Paint App Refactor Plan

Overview

This document outlines a comprehensive refactor plan for the Paint App to address state management issues, tool lifecycle, and gizmo management through a proper architectural foundation.

Core Architecture

State Machine

```
enum EditorState {
    Idle,
    Drawing {
        tool: DrawingTool,
        stroke: Option<Stroke>,
    },
    Selecting {
        mode: SelectionMode,
        in_progress: Option<SelectionInProgress>,
    },
    Transforming {
        layer_id: LayerId,
        gizmo: TransformGizmo,
    }
}

// Valid state transitions
impl EditorState {
    fn can_transition_to(&self, new_state: &EditorState) -> bool {
        match (self, new_state) {
            // From Idle, we can go to any state
            (EditorState::Idle, _) => true,

            // From Drawing, we can only finish or cancel
            (EditorState::Drawing { .. }, EditorState::Idle) => true,

            // From Selecting, we can finish selection or cancel
            (EditorState::Selecting { .. }, EditorState::Idle) => true,
            (EditorState::Selecting { .. }, EditorState::Transforming { ..
}) => true,

            // From Transforming, we can only finish or cancel
            (EditorState::Transforming { .. }, EditorState::Idle) => true,

            // All other transitions are invalid
            _ => false,
        }
    }
}
```

```
struct EditorContext {
    state: EditorState,
    document: Document,
    renderer: Renderer,
    event_bus: EventBus,
}

// State transition events
enum StateTransitionEvent {
    BeginDrawing { tool: DrawingTool },
    EndDrawing { commit: bool },
    BeginSelection { mode: SelectionMode },
    EndSelection { commit: bool },
    BeginTransform { layer_id: LayerId },
    EndTransform { commit: bool },
    Cancel,
}
```

Tool System

```
trait Tool {
    fn on_activate(&mut self, ctx: &mut EditorContext);
    fn on_deactivate(&mut self, ctx: &mut EditorContext);
    fn update(&mut self, ctx: &mut EditorContext, input: &InputState);
    fn render(&self, ctx: &EditorContext, painter: &Painter);
}

// Tool-specific state containers
struct BrushState {
    color: Color32,
    size: f32,
    pressure: f32,
    current_stroke: Option<Stroke>,
}

struct SelectionState {
    mode: SelectionMode,
    in_progress: Option<SelectionInProgress>,
    current_selection: Option<Selection>,
}

struct TransformState {
    affected_layer: LayerId,
    original_transform: Transform,
    current_transform: Transform,
    gizmo: TransformGizmo,
}

enum ToolType {
    Brush(BrushTool),
    Eraser(EraserTool),
    Selection(SelectionTool),
    Transform(TransformTool),
}
```

Event System

```
enum EditorEvent {
    ToolChanged { old: ToolType, new: ToolType },
    StateChanged { old: EditorState, new: EditorState },
    LayerChanged(LayerEvent),
    SelectionChanged(SelectionEvent),
    DocumentChanged(DocumentEvent),
}

struct EventBus {
    subscribers: Vec<Box<dyn EventHandler>>,
}

// Event handlers for specific components
trait EventHandler: Send {
    fn handle_event(&mut self, event: &EditorEvent);
}

struct ToolEventHandler {
    current_tool: ToolType,
}

struct LayerEventHandler {
    document: Document,
}

struct UndoRedoEventHandler {
    history: CommandHistory,
}
```

Command System

```
enum Command {
    SetTool(ToolType),
    BeginOperation(EditorState),
    EndOperation,
    TransformLayer { layer_id: LayerId, transform: Transform },
    AddStroke { layer_id: LayerId, stroke: Stroke },
    SetSelection { selection: Selection },
}

struct CommandContext {
    document: Document,
    current_tool: ToolType,
    event_bus: EventBus,
}

struct CommandHistory {
    undo_stack: Vec<Command>,
    redo_stack: Vec<Command>,
}
```

Implementation Phases

Phase 1: Core Infrastructure (Week 1)

Goals

- Establish foundational architecture
- Set up state management system
- Create event system backbone

Tasks

1. Create new module structure:

   ```
   src/
   ├── state/
   │   ├── mod.rs           - Re-exports and state module organization
   │   ├── editor_state.rs  - EditorState enum and transitions
   │   └── context.rs       - EditorContext implementation
   ├── event/
   │   ├── mod.rs          - Event system exports
   │   ├── bus.rs          - EventBus implementation
   │   └── events.rs       - Event type definitions
   ├── tool/
   │   ├── mod.rs          - Tool system organization
   │   ├── trait.rs        - Tool trait definition
   │   └── types/          - Individual tool implementations
   └── command/
       ├── mod.rs          - Command system organization
       └── commands.rs     - Command definitions and execution
   ```

2. Implement core state machine:

   - EditorState enum with all possible states
   - State transition validation
   - State change event emission
   - Context management for state data

   Implementation order: a. Define EditorState enum b. Implement state transition validation c. Create state change events d. Build EditorContext e. Add state persistence

