**py_ver/main_all.py**

```
import sys
import os
import numpy as np
import yaml  # PyYAMLをインポート
from PySide6.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout,
                               QHBoxLayout, QMenuBar, QMenu, QLabel, QPushButton,
                               QFileDialog, QScrollArea, QSplitter, QGesture,
                               QPinchGesture, QSlider, QCheckBox, QFrame, QTextEdit,
                               QMessageBox, QDialog, QLineEdit, QToolTip)
from PySide6.QtCore import Qt, QPoint, Signal, QEvent, QSize, QMimeData, QTimer, QRect
from PySide6.QtGui import (QPixmap, QImage, QWheelEvent, QPainter, QPen, QCursor,
                           QDrag, QColor)  # QDragをQtGuiからインポート
from enum import Enum

# 共通のスタイル定義
COMMON_STYLES = """
    QWidget {
        font-size: 11px;
    }
"""

# Waypointのエクスポート/インポートフォーマット定義
WAYPOINT_FORMAT = {
    'version': '1.0',
    'format': {
        'number': 'int',
        'x': 'float',
        'y': 'float',
        'angle_degrees': 'float',
        'angle_radians': 'float'
    }
}

# 共通のレイアウト設定
LAYOUT_MARGINS = 8
WIDGET_SPACING = 5
STANDARD_HEIGHT = 25

# ピンチジェスチャーの感度調整用定数
SCALE_SENSITIVITY = 0.2

# スケール関連の定数を追加
MIN_SCALE = 0.02  # 1/50 (スライダー値1に対応)
MAX_SCALE = 2.0   # 100/50 (スライダー値100に対応)
DEFAULT_SCALE = 1.0  # 50/50 (スライダー値50に対応)

class DrawingMode(Enum):
    """描画モードを定義"""
    NONE = 0
```

```
    PEN = 1
    ERASER = 2
    WAYPOINT = 3

class Waypoint:
    """ウェイポイントを管理するクラス"""
    counter = 0

    @classmethod
    def reset_counter(cls):
        """カウンターをリセット"""
        cls.counter = 0

    def __init__(self, pixel_x, pixel_y, angle=0, name=None):
        Waypoint.counter += 1
        self.number = Waypoint.counter
        self.pixel_x = pixel_x
        self.pixel_y = pixel_y
        self.x = 0
        self.y = 0
        self.angle = angle
        self.name = name if name else f"Waypoint {self.number}"
        self.resolution = 0.05  # 解像度を保存
        self.update_display_name()
        self.attributes = {} # 属性を保存するディクショナリを追加

    def set_angle(self, angle):
        """角度を設定し、表示名を更新"""
        self.angle = angle
        self.update_display_name()

    def update_display_name(self):
        """表示名を更新"""
        degrees = int(self.angle * 180 / np.pi)  # ラジアンを度に変換
        self.display_name = f"#{self.number:02d} ({self.x:.2f}, {self.y:.2f}) {degrees}°"

    def update_metric_coordinates(self, origin_x, origin_y, resolution):
        """ピクセル座標からメートル座標を計算"""
        # 原点情報を保存
        self._origin_x = origin_x
        self._origin_y = origin_y
        self.resolution = resolution

        # 原点からの相対位置をピクセルで計算
        rel_x = self.pixel_x - origin_x
        rel_y = origin_y - self.pixel_y  # y軸は反転

        # メートル単位に変換
        self.x = rel_x * resolution
        self.y = rel_y * resolution
        self.update_display_name()
```

```
101
102     def renumber(self, new_number):
103         """ウェイポイントの番号を変更"""
104         self.number = new_number
105         self.name = f"Waypoint {self.number}"
106         self.update_display_name()
107
108     def set_position(self, x, y):
109         """ピクセル座標を更新"""
110         self.pixel_x = x
111         self.pixel_y = y
112         if hasattr(self, '_origin_x') and hasattr(self, '_origin_y'):
113             # 既存の原点情報がある場合は座標を更新
114             self.update_metric_coordinates(self._origin_x, self._origin_y, self.resolution)
115
116     def set_attribute(self, key, value):
117         """属性を設定"""
118         self.attributes[key] = value
119
120     def get_attribute(self, key, default=None):
121         """属性を取得"""
122         return self.attributes.get(key, default)
123
124 class CustomScrollArea(QScrollArea):
125     """カスタムスクロールエリアクラス
126     画像の表示領域とスクロール・ズーム機能を提供"""
127     scale_changed = Signal(float)
128
129     def __init__(self):
130         super().__init__()
131         self.setMouseTracking(True)
132         self.last_pos = None
133         self.mouse_pressed = False
134         self.drawing_mode_enabled = False  # 描画モード状態を追加
135
136         # ジェスチャー設定を1箇所に集約
137         for attr in [self, self.viewport()]:
138             attr.setAttribute(Qt.WidgetAttribute.WA_AcceptTouchEvents)
139         self.grabGesture(Qt.GestureType.PinchGesture)
140
141     def set_drawing_mode(self, enabled):
142         """描画モードの有効/無効を設定"""
143         self.drawing_mode_enabled = enabled
144         # 描画モード時はビューポートのマウストラッキングを有効化
145         self.viewport().setMouseTracking(enabled)
146
147     def mousePressEvent(self, event):
148         """マウス押下時のイベント処理
149         左クリックでドラッグ開始"""
150         if self.drawing_mode_enabled:
151             # 描画モード時はイベントを親に伝播
```

```
            event.ignore()
        else:
            # 通常モード時は既存のスクロール処理
            if event.button() == Qt.MouseButton.LeftButton:
                self.mouse_pressed = True
                self.last_pos = event.position().toPoint()  # 修正
                self.setCursor(Qt.CursorShape.ClosedHandCursor)
                event.accept()
            else:
                super().mousePressEvent(event)

    def mouseReleaseEvent(self, event):
        if self.drawing_mode_enabled:
            event.ignore()
        else:
            if event.button() == Qt.MouseButton.LeftButton:
                self.mouse_pressed = False
                self.setCursor(Qt.CursorShape.ArrowCursor)
                event.accept()
            else:
                super().mouseReleaseEvent(event)

    def mouseMoveEvent(self, event):
        """マウス移動時のイベント処理
        ドラッグによるスクロール処理を実装"""
        if self.drawing_mode_enabled:
            event.ignore()
        else:
            if self.mouse_pressed and self.last_pos:
                delta = event.position().toPoint() - self.last_pos  # 修正
                self.horizontalScrollBar().setValue(
                    self.horizontalScrollBar().value() - delta.x())
                self.verticalScrollBar().setValue(
                    self.verticalScrollBar().value() - delta.y())
                self.last_pos = event.position().toPoint()  # 修正
                event.accept()
            else:
                super().mouseMoveEvent(event)

    def wheelEvent(self, event: QWheelEvent):
        if event.modifiers() & Qt.KeyboardModifier.ControlModifier:
            # 現在のスケールを考慮して調整
            factor = 1.04 if event.angleDelta().y() > 0 else 0.96
            self.scale_changed.emit(factor)
            event.accept()
        else:
            super().wheelEvent(event)

    def event(self, event):
        # ジェスチャー処理を簡略化
        if event.type() == QEvent.Type.Gesture:
```

```
            if gesture := event.gesture(Qt.GestureType.PinchGesture):
                # ピンチジェスチャーのスケール係数をスライダーの単位に合わせて調整
                total_scale = gesture.totalScaleFactor()
                if abs(total_scale - 1.0) > 0.01:
                    # より滑らかなスケーリングのために調整
                    scale = 1.0 + ((total_scale - 1.0) * 0.05)
                    self.scale_changed.emit(scale)
                return True
        return super().event(event)

class DrawableLabel(QLabel):
    """描画可能なラベルクラス"""
    waypoint_clicked = Signal(QPoint)  # ウェイポイト追加用のシグナルを追加
    waypoint_updated = Signal(Waypoint)  # 角度更新用のシグナルを追加
    waypoint_completed = Signal(QPoint)  # 角度確定用のシグナルを追加
    mouse_position_changed = Signal(QPoint)  # マウス位置シグナルを追加
    waypoint_edited = Signal(Waypoint)  # ウェイポイント編集完了時のシグナル

    def __init__(self, parent=None):
        super().__init__(parent)
        self.drawing_enabled = False
        self.parent_viewer = None
        self.cursor_pixmap = None  # カーソル用のピクスマップ
        self.current_cursor_size = 0  # 現在のカーソルサイズ
        self.setMouseTracking(True)
        self.temp_waypoint = None  # 一時的なウェイポイント保存用
        self.is_setting_angle = False  # 角度設定中フラグ
        self.click_pos = None  # クリック位置を保存
        self.edit_mode = False  # 編集モード状態
        self.editing_waypoint = None  # 編集中のウェイポイント
        self.is_dragging = False
        self.drag_start = None
        self.last_pos = None  # Add this line
        self.is_editing_angle = False

    def set_drawing_mode(self, enabled):
        self.drawing_enabled = enabled
        if enabled:
            self.updateCursor()  # カーソルを更新
        else:
            self.setCursor(Qt.CursorShape.ArrowCursor)

    def updateCursor(self):
        """カーソルを更新"""
        if not self.parent_viewer:
            return

        # 現在のツールのサイズを取得
        size = self.parent_viewer.pen_size if self.parent_viewer.drawing_mode == DrawingMode.PEN else
self.parent_viewer.eraser_size

