HEIGHT = 480
CANVAS_WIDTH = 480
CANVAS_HEIGHT = 480
TOOLBAR_WIDTH = 160
```

- `TOUCH(0)`: Initializes the touch sensor on pin 0, `tp` is used to read touch data later.
- `DISPLAY_WIDTH` and `DISPLAY_HEIGHT`: Define the total screen resolution as 640x480.
- `CANVAS_WIDTH` and `CANVAS_HEIGHT`: Define the canvas area's resolution to 480x480.
- `TOOLBAR_WIDTH`: Define the toolbar width to 160 pixels.

3. Define the `DrawingApp` class

```
class DrawingApp:
    def __init__(self):
        """Initialize the drawing program"""
```

The `DrawingApp` class is the core of the entire drawing application, responsible for managing the drawing state, user interface drawing, and handling touch events.

3.1 Initializing the Drawing State and UI Parameters

```
self.current_color = (0, 0, 0)
self.current_brush_size = 5
self.is_drawing = False
self.last_x = None
self.last_y = None
```

- `current_color`: The currently selected drawing color, defaults to black.
- `current_brush_size`: The current brush size, defaults to 5 pixels.
- `is_drawing`: A Boolean value that records whether drawing is in progress.
- `last_x` and `last_y`: Record the coordinates of the last touch point, used for drawing continuous lines.

In addition, the code defines a color palette, brush size options, and UI layout parameters (such as the position and size of the color selection area, brush size selection area, and control button area).

3.2 Creating the Display Image and Initializing the Interface

```
self.display_image = image.Image(DISPLAY_WIDTH, DISPLAY_HEIGHT, image.RGB888)
self.clear_canvas()
self.draw_ui()
```

- Create an RGB888 image object, `display_image`, to display the entire interface and drawing content.
- Call `clear_canvas()` to clear the canvas and set a white background.
- Call `draw_ui()` to draw the user interface (toolbar, color selection, brush size selection, and control buttons).

4. User Interface Drawing Methods

4.1 Clearing the Canvas

```
def clear_canvas(self):
    """Clear the canvas"""
    self.display_image.clear()
    self.display_image.draw_rectangle(0, 0, CANVAS_WIDTH, CANVAS_HEIGHT, color=
(255, 255, 255), fill=True)
    self.display_image.draw_rectangle(CANVAS_WIDTH, 0, TOOLBAR_WIDTH,
CANVAS_HEIGHT, color=(240, 240, 240), fill=True)
    self.display_image.draw_line(CANVAS_WIDTH, 0, CANVAS_WIDTH, CANVAS_HEIGHT,
color=(200, 200, 200), thickness=2)
```

- Clear the entire display image.
- Draw a white canvas background and a gray toolbar background.
- Draw a separator line between the canvas and toolbar.

4.2 Drawing the User Interface

```
def draw_ui(self):
    """Drawing the user interface"""
```

- `draw_ui()` calls several sub-methods to draw the toolbar, color palette, brush size selection, and control buttons.
- Specific methods include `draw_color_palette()` (drawing the color selection), `draw_brush_size_palette()` (drawing the brush size selection), `draw_control_buttons()` (drawing the control buttons), and `draw_status_info()` (drawing the current status information).

5. Touch Event Handling Logic

5.1 Main Touch Handling Method

```
def handle_touch(self, x, y, event):
    """Handling touch events"""
    if x < CANVAS_WIDTH:
        self.handle_drawing(x, y, event)
    else:
        self.handle_ui_interaction(x, y, event)
```

- Based on the touch point's location, determine whether it's within the canvas area (`x < CANVAS_WIDTH`) or the toolbar area, and call `handle_drawing()` or `handle_ui_interaction()` to handle the situation, respectively.

5.2 Handling Drawing Operations

```
def handle_drawing(self, x, y, event):
    """Handling painting operations"""
    if event == TOUCH.EVENT_DOWN:
        self.is_drawing = True
        self.last_x = x
        self.last_y = y
        self.display_image.draw_circle(x, y, self.current_brush_size//2,
color=self.current_color, fill=True)
    elif event == TOUCH.EVENT_MOVE and self.is_drawing:
        if self.last_x is not None and self.last_y is not None:
            self.display_image.draw_line(self.last_x, self.last_y, x, y,
color=self.current_color, thickness=self.current_brush_size)
        self.last_x = x
        self.last_y = y
    elif event == TOUCH.EVENT_UP:
        self.is_drawing = False
        self.last_x = None
        self.last_y = None
```

- Touch Down Event (`EVENT_DOWN`): Starts drawing, records the starting point, and draws a circle.

- Touch Move Event (`EVENT_MOVE`): If drawing is in progress, draws a line connecting the previous point and the current point.
- Touch Up Event (`EVENT_UP`): Ends drawing and resets the coordinates of the previous point.

5.3 Handling UI Interactions

```
def handle_ui_interaction(self, x, y, event):
    """Handling UI interactions"""
    if event != TOUCH.EVENT_DOWN:
        return
    if self.color_palette_y <= y <= self.color_palette_y + 60:
        self.handle_color_selection(x, y)
    elif self.brush_palette_y <= y <= self.brush_palette_y +
len(self.brush_sizes) * self.brush_spacing:
        self.handle_brush_size_selection(x, y)
    elif self.control_buttons_y <= y <= self.control_buttons_y + 80:
        self.handle_control_buttons(x, y)
```

- Handles only touch-down events (`EVENT_DOWN`).
- Based on the Y coordinate of the touch point, determines whether it is in the color selection area, brush size selection area, or control button area, and calls the corresponding method.

6. Update the display

```
def update_display(self):
    """Update display - simplified version, avoiding pixel level manipulation"""
    Display.show_image(self.display_image)
```

- Use `Display.show_image()` to update the screen to display the current image, avoiding complex pixel-level operations.

7. Main Program Logic

```
def main():
    """Main program"""
    print("Launching the enhanced touch painting program...")
    Display.init(Display.ST7701, width=DISPLAY_WIDTH, height=DISPLAY_HEIGHT,
to_ide=True)
    MediaManager.init()
    app = DrawingApp()
```

- Initialize the display and media manager.
- Create a `DrawingApp` instance and start the drawing application.

7.1 Main Loop: Reading and Processing Touch Data

```
while True:
    os.exitpoint()
    points = tp.read(1)
    if len(points) > 0:
        pt = points[0]
        app.handle_touch(pt.x, pt.y, pt.event)
        app.update_display()
    time.sleep(0.02)
```

- Continuously read touch data.
- If there is a touch event, call `handle_touch()` to handle it and update the display.
- Update every 20 milliseconds (approximately 50 FPS).

7.2 Exception Handling and Resource Release

```
except KeyboardInterrupt:
    print("User stops the program")
except Exception as e:
    print(f"Program exception: {e}")
finally:
    Display.deinit()
    os.exitpoint(os.EXITPOINT_ENABLE_SLEEP)
    time.sleep_ms(100)
    MediaManager.deinit()
```

- Capture user interrupts (KeyboardInterrupt) and other exceptions (Exception) and print relevant information.
- Deinitialize the display, set the exit point to sleep mode, and release media resources.

8. Main program entry

```
if __name__ == "__main__":
    os.exitpoint(os.EXITPOINT_ENABLE)
    main()
```