3. Set up event system:

   - Event bus implementation
   - Event handlers
   - Event dispatch mechanism

   Implementation order: a. Define event types b. Create EventBus c. Implement event handlers d. Add event subscription system e. Test event propagation

4. Create basic command infrastructure:

   - Command enum
   - Command execution
   - Command history

   Implementation order: a. Define Command enum b. Create CommandContext c. Implement command execution d. Add undo/redo support e. Test command system

Phase 2: Tool Refactor (Week 2)

Goals

- Move tool logic into separate implementations
- Establish tool lifecycle
- Connect tools to event system

Tasks

1. Create tool implementations:

   ```
   src/tool/types/
   ├── brush.rs
   ├── eraser.rs
   ├── selection.rs
   └── transform.rs
   ```

2. Implement for each tool:

   - State management
   - Input handling
   - Rendering
   - Event handling

3. Tool-specific features:

   - BrushTool: Stroke management
   - SelectionTool: Selection modes
   - TransformTool: Gizmo management

4. Testing:

  - Unit tests for each tool
  - Integration tests for tool interactions

Phase 3: State Management (Week 3)

Goals

- Implement state transitions
- Move gizmo management to TransformTool
- Connect state changes to event system

Tasks

1. State Transitions:

   - Define valid state transitions
   - Implement transition guards
   - Add transition events

2. Gizmo Management:

   - Move gizmo code to TransformTool
   - Implement proper cleanup
   - Add gizmo state events

3. State Persistence:

   - Serialize state
   - Handle state recovery
   - Manage undo/redo

4. Testing:

   - State transition tests
   - Gizmo behavior tests
   - Edge case handling

Phase 4: UI Integration (Week 4)

Goals

- Update UI to work with new state machine
- Implement proper input handling
- Connect UI events to command system

Tasks

1. UI Updates:

   - Modify PaintApp to use new architecture
   - Update tool panel

    - Update layer panel

2. Input Handling:

    - Create InputState struct
    - Implement input routing
    - Add gesture support

3. Command Integration:

    - Connect UI actions to commands
    - Implement command validation
    - Add command feedback

4. Testing:

    - UI interaction tests
    - Command execution tests
    - Integration tests

Phase 5: Cleanup and Polish (Week 5)

Goals

- Remove old code
- Clean up PaintApp
- Add error handling
- Documentation

Tasks

1. Code Cleanup:

    - Remove deprecated code
    - Consolidate shared functionality
    - Optimize performance

2. Error Handling:

    - Add error types
    - Implement error recovery
    - Add user feedback

3. Documentation:

    - Add API documentation
    - Create usage examples
    - Update README

4. Final Testing:

    - Performance testing

- Memory leak checks
- Full integration tests

Migration Strategy

1. **Parallel Implementation**
    - Keep existing code working
    - Build new system alongside
    - Gradually migrate features
2. **Feature Parity Testing**
    - Test each feature in new system
    - Compare with old behavior
    - Document differences
3. **Rollout Plan**
    - Phase-by-phase testing
    - Feature flags for new system
    - Gradual user migration

Success Metrics

For each phase, we'll measure success by:

1. **State Management**
    - All state transitions are explicit and validated
    - No tool state leakage
    - Clean state serialization/deserialization
2. **Event System**
    - All state changes emit appropriate events
    - Event handlers receive expected events
    - No event cycles or cascades
3. **Command System**
    - All document modifications go through commands
    - Undo/redo works for all operations
    - Command history is properly maintained
4. **Tool System**
    - Tools properly activate/deactivate
    - Tool state is isolated
    - Tools interact properly with commands

Testing Strategy

Each phase should include:

1. **Unit Tests**

   - State transitions
   - Event propagation
   - Command execution
   - Tool operations

2. **Integration Tests**

   - Tool interactions
   - State-Command-Event flow
   - Undo/redo sequences

3. **End-to-End Tests**

   - Complete user operations
   - State persistence
   - Error handling

Success Criteria

1. **Technical**

   - Clean state transitions
   - No tool state bugs
   - Proper gizmo management
   - Improved performance

2. **User Experience**

   - No regressions
   - Consistent behavior
   - Better error feedback

3. **Code Quality**

   - Increased test coverage
   - Reduced complexity
   - Better maintainability

Risk Management

1. **Technical Risks**

   - State machine complexity
   - Performance impact
   - Migration bugs

2. **Mitigation Strategies**

  - Comprehensive testing
  - Performance benchmarking
  - Feature flags
  - Rollback plan

Timeline

- Week 1: Phase 1 - Core Infrastructure
- Week 2: Phase 2 - Tool Refactor
- Week 3: Phase 3 - State Management
- Week 4: Phase 4 - UI Integration
- Week 5: Phase 5 - Cleanup and Polish

Total Duration: 5 weeks

Future Considerations

1. **Extensibility**

   - Plugin system
   - Custom tools
   - Additional state types

2. **Performance**

   - State caching
   - Event optimization
   - Render batching

3. **Features**

   - Additional tools
   - Enhanced gizmo capabilities
   - Layer effects