```

```
253         # スケールに応じてカーソルサイズを調整（実際の描画サイズと同じになるように計算）
254         pixmap_geometry = self.geometry()
255         if self.parent_viewer.drawing_layer.pixmap:
256             scale_x = pixmap_geometry.width() / self.parent_viewer.drawing_layer.pixmap.width()
257             scaled_size = int(size * scale_x)
258         else:
259             scaled_size = size
260 
261         # サイズが変更された場合のみ新しいカーソルを作成
262         if (scaled_size != self.current_cursor_size):
263             self.current_cursor_size = scaled_size
264             # カーソルサイズを実際の描画サイズに合わせて調整（*2を削除）
265             cursor_size = max(scaled_size, 8)  # カーソルの最小サイズを8ピクセルに設定
266 
267             # カーソル用のピクスマップを作成
268             self.cursor_pixmap = QPixmap(cursor_size, cursor_size)
269             self.cursor_pixmap.fill(Qt.GlobalColor.transparent)
270 
271             # 円を描画
272             painter = QPainter(self.cursor_pixmap)
273             painter.setPen(QPen(Qt.GlobalColor.black, 1))
274             painter.setBrush(Qt.BrushStyle.NoBrush)
275             painter.drawEllipse(0, 0, cursor_size-1, cursor_size-1)
276             painter.end()
277 
278             # カーソルを設定
279             cursor = QCursor(self.cursor_pixmap, cursor_size // 2, cursor_size // 2)
280             self.setCursor(cursor)
281 
282     def mouseDoubleClickEvent(self, event):
283         """ウェイポイントをダブルクリックして編集モードの切り替え"""
284         if not self.parent_viewer or self.parent_viewer.drawing_mode != DrawingMode.NONE:
285             return
286 
287         pos = event.position().toPoint()
288         pixmap_geometry = self.geometry()
289         if self.parent_viewer.drawing_layer.pixmap:
290             scale_x = self.parent_viewer.drawing_layer.pixmap.width() / pixmap_geometry.width()
291             scale_y = self.parent_viewer.drawing_layer.pixmap.height() / pixmap_geometry.height()
292             x = int(pos.x() * scale_x)
293             y = int(pos.y() * scale_y)
294 
295             if self.edit_mode and self.editing_waypoint:
296                 # 編集モードを終了
297                 self.edit_mode = False
298                 self.editing_waypoint = None
299                 self.setCursor(Qt.CursorShape.ArrowCursor)
300                 if self.parent_viewer:
301                     self.parent_viewer.update_display()  # 表示を更新して赤色に戻す
302             else:
303                 # クリックされた位置にあるウェイポイントを探す
```

```
            for waypoint in self.parent_viewer.waypoints:
                if abs(waypoint.pixel_x - x) < 15 and abs(waypoint.pixel_y - y) < 15:
                    self.edit_mode = True
                    self.editing_waypoint = waypoint
                    self.setCursor(Qt.CursorShape.SizeAllCursor)
                    # ステータスメッセージを表示
                    if self.parent_viewer:
                        self.parent_viewer.show_edit_message("ドラッグで移動、Shift+ドラッグで角度を変更")
                        self.parent_viewer.update_display()  # 表示を更新して青色に変更
                    break

    def mousePressEvent(self, event):
        if self.drawing_enabled and self.parent_viewer:
            if self.parent_viewer.drawing_mode == DrawingMode.WAYPOINT:
                if event.button() == Qt.MouseButton.LeftButton:
                    pos = event.position().toPoint()
                    self.click_pos = pos  # クリック位置を保存
                    self.is_setting_angle = True
                    self.waypoint_clicked.emit(pos)
            else:
                pos = event.position().toPoint()
                self.last_pos = pos
                self.parent_viewer.draw_line(pos, pos)  # 点を描画
        elif self.edit_mode and self.editing_waypoint:
            pos = event.position().toPoint()  # 修正
            pixmap_geometry = self.geometry()
            if self.parent_viewer.drawing_layer.pixmap:
                scale_x = self.parent_viewer.drawing_layer.pixmap.width() / pixmap_geometry.width()
                scale_y = self.parent_viewer.drawing_layer.pixmap.height() / pixmap_geometry.height()
                x = int(pos.x() * scale_x)
                y = int(pos.y() * scale_y)

                if event.modifiers() & Qt.KeyboardModifier.ShiftModifier:
                    # Shiftキーが押されている場合は角度編集モード
                    self.is_editing_angle = True
                    self.editing_start_pos = pos
                else:
                    # 通常クリックは位置の移動
                    self.editing_waypoint.set_position(x, y)
                    if self.parent_viewer:
                        self.parent_viewer.update_display()
                        self.parent_viewer.waypoint_edited.emit(self.editing_waypoint)
        else:
            super().mousePressEvent(event)

    def mouseMoveEvent(self, event):
        """マウス移動時のイベント処理"""
        # マウス位置を通知
        pos = event.position().toPoint()
        self.mouse_position_changed.emit(pos)

```

```
        if self.parent_viewer and self.parent_viewer.waypoints:
            pixmap_geometry = self.geometry()
            if self.parent_viewer.pgm_layer.pixmap:
                scale_x = self.parent_viewer.pgm_layer.pixmap.width() / pixmap_geometry.width()
                scale_y = self.parent_viewer.pgm_layer.pixmap.height() / pixmap_geometry.height()

                # スケールを考慮した座標に変換
                x = int(pos.x() * scale_x)
                y = int(pos.y() * scale_y)

                # ホバー検出範囲を設定
                hover_range = 15  # ピクセル単位での検出範囲

                for waypoint in self.parent_viewer.waypoints:
                    if abs(waypoint.pixel_x - x) < hover_range and abs(waypoint.pixel_y - y) < hover_range:
                        if waypoint.attributes:
                            tooltip = "<b>Actions:</b><br>"
                            for key, value in waypoint.attributes.items():
                                tooltip += f"{key}: {value}"
                                # 最後の要素以外には改行を追加
                                if key != list(waypoint.attributes.keys())[-1]:
                                    tooltip += "<br>"
                            # QPointFからQPointに変換
                            global_pos = QPoint(
                                int(event.globalPosition().x()),
                                int(event.globalPosition().y())
                            )
                            QToolTip.showText(global_pos, tooltip.strip())
                            return

                QToolTip.hideText()

        # 既存の描画モード処理を続行
        if self.drawing_enabled and self.parent_viewer:
            if self.is_setting_angle and self.parent_viewer.drawing_mode == DrawingMode.WAYPOINT:
                if self.temp_waypoint and self.click_pos:
                    # プレビュー用の角度計算（Y軸を反転）
                    dx = pos.x() - self.click_pos.x()
                    dy = -(pos.y() - self.click_pos.y())  # Y軸を反転
                    angle = np.arctan2(dy, dx)
                    self.temp_waypoint.set_angle(angle)
                    self.waypoint_updated.emit(self.temp_waypoint)
            elif self.last_pos:
                self.parent_viewer.draw_line(self.last_pos, pos)  # 線を描画
                self.last_pos = pos
                self.updateCursor()  # マウス移動時にカーソルを更新
        elif self.edit_mode and self.editing_waypoint:
            pos = event.position().toPoint()  # 修正
            pixmap_geometry = self.geometry()
            if self.parent_viewer.drawing_layer.pixmap:
                scale_x = self.parent_viewer.drawing_layer.pixmap.width() / pixmap_geometry.width()
```

```
            scale_y = self.parent_viewer.drawing_layer.pixmap.height() / pixmap_geometry.height()
            x = int(pos.x() * scale_x)
            y = int(pos.y() * scale_y)

            if self.is_editing_angle:
                # 角度の計算
                dx = pos.x() - self.editing_start_pos.x()
                dy = -(pos.y() - self.editing_start_pos.y())  # Y軸を反転
                angle = np.arctan2(dy, dx)
                self.editing_waypoint.set_angle(angle)
            else:
                # 位置の更新
                self.editing_waypoint.set_position(x, y)

            if self.parent_viewer:
                self.parent_viewer.update_display()
                self.parent_viewer.waypoint_edited.emit(self.editing_waypoint)
        else:
            super().mouseMoveEvent(event)

    def mouseReleaseEvent(self, event):
        if self.drawing_enabled:
            if self.is_setting_angle and self.click_pos:
                # 最終的な角度計算（Y軸を反転）
                current_pos = event.position().toPoint()  # 修正
                dx = current_pos.x() - self.click_pos.x()
                dy = -(current_pos.y() - self.click_pos.y())  # Y軸を反転
                if self.temp_waypoint:
                    final_angle = np.arctan2(dy, dx)
                    self.temp_waypoint.set_angle(final_angle)
                    self.waypoint_updated.emit(self.temp_waypoint)
                self.is_setting_angle = False
                self.click_pos = None
                self.temp_waypoint = None
            self.last_pos = None
        elif self.edit_mode and self.editing_waypoint:
            self.is_editing_angle = False
            if self.parent_viewer:
                self.parent_viewer.update_display()
                self.parent_viewer.waypoint_edited.emit(self.editing_waypoint)
        else:
            super().mouseReleaseEvent(event)

    def contextMenuEvent(self, event):
        """右クリックメニューの表示"""
        if not self.parent_viewer:
            return

        # クリックされた位置のウェイポイントを探す
        pos = event.pos()
        pixmap_geometry = self.geometry()
```

```
        if self.parent_viewer.drawing_layer.pixmap:
            scale_x = self.parent_viewer.drawing_layer.pixmap.width() / pixmap_geometry.width()
            scale_y = self.parent_viewer.drawing_layer.pixmap.height() / pixmap_geometry.height()
            x = int(pos.x() * scale_x)
            y = int(pos.y() * scale_y)

            for waypoint in self.parent_viewer.waypoints:
                if abs(waypoint.pixel_x - x) < 15 and abs(waypoint.pixel_y - y) < 15:
                    menu = QMenu(self)
                    menu.setStyleSheet("""
                        QMenu {
                            background-color: #0078d7;
                            color: white;
                        }
                        QMenu::item {
                            background-color: transparent;
                            padding: 5px 20px;
                            color: white;
                            font-weight: bold;
                        }
                        QMenu::item:selected {
                            background-color: #0058a3;
                            color: white;
                        }
                    """)
                    edit_action = menu.addAction("Add Actions")
                    action = menu.exec(event.globalPos())

                    # ウェイポイントの編集ダイアログを表示
                    if action == edit_action:
                        dialog = AttributeDialog(waypoint, format_manager.get_format(), self)
                        if dialog.exec() == QDialog.DialogCode.Accepted:
                            waypoint.attributes = dialog.get_attributes()
                            self.parent_viewer.update_display()
                            if self.parent_viewer:
                                self.parent_viewer.waypoint_edited.emit(waypoint)
                    break

class Layer(QWidget):
    """レイヤークラス"""
    changed = Signal()  # レイヤーの状態変更通知用シグナル

    def __init__(self, name, visible=True):
        super().__init__()
        self.name = name
        self.visible = visible
        self.pixmap = None
        self.opacity = 1.0

    def set_visible(self, visible):
        if (self.visible != visible):
```

```
508             self.visible = visible
509             self.changed.emit()
510 
511     def set_opacity(self, opacity):
512         new_opacity = max(0.0, min(1.0, opacity))
513         if (self.opacity != new_opacity):
514             self.opacity = new_opacity
515             self.changed.emit()
516 
517 class ImageViewer(QWidget):
518     """画像表示用ウィジェット
519     PGM画像の表示とズーム機能を管理"""
520     # スケール変更通知用のシグナルを追加
521     scale_changed = Signal(float)
522     layer_changed = Signal()  # レイヤーの状態変更通知用
523     waypoint_added = Signal(Waypoint)  # ウェイポイント追加通知用のシグナル
524     waypoint_removed = Signal(int)  # 削除シグナルを追加
525     waypoint_edited = Signal(Waypoint)  # 編集完了シグナルを追加
526     history_changed = Signal(bool, bool)  # (can_undo, can_redo)
527 
528     def __init__(self):
529         super().__init__()
530         self.scale_factor = 1.0  # 画像の拡大率
531         self.drawing_mode = DrawingMode.NONE
532         self.last_point = None
533         self.pen_color = Qt.GlobalColor.black
534         self.pen_size = 2      # デフォルトのペンサイズ
535         self.eraser_size = 10  # デフォルトの消しゴムサイズ
536         self.is_drawing = False  # 描画中フラグを追加
537         self.current_drawing_points = []  # 現在の描画ストロークを保存
538 
539         # スクロールエリアを最初に初期化
540         self.scroll_area = CustomScrollArea()
541         self.scroll_area.setWidgetResizable(True)
542         self.scroll_area.setMinimumSize(600, 400)
543         self.scroll_area.setStyleSheet("QScrollArea { border: 2px solid #ccc; background-color: white; }")
544 
545         # 座標表示用ラベルを初期化
546         self.coord_label = QLabel(self.scroll_area.viewport())
547         self.coord_label.setStyleSheet("""
548             QLabel {
549                 background-color: rgba(255, 255, 255, 0.8);
550                 border: 1px solid #ccc;
551                 border-radius: 3px;
552                 padding: 3px 5px;
553                 font-size: 12px;
554                 font-family: monospace;
555                 min-height: 40px;
556                 min-width: 150px;
557             }
558         """)
```

```
559         self.coord_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
560         self.coord_label.hide()
561 
562         # レイヤー管理の初期化
563         self.layers = []
564         self.active_layer = None
565 
566         # 基本レイヤーの作成とシグナル接続
567         self.pgm_layer = Layer("PGM Layer")
568         self.drawing_layer = Layer("Drawing Layer")
569         self.waypoint_layer = Layer("Waypoint Layer")
570         self.origin_layer = Layer("Origin Layer")
571         self.path_layer = Layer("Path Layer")
572         self.layers = [
573             self.pgm_layer,      # 1. PGM画像（最下層）
574             self.drawing_layer,  # 2. ペンと消しゴムの描画
575             self.path_layer,     # 3. パス
576             self.waypoint_layer, # 4. ウェイポイント
577             self.origin_layer    # 5. 原点（最上層）
578         ]
579         self.active_layer = self.drawing_layer
580 
581         for layer in self.layers:
582             layer.changed.connect(self.on_layer_changed)
583 
584         self.waypoints = []
585         self.waypoint_size = 15
586         self.show_grid = False
587         self.grid_size = 50
588         self.origin_point = None
589         self.resolution = 0.05
590 
591         # 各コンポーネントの設定
592         self.setup_display()
593         self.setup_scroll_area()
594         self.setup_drawing_tools()
595 
596         # シグナルを接続
597         self.pgm_display.waypoint_edited.connect(self.handle_waypoint_edited)
598         self.scroll_area.scale_changed.connect(self.handle_scale_change)
599 
600         # 履歴管理用の変数を追加
601         self.history = []  # 操作履歴
602         self.current_index = -1  # 現在の履歴インデックス
603         self.max_history = 50  # 最大履歴数
604 
605     def setup_display(self):
606         """画像表示用ラベルの設定を集約"""
607         self.pgm_display = DrawableLabel()
608         self.pgm_display.parent_viewer = self
609         self.pgm_display.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

```
        self.pgm_display.setStyleSheet("background-color: white;")
        self.pgm_display.waypoint_clicked.connect(self.add_waypoint)
        self.pgm_display.waypoint_updated.connect(self.update_waypoint)
        self.pgm_display.mouse_position_changed.connect(self.update_mouse_position)

        # ステータスメッセージ用のラベルを設定
        self.status_label = QLabel(self.scroll_area.viewport())
        self.status_label.setStyleSheet("""
            QLabel {
                background-color: rgba(0, 0, 0, 0.7);
                color: white;
                padding: 8px 16px;
                border-radius: 4px;
                font-size: 12px;
            }
        """)
        self.status_label.hide()

        # スクロールエリアにpgm_displayを設定
        self.scroll_area.setWidget(self.pgm_display)

    def setup_scroll_area(self):
        """スクロールエリアの設定を集約"""
        self.scroll_area.scale_changed.connect(self.handle_scale_change)

        self.setLayout(QVBoxLayout())
        self.layout().addWidget(self.scroll_area)

        # 座標表示用ラベルをスクロールエリアのビューポートの子として設定
        self.coord_label.setParent(self.scroll_area.viewport())
        self.coord_label.hide()  # 初期状態では非表示

        # ステータスメッセージ用のラベルをスクロールエリアのビューポートの子として設定
        self.status_label = QLabel()
        self.status_label.setParent(self.scroll_area.viewport())
        self.status_label.setStyleSheet("""
            QLabel {
                background-color: rgba(0, 0, 0, 0.7);
                color: white;
                padding: 8px;
                border-radius: 4px;
                font-size: 12px;
            }
        """)
        self.status_label.hide()

        # スクロールエリアのリサイズイベントをオーバーライド
        original_resize_event = self.scroll_area.resizeEvent
        def new_resize_event(event):
            original_resize_event(event)
            # 座標ラベルを右上に配置（右端から30ピクセル離す）
```

```
            label_width = 150
            label_height = 40  # min-heightに合わせて調整
            new_x = self.scroll_area.viewport().width() - label_width - 30  # 30ピクセル左に移動
            new_y = 10  # 上端から10ピクセルの位置
            self.coord_label.setGeometry(new_x, new_y, label_width, label_height)

            # ステータスメッセージを下部中央に配置
            status_width = 300
            status_height = 30
            status_x = (self.scroll_area.viewport().width() - status_width) // 2
            status_y = self.scroll_area.viewport().height() - status_height - 10  # 下部に10ピクセルのマージン
            self.status_label.setGeometry(status_x, status_y, status_width, status_height)

            self.coord_label.raise_()
            self.status_label.raise_()
        self.scroll_area.resizeEvent = new_resize_event

    def show_edit_message(self, message):
        """編集時のヘルプメッセージを表示"""
        # メッセージの表示位置を調整（ビューアーの中央下）
        self.status_label.setText(message)
        self.status_label.adjustSize()

        # スクロールエリアのビューポート内での位置を計算
        viewport = self.scroll_area.viewport()
        x = (viewport.width() - self.status_label.width()) // 2
        y = viewport.height() - self.status_label.height() - 20  # 下端から20ピクセル上

        self.status_label.move(x, y)
        self.status_label.show()
        self.status_label.raise_()  # 最前面に表示

        # 3秒後にメッセージを非表示
        QTimer.singleShot(3000, self.status_label.hide)

    def setup_drawing_tools(self):
        """描画ツールの設定"""
        tools_layout = QVBoxLayout()

        # ボタンのレイアウト
        buttons_layout = QHBoxLayout()

        # ペンボタン
        self.pen_button = QPushButton("ペン")
        self.pen_button.setCheckable(True)
        self.pen_button.clicked.connect(lambda: self.set_drawing_mode(DrawingMode.PEN))

        # 消しゴムボタン
        self.eraser_button = QPushButton("消しゴム")
        self.eraser_button.setCheckable(True)
        self.eraser_button.clicked.connect(lambda: self.set_drawing_mode(DrawingMode.ERASER))
```

```
712
713         # ウェイポイントボタンを追加
714         self.waypoint_button = QPushButton("ウェイポイント")
715         self.waypoint_button.setCheckable(True)
716         self.waypoint_button.clicked.connect(lambda: self.set_drawing_mode(DrawingMode.WAYPOINT))
717
718         buttons_layout.addWidget(self.pen_button)
719         buttons_layout.addWidget(self.eraser_button)
720         buttons_layout.addWidget(self.waypoint_button)
721
722         # スライダーのレイアウト
723         sliders_layout = QHBoxLayout()
724
725         # ペンの太さスライダー
726         pen_slider_layout = QVBoxLayout()
727         pen_slider_label = QLabel("ペンの太さ")
728         self.pen_slider = QSlider(Qt.Orientation.Horizontal)
729         self.pen_slider.setRange(1, 20)
730         self.pen_slider.setValue(self.pen_size)
731         self.pen_slider.valueChanged.connect(self.set_pen_size)
732         pen_slider_layout.addWidget(pen_slider_label)
733         pen_slider_layout.addWidget(self.pen_slider)
734
735         # 消しゴムの太さスライダー
736         eraser_slider_layout = QVBoxLayout()
737         eraser_slider_label = QLabel("消しゴムの太さ")
738         self.eraser_slider = QSlider(Qt.Orientation.Horizontal)
739         self.eraser_slider.setRange(5, 50)
740         self.eraser_slider.setValue(self.eraser_size)
741         self.eraser_slider.valueChanged.connect(self.set_eraser_size)
742         eraser_slider_layout.addWidget(eraser_slider_label)
743         eraser_slider_layout.addWidget(self.eraser_slider)
744
745         # スライダーをレイアウトに追加
746         sliders_layout.addLayout(pen_slider_layout)
747         sliders_layout.addLayout(eraser_slider_layout)
748
749         # メインレイアウトに追加
750         tools_layout.addLayout(buttons_layout)
751         tools_layout.addLayout(sliders_layout)
752         self.layout().insertLayout(0, tools_layout)
753
754     def set_pen_size(self, size):
755         """ペンの太さを設定"""
756         self.pen_size = size
757         if self.drawing_mode == DrawingMode.PEN:
758             self.pgm_display.updateCursor()
759
760     def set_eraser_size(self, size):
761         """消しゴムの太さを設定"""
762         self.eraser_size = size
```

```
763         if self.drawing_mode == DrawingMode.ERASER:
764             self.pgm_display.updateCursor()
765 
766     def set_drawing_mode(self, mode):
767         """描画モードの切り替え"""
768         # 同じモードを選択した場合は描画モードを解除
769         if self.drawing_mode == mode:
770             self.drawing_mode = DrawingMode.NONE
771             self.pen_button.setChecked(False)
772             self.eraser_button.setChecked(False)
773             self.waypoint_button.setChecked(False)
774             self.pgm_display.set_drawing_mode(False)
775             self.scroll_area.set_drawing_mode(False)
776             return
777 
778         # 異なるモードを選択した場合は描画モードを変更
779         self.drawing_mode = mode
780         self.pen_button.setChecked(mode == DrawingMode.PEN)
781         self.eraser_button.setChecked(mode == DrawingMode.ERASER)
782         self.waypoint_button.setChecked(mode == DrawingMode.WAYPOINT)
783 
784         # ラベルの描画モードを設定
785         self.pgm_display.set_drawing_mode(mode != DrawingMode.NONE)
786         # スクロールエリアの描画モードを設定
787         self.scroll_area.set_drawing_mode(mode != DrawingMode.NONE)
788         # カーソルを更新
789         if (mode != DrawingMode.NONE):
790             if (mode == DrawingMode.WAYPOINT):
791                 self.pgm_display.setCursor(Qt.CursorShape.CrossCursor)
792             else:
793                 self.pgm_display.updateCursor()
794 
795     def draw_line(self, start_pos, end_pos):
796         """2点間に線を描画"""
797         if not self.drawing_layer.pixmap or self.drawing_mode == DrawingMode.NONE:
798             return
799 
800         # 描画前の状態を保存
801         old_pixmap = self.drawing_layer.pixmap.copy()
802 
803         pixmap_geometry = self.pgm_display.geometry()
804         scale_x = self.drawing_layer.pixmap.width() / pixmap_geometry.width()
805         scale_y = self.drawing_layer.pixmap.height() / pixmap_geometry.height()
806 
807         scaled_start = QPoint(int(start_pos.x() * scale_x), int(start_pos.y() * scale_x))
808         scaled_end = QPoint(int(end_pos.x() * scale_x), int(end_pos.y() * scale_x))
809 
810         # スケールに応じて描画サイズを調整
811         scaled_pen_size = int(self.pen_size * scale_x)
812         scaled_eraser_size = int(self.eraser_size * scale_x)
813 
```

```
        painter = QPainter(self.drawing_layer.pixmap)
        if self.drawing_mode == DrawingMode.PEN:
            painter.setPen(QPen(self.pen_color, scaled_pen_size, Qt.PenStyle.SolidLine, Qt.PenCapStyle.RoundCap, Qt.PenJoinStyle.RoundJoin))
        else:  # ERASER
            # 消しゴムを白色に変更
            painter.setPen(QPen(Qt.GlobalColor.white, scaled_eraser_size, Qt.PenStyle.SolidLine, Qt.PenCapStyle.RoundCap, Qt.PenJoinStyle.RoundJoin))

        painter.drawLine(scaled_start, scaled_end)
        painter.end()

        self.update_display()

        # 描画後の状態を履歴に追加
        self.add_to_history({
            'type': 'draw',
            'old_pixmap': old_pixmap,
            'new_pixmap': self.drawing_layer.pixmap.copy()
        })

    def mousePressEvent(self, event):
        if self.drawing_mode != DrawingMode.NONE:
            self.last_point = event.position().toPoint()  # 修正
            self.draw_line(event.position().toPoint(), event.position().toPoint())  # 修正
            event.accept()
        else:
            super().mousePressEvent(event)

    def mouseMoveEvent(self, event):
        if self.drawing_mode != DrawingMode.NONE and self.last_point:
            current_pos = event.position().toPoint()  # 修正
            self.draw_line(self.last_point, current_pos)
            self.last_point = current_pos
            event.accept()
        else:
            super().mouseMoveEvent(event)

    def mouseReleaseEvent(self, event):
        if self.drawing_mode != DrawingMode.NONE:
            self.last_point = None
            event.accept()
        else:
            super().mouseReleaseEvent(event)

    def load_image(self, img_array, width, height):
        """PGM画像データを読み込んでPGMレイヤーに設定"""
        bytes_per_line = width
        q_img = QImage(img_array.data, width, height, bytes_per_line,
                       QImage.Format.Format_Grayscale8)
        self.pgm_layer.pixmap = QPixmap.fromImage(q_img)
```

```
        self.drawing_layer.pixmap = QPixmap(self.pgm_layer.pixmap.size())
        self.drawing_layer.pixmap.fill(Qt.GlobalColor.transparent)
        self.update_display()
        self.coord_label.show()  # 画像読み込み時に座標表示を有効化

    def zoom_in(self):
        self.scale_factor *= 1.2
        self.update_display()

    def zoom_out(self):
        self.scale_factor /= 1.2
        self.update_display()

    def zoom_reset(self):
        self.scale_factor = 1.0
        self.update_display()

    def handle_scale_change(self, factor):
        """ジェスチャーやホイールによるスケール変更を処理"""
        new_scale = self.scale_factor * factor
        if MIN_SCALE <= new_scale <= MAX_SCALE:
            self.scale_factor = new_scale
            self.update_display()
            self.scale_changed.emit(self.scale_factor)
            # スケール変更時にカーソルを更新
            if self.drawing_mode != DrawingMode.NONE:
                self.pgm_display.updateCursor()

    def add_waypoint(self, pos):
        """ウェイポイントを追加"""
        if not self.pgm_layer.pixmap:
            return

        # 表示サイズからピクセル座標に変換
        pixmap_geometry = self.pgm_display.geometry()
        if self.pgm_layer.pixmap:
            scale_x = self.pgm_layer.pixmap.width() / pixmap_geometry.width()
            scale_y = self.pgm_layer.pixmap.height() / pixmap_geometry.height()
            x = int(pos.x() * scale_x)
            y = int(pos.y() * scale_y)
        else:
            x = pos.x()
            y = pos.y()

        waypoint = Waypoint(x, y)

        # 原点が設定されている場合は、メートル座標を計算
        if self.origin_point:
            origin_x, origin_y = self.origin_point

            waypoint.update_metric_coordinates(origin_x, origin_y, self.resolution)
```

```
913
914             self.waypoints.append(waypoint)
915             self.pgm_display.temp_waypoint = waypoint
916
917             # ウェイポイントレイヤーの初期化（必要な場合）
918             if not self.waypoint_layer.pixmap:
919                 self.waypoint_layer.pixmap = QPixmap(self.pgm_layer.pixmap.size())
920                 self.waypoint_layer.pixmap.fill(Qt.GlobalColor.transparent)
921
922             self.waypoint_added.emit(waypoint)
923             self.update_display()
924
925             # 履歴に追加
926             self.add_to_history({
927                 'type': 'waypoint_add',
928                 'waypoint': waypoint
929             })
930
931         def update_waypoint(self, waypoint):
932             """ウェイポイントの更新（角度変更時）"""
933             self.update_display()
934             # 対応するラベルを更新するためにシグナルを再発行
935             self.waypoint_added.emit(waypoint)
936
937         def update_display(self):
938             """複数レイヤーを合成して表示"""
939             if not self.pgm_layer.pixmap:
940                 return
941
942             # 合成用の新しいピクスマップを作成
943             result = QPixmap(self.pgm_layer.pixmap.size())
944             result.fill(Qt.GlobalColor.white)
945
946             painter = QPainter(result)
947
948             # 1. PGMレイヤーを描画（最下層）
949             if (self.pgm_layer.visible and self.pgm_layer.pixmap):
950                 painter.setOpacity(self.pgm_layer.opacity)
951                 painter.drawPixmap(0, 0, self.pgm_layer.pixmap)
952
953             # 2. グリッドの描画
954             if self.show_grid:
955                 painter.setOpacity(0.3)
956                 pen = QPen(Qt.GlobalColor.gray)
957                 pen.setStyle(Qt.PenStyle.DashLine)
958                 painter.setPen(pen)
959
960                 for x in range(0, result.width(), self.grid_size):
961                     painter.drawLine(x, 0, x, result.height())
962                 for y in range(0, result.height(), self.grid_size):
```

```
            painter.drawLine(0, y, result.width(), y)

        # 3. 描画レイヤーを描画
        if self.drawing_layer.visible and self.drawing_layer.pixmap:
            painter.setOpacity(self.drawing_layer.opacity)
            painter.drawPixmap(0, 0, self.drawing_layer.pixmap)

        # 4. パスレイヤーを描画
        if self.path_layer.visible and self.path_layer.pixmap:
            painter.setOpacity(self.path_layer.opacity)
            painter.drawPixmap(0, 0, self.path_layer.pixmap)

        # 5. ウェイポイントレイヤーを描画
        if self.waypoints and self.waypoint_layer.visible:
            painter.setOpacity(self.waypoint_layer.opacity)
            painter.setRenderHint(QPainter.RenderHint.Antialiasing)

            for waypoint in self.waypoints:
                x, y = waypoint.pixel_x, waypoint.pixel_y
                size = self.waypoint_size

                # 編集中のウェイポイントは特別な表示
                is_editing = (self.pgm_display.edit_mode and
                              self.pgm_display.editing_waypoint and
                              self.pgm_display.editing_waypoint.number == waypoint.number)

                # 編集中は青色で表示
                color = QColor(0, 120, 255, 255) if is_editing else QColor(255, 0, 0, 255)
                size_multiplier = 1.2 if is_editing else 1.0

                # 矢印の描画
                pen = QPen(color)
                pen.setWidth(3)
                painter.setPen(pen)

                adjusted_size = size * size_multiplier
                angle_line_length = adjusted_size * 3
                end_x = x + int(angle_line_length * np.cos(waypoint.angle))
                end_y = y - int(angle_line_length * np.sin(waypoint.angle))
                painter.drawLine(x, y, end_x, end_y)

                # 矢印の先端
                arrow_size = adjusted_size // 2
                arrow_angle1 = waypoint.angle + np.pi * 3/4
                arrow_angle2 = waypoint.angle - np.pi * 3/4

                arrow_x1 = end_x + int(arrow_size * np.cos(arrow_angle1))
                arrow_y1 = end_y - int(arrow_size * np.sin(arrow_angle1))
                arrow_x2 = end_x + int(arrow_size * np.cos(arrow_angle2))
                arrow_y2 = end_y - int(arrow_size * np.sin(arrow_angle2))

```

```
            painter.drawLine(end_x, end_y, arrow_x1, arrow_y1)
            painter.drawLine(end_x, end_y, arrow_x2, arrow_y2)

            # 円を描画
            painter.setPen(Qt.PenStyle.NoPen)
            painter.setBrush(color)
            painter.drawEllipse(x - adjusted_size, y - adjusted_size,
                                adjusted_size * 2, adjusted_size * 2)

            # 番号を描画
            painter.setPen(QColor(255, 255, 255, 230))
            font = self.font()
            font.setPointSize(19)
            font.setBold(True)
            painter.setFont(font)
            number_text = str(waypoint.number)
            font_metrics = painter.fontMetrics()
            text_width = font_metrics.horizontalAdvance(number_text)
            text_height = font_metrics.height()
            text_x = x - text_width // 2
            text_y = y + text_height // 3
            painter.drawText(text_x, text_y, number_text)

            # 属性の数を描画（右上に小さく表示）
            num_attributes = len(waypoint.attributes)
            if (num_attributes > 0):
                painter.setPen(QColor(255, 255, 255))
                font.setPointSize(10)
                font.setBold(True)
                painter.setFont(font)
                attr_text = str(num_attributes)
                attr_x = x + adjusted_size - 5
                attr_y = y - adjusted_size + 5
                # 背景円を描画
                painter.setBrush(QColor(50, 50, 50, 200))
                painter.drawEllipse(attr_x - 8, attr_y - 12, 16, 16)
                # 数字を描画
                painter.drawText(attr_x - 3, attr_y, attr_text)

            # ホバー時のツールチップ領域を設定
            hover_rect = QRect(
                x - adjusted_size,
                y - adjusted_size,
                adjusted_size * 2,
                adjusted_size * 2
            )
            waypoint.hover_rect = hover_rect  # 後でホバー判定に使用

        # 6. 原点レイヤーを描画（最上層）
        if self.origin_layer.visible and self.origin_layer.pixmap:
            painter.setOpacity(self.origin_layer.opacity)
```

```
            painter.drawPixmap(0, 0, self.origin_layer.pixmap)

        painter.end()

        # スケーリングして表示
        new_size = QSize(
            int(result.width() * self.scale_factor),
            int(result.height() * self.scale_factor)
        )

        scaled_pixmap = result.scaled(
            new_size,
            Qt.AspectRatioMode.KeepAspectRatio,
            Qt.TransformationMode.SmoothTransformation  # スムージングを有効化
        )

        self.pgm_display.setPixmap(scaled_pixmap)
        self.pgm_display.adjustSize()

    def on_layer_changed(self):
        """レイヤーの状態が変更された時の処理"""
        self.update_display()
        self.layer_changed.emit()

    def remove_waypoint(self, number):
        """ウェイポイントを削除"""
        # 削除対象のウェイポイントを除外
        self.waypoints = [wp for wp in self.waypoints if wp.number != number]

        # ナンバリングを振り直し
        Waypoint.reset_counter()

        # 一旦右パネルのリストをクリア
        self.waypoint_removed.emit(number)

        # ウェイポイントを順番に振り直してUIを更新
        for wp in self.waypoints:
            Waypoint.counter += 1
            wp.renumber(Waypoint.counter)
            self.waypoint_added.emit(wp)

        self.update_display()

    def remove_all_waypoints(self):
        """全てのウェイポイントを削除"""
        self.waypoints.clear()
        Waypoint.reset_counter()
        self.waypoint_removed.emit(-1)  # 特別な値-1で全削除を通知
        self.update_display()

    def reorder_waypoints(self, source_number, target_number):
```

```
        """ウェイポイントの順序を変更"""
        if not self.waypoints:
            return

        # 対象のウェイポイントを見つける
        source_wp = next((wp for wp in self.waypoints if wp.number == source_number), None)
        if not source_wp:
            return

        # 現在のインデックスを取得
        source_index = self.waypoints.index(source_wp)
        target_index = next((i for i, wp in enumerate(self.waypoints)
                             if wp.number == target_number), -1)

        if target_index == -1:
            return

        # リストから削除して新しい位置に挿入
        self.waypoints.pop(source_index)

        # ターゲットの位置に挿入
        if source_index < target_index:
            # 上から下にドラッグする場合
            self.waypoints.insert(target_index, source_wp)
        else:
            # 下から上にドラッグする場合
            self.waypoints.insert(target_index, source_wp)

        # 番号を振り直し
        Waypoint.reset_counter()

        # UIを更新するために一旦全てのウェイポイントを削除
        self.waypoint_removed.emit(-1)

        # ウェイポイントを順番に振り直してUIを更新
        for wp in self.waypoints:
            Waypoint.counter += 1
            wp.renumber(Waypoint.counter)
            self.waypoint_added.emit(wp)

        self.update_display()

    def toggle_grid(self):
        """グリッド表示の切り替え"""
        self.show_grid = not self.show_grid
        self.update_display()

    def update_mouse_position(self, pos):
        """マウス位置の更新とラベル表示"""
        if not self.pgm_layer.pixmap or not self.origin_point:
            return
```

```
1167
1168            # 表示座標からピクセル座標に変換
1169            pixmap_geometry = self.pgm_display.geometry()
1170            scale_x = self.pgm_layer.pixmap.width() / pixmap_geometry.width()
1171            scale_y = self.pgm_layer.pixmap.height() / pixmap_geometry.height()
1172
1173            pixel_x = int(pos.x() * scale_x)
1174            pixel_y = int(pos.y() * scale_y)
1175
1176            # 原点からの相対位置を計算
1177            origin_x, origin_y = self.origin_point
1178            rel_x = (pixel_x - origin_x) * self.resolution
1179            rel_y = (origin_y - pixel_y) * self.resolution  # y軸は反転
1180
1181            # 座標を表示（ピクセル座標と相対座標）
1182            self.coord_label.setText(f"Pixel: ({pixel_x}, {pixel_y})\nMetric: ({rel_x:.2f}, {rel_y:.2f})")
1183            self.coord_label.show()
1184
1185        def load_yaml_file(self, file_path):
1186            """YAMLファイルを読み込みorigin点を設定"""
1187            try:
1188                with open(file_path, 'r') as f:
1189                    yaml_data = yaml.safe_load(f)
1190
1191                # YAMLファイルから直接originとresolutionを読み取る
1192                if 'origin' in yaml_data:
1193                    origin = yaml_data['origin']
1194                    if len(origin) >= 2:
1195                        # 解像度を保存
1196                        self.resolution = float(yaml_data.get('resolution', 0.05))
1197                        x_pixel = int(-origin[0] / self.resolution)
1198                        y_pixel = int(-origin[1] / self.resolution)
1199
1200                        if self.pgm_layer.pixmap:
1201                            height = self.pgm_layer.pixmap.height()
1202                            y_pixel = height - y_pixel
1203
1204                        self.origin_point = (x_pixel, y_pixel)
1205                        self.draw_origin_point()
1206
1207                        # 既存のウェイポイントの座標を更新
1208                        self.update_all_waypoint_coordinates()
1209                        print(f"Origin point set to: {self.origin_point} (resolution: {self.resolution})")
1210                        print(f"Original coordinates: x={origin[0]}, y={origin[1]} meters")
1211
1212            except Exception as e:
1213                print(f"Error loading YAML file: {str(e)}")
1214                import traceback
1215                traceback.print_exc()
1216
1217        def update_all_waypoint_coordinates(self):
```

```
        """全てのウェイポイントの座標を更新"""
        if not self.origin_point:
            return

        origin_x, origin_y = self.origin_point
        for waypoint in self.waypoints:
            waypoint.update_metric_coordinates(origin_x, origin_y, self.resolution)
            self.waypoint_added.emit(waypoint)  # UIを更新

    def draw_origin_point(self):
        """原点マーカーを描画"""
        if not self.origin_point or not self.pgm_layer.pixmap:
            return

        # origin_layerのピクスマップを初期化
        if not self.origin_layer.pixmap or self.origin_layer.pixmap.size() != self.pgm_layer.pixmap.size():
            self.origin_layer.pixmap = QPixmap(self.pgm_layer.pixmap.size())
            self.origin_layer.pixmap.fill(Qt.GlobalColor.transparent)

        # 既存の描画をクリア
        self.origin_layer.pixmap.fill(Qt.GlobalColor.transparent)

        # 原点マーカーを描画
        painter = QPainter(self.origin_layer.pixmap)
        painter.setRenderHint(QPainter.RenderHint.Antialiasing)

        # 原点の位置を取得
        x, y = self.origin_point

        # クロスマーカーを描画
        marker_size = 20
        pen = QPen(QColor(255, 0, 0))  # 赤色
        pen.setWidth(3)
        painter.setPen(pen)

        # 十字マーカー
        painter.drawLine(x - marker_size, y, x + marker_size, y)
        painter.drawLine(x, y - marker_size, x, y + marker_size)

        # 円を描画
        painter.drawEllipse(x - marker_size//2, y - marker_size//2, marker_size, marker_size)

        painter.end()

        self.update_display()

    def generate_path(self):
        """ウェイポイント間のパスを生成または非表示"""
        if self.waypoints and len(self.waypoints) >= 2:

            if not self.path_layer.pixmap or self.path_layer.pixmap.size() != self.pgm_layer.pixmap.size():
```

```
1268            self.path_layer.pixmap = QPixmap(self.pgm_layer.pixmap.size())
1269
1270        # パスレイヤーをクリア
1271        self.path_layer.pixmap.fill(Qt.GlobalColor.transparent)
1272
1273        # パス生成ボタンがチェックされている場合のみパスを描画
1274        parent = self.parent()
1275        while parent and not isinstance(parent, MainWindow):
1276            parent = parent.parent()
1277
1278        if parent and parent.right_panel.generate_path_button.isChecked():
1279            if self.waypoints and len(self.waypoints) >= 2:
1280                painter = QPainter(self.path_layer.pixmap)
1281                painter.setRenderHint(QPainter.RenderHint.Antialiasing)
1282
1283                # パスのスタイル設定
1284                pen = QPen(Qt.GlobalColor.green, 3)  # 青色、太さ3
1285                pen.setStyle(Qt.PenStyle.SolidLine)
1286                painter.setPen(pen)
1287
1288                # ウェイポイントを順番に接続
1289                for i in range(len(self.waypoints) - 1):
1290                    start = self.waypoints[i]
1291                    end = self.waypoints[i + 1]
1292                    painter.drawLine(start.pixel_x, start.pixel_y, end.pixel_x, end.pixel_y)
1293
1294                painter.end()
1295
1296        self.update_display()
1297
1298    def handle_waypoint_edited(self, waypoint):
1299        """ウェイポイント編集時の処理"""
1300        # 編集前の状態を保存
1301        old_state = {
1302            'pixel_x': waypoint.pixel_x,
1303            'pixel_y': waypoint.pixel_y,
1304            'angle': waypoint.angle
1305        }
1306
1307        if self.origin_point:  # 原点が設定されている場合
1308            origin_x, origin_y = self.origin_point
1309            waypoint.update_metric_coordinates(origin_x, origin_y, self.resolution)
1310        self.waypoint_edited.emit(waypoint)
1311        self.update_display()
1312
1313        # 編集後の状態を履歴に追加
1314        new_state = {
1315            'pixel_x': waypoint.pixel_x,
1316            'pixel_y': waypoint.pixel_y,
1317            'angle': waypoint.angle
```

```
        }

        self.add_to_history({
            'type': 'waypoint_edit',
            'waypoint': waypoint,
            'old_state': old_state,
            'new_state': new_state
        })

    def enter_edit_mode(self, waypoint):
        """ウェイポイントの編集モードに入る"""
        self.pgm_display.edit_mode = True
        self.pgm_display.editing_waypoint = waypoint
        self.pgm_display.setCursor(Qt.CursorShape.SizeAllCursor)

    def exit_edit_mode(self):
        """編集モードを終了"""
        self.pgm_display.edit_mode = False
        self.pgm_display.editing_waypoint = None
        self.pgm_display.setCursor(Qt.CursorShape.ArrowCursor)

    def get_combined_pixmap(self):
        """全レイヤーを合成したピクスマップを取得（エクスポート用）"""
        if not self.pgm_layer.pixmap:
            return None

        # 合成用の新しいピクスマップを作成
        result = QPixmap(self.pgm_layer.pixmap.size())
        result.fill(Qt.GlobalColor.white)

        painter = QPainter(result)

        # エクスポートに含めるレイヤーのみを描画
        # origin_layerを除外し、他のレイヤーのみを描画
        export_layers = [
            self.pgm_layer,
            self.drawing_layer,
        ]

        for layer in export_layers:
            if layer.visible and layer.pixmap:
                painter.setOpacity(layer.opacity)
                painter.drawPixmap(0, 0, layer.pixmap)

        painter.end()
        return result

    def import_waypoints_from_yaml(self, yaml_data):
        """YAMLデータからウェイポイントをインポート"""

        if 'waypoints' not in yaml_data:
```

```
            return

        # 既存のウェイポイントをクリア
        self.waypoints.clear()
        Waypoint.reset_counter()

        for wp_data in yaml_data['waypoints']:
            try:
                # ピクセル座標を計算
                if self.origin_point and hasattr(self, 'resolution'):
                    origin_x, origin_y = self.origin_point
                    x_meters = wp_data['x']
                    y_meters = wp_data['y']

                    # メートル座標からピクセル座標に変換
                    pixel_x = int(origin_x + (x_meters / self.resolution))
                    pixel_y = int(origin_y - (y_meters / self.resolution))

                    # 角度の取得（度からラジアンに変換）
                    angle = np.radians(wp_data['angle_degrees'])

                    # 新しいウェイポイントを作成
                    waypoint = Waypoint(pixel_x, pixel_y, angle)
                    waypoint.update_metric_coordinates(origin_x, origin_y, self.resolution)

                    # 追加の属性をインポート
                    for key, value in wp_data.items():
                        if key not in ['number', 'x', 'y', 'angle_degrees', 'angle_radians']:
                            waypoint.set_attribute(key, value)

                    self.waypoints.append(waypoint)
                    self.waypoint_added.emit(waypoint)
            except KeyError as e:
                print(f"Error importing waypoint: Missing key {e}")
                continue
            except Exception as e:
                print(f"Error importing waypoint: {e}")
                continue

        self.update_display()

    def mouseMoveEvent(self, event):
        """マウス移動時のイベント処理"""
        # 既存のmouseMoveEventの処理を維持
        super().mouseMoveEvent(event)

        # ウェイポイントのホバー判定とツールチップ表示
        if self.pgm_layer.pixmap:
            pixmap_geometry = self.pgm_display.geometry()
            scale_x = pixmap_geometry.width() / self.pgm_layer.pixmap.width()
            mouse_pos = event.pos()
```

```
1419
1420            # スケールを考慮した座標に変換
1421            scaled_pos = QPoint(
1422                int(mouse_pos.x() / scale_x),
1423                int(mouse_pos.y() / scale_x)
1424            )
1425
1426            for waypoint in self.waypoints:
1427                if hasattr(waypoint, 'hover_rect') and waypoint.hover_rect.contains(scaled_pos):
1428                    # 属性情報のツールチップを作成
1429                    if waypoint.attributes:
1430                        tooltip = "Attributes:\n"
1431                        for key, value in waypoint.attributes.items():
1432                            tooltip += f"{key}: {value}\n"
1433                        QToolTip.showText(event.globalPos(), tooltip.strip())
1434                        return
1435
1436            QToolTip.hideText()
1437
1438    def add_to_history(self, action):
1439        """履歴に操作を追加"""
1440        # 現在位置より後の履歴を削除
1441        self.history = self.history[:self.current_index + 1]
1442
1443        # 履歴に追加
1444        self.history.append(action)
1445
1446        # 最大履歴数を超えた場合、古い履歴を削除
1447        if len(self.history) > self.max_history:
1448            self.history.pop(0)
1449        else:
1450            self.current_index += 1
1451
1452        # 履歴状態を通知
1453        self.history_changed.emit(
1454            self.can_undo(),
1455            self.can_redo()
1456        )
1457
1458    def can_undo(self):
1459        """操作を戻せるかどうか"""
1460        return self.current_index >= 0
1461
1462    def can_redo(self):
1463        """操作を進められるかどうか"""
1464        return self.current_index < len(self.history) - 1
1465
1466    def undo(self):
1467        """操作を戻す"""
1468        if not self.can_undo():
1469            return
```

```
        action = self.history[self.current_index]
        self.current_index -= 1

        if action['type'] == 'waypoint_add':
            self.remove_waypoint(action['waypoint'].number)
        elif action['type'] == 'waypoint_remove':
            # ウェイポイントを復元
            self.waypoints.append(action['waypoint'])
            self.waypoint_added.emit(action['waypoint'])
        elif action['type'] == 'waypoint_edit':
            # 以前の状態に戻す
            waypoint = action['waypoint']
            old_state = action['old_state']
            waypoint.pixel_x = old_state['pixel_x']
            waypoint.pixel_y = old_state['pixel_y']
            waypoint.angle = old_state['angle']
            if self.origin_point:
                origin_x, origin_y = self.origin_point
                waypoint.update_metric_coordinates(origin_x, origin_y, self.resolution)
            self.waypoint_edited.emit(waypoint)
        elif action['type'] == 'draw':
            # 描画レイヤーを以前の状態に戻す
            self.drawing_layer.pixmap = action['old_pixmap']

        self.update_display()
        self.history_changed.emit(self.can_undo(), self.can_redo())

    def redo(self):
        """操作を進める"""
        if not self.can_redo():
            return

        self.current_index += 1
        action = self.history[self.current_index]

        if action['type'] == 'waypoint_add':
            self.waypoints.append(action['waypoint'])
            self.waypoint_added.emit(action['waypoint'])
        elif action['type'] == 'waypoint_remove':
            self.remove_waypoint(action['waypoint'].number)
        elif action['type'] == 'waypoint_edit':
            # 新しい状態に進める
            waypoint = action['waypoint']
            new_state = action['new_state']
            waypoint.pixel_x = new_state['pixel_x']
            waypoint.pixel_y = new_state['pixel_y']
            waypoint.angle = new_state['angle']
            if self.origin_point:
                origin_x, origin_y = self.origin_point
                waypoint.update_metric_coordinates(origin_x, origin_y, self.resolution)
```

```
            self.waypoint_edited.emit(waypoint)
        elif action['type'] == 'draw':
            # 描画レイヤーを新しい状態に進める
            self.drawing_layer.pixmap = action['new_pixmap']

        self.update_display()
        self.history_changed.emit(self.can_undo(), self.can_redo())

class MenuPanel(QWidget):
    """メニューパネル
    ファイル操作とズーム制御のUIを提供"""

    # シグナルの定義
    file_selected = Signal(str)  # ファイル選択時のシグナル
    zoom_value_changed = Signal(int)  # ズーム値変更時のシグナル
    yaml_selected = Signal(str)  # YAMLファイル選択用のシグナルを追加
    undo_requested = Signal()  # 戻るボタン用シグナル
    redo_requested = Signal()  # 進むボタン用シグナル

    def __init__(self, parent=None):
        super().__init__(parent)
        self.setup_ui()

    def setup_ui(self):
        """UIコンポーネントの初期化と配置"""
        layout = QVBoxLayout()
        self.setStyleSheet("background-color: #e8e8e8;")
        self.setFixedHeight(120)

        # メニューバー
        menu_bar = QMenuBar()
        file_menu = QMenu("File", self)
        edit_menu = QMenu("Edit", self)

        # 戻る/進むボタンの追加
        button_layout = QHBoxLayout()

        self.undo_button = QPushButton("↩ Undo")
        self.undo_button.setEnabled(False)  # 初期状態は無効
        self.undo_button.clicked.connect(self.undo_requested.emit)
        self.undo_button.setStyleSheet("""
            QPushButton {
                background-color: #f0f0f0;
                border: 1px solid #ccc;
                border-radius: 3px;
                padding: 5px 10px;
                min-width: 80px;
            }
            QPushButton:disabled {
                background-color: #e0e0e0;
                color: #999;
```

```
        }
    """)

    self.redo_button = QPushButton("Redo ↪")
    self.redo_button.setEnabled(False)  # 初期状態は無効
    self.redo_button.clicked.connect(self.redo_requested.emit)
    self.redo_button.setStyleSheet("""
        QPushButton {
            background-color: #f0f0f0;
            border: 1px solid #ccc;
            border-radius: 3px;
            padding: 5px 10px;
            min-width: 80px;
        }
        QPushButton:disabled {
            background-color: #e0e0e0;
            color: #999;
        }
    """)

    # button_layout.addWidget(self.undo_button)
    # button_layout.addWidget(self.redo_button)
    # button_layout.addStretch()  # 残りのスペースを埋める

    # ファイルメニューのアクションを作成
    open_action = file_menu.addAction("Open PGM")
    save_action = file_menu.addAction("Save PGM")
    file_menu.addSeparator()
    exit_action = file_menu.addAction("Exit")

    # 編集メニューにUndo/Redoアクションを追加
    undo_action = edit_menu.addAction("Undo")
    undo_action.setShortcut("Ctrl+Z")
    undo_action.triggered.connect(self.undo_requested.emit)

    redo_action = edit_menu.addAction("Redo")
    redo_action.setShortcut("Ctrl+Shift+Z")
    redo_action.triggered.connect(self.redo_requested.emit)

    menu_bar.addMenu(file_menu)
    menu_bar.addMenu(edit_menu)

    # ファイル選択部分のレイアウト
    file_layout = QHBoxLayout()
    self.select_button = QPushButton("Select PGM File")
    self.select_button.clicked.connect(self.open_file_dialog)

    # YAMLファイル選択ボタンを追加
    self.yaml_button = QPushButton("Select YAML File")
    self.yaml_button.clicked.connect(self.open_yaml_dialog)

```

```
        self.file_name_label = QLabel("No file selected")  # ファイル名表示用ラベル
        self.file_name_label.setStyleSheet("color: #666; padding: 0 10px;")

        file_layout.addWidget(self.select_button)
        file_layout.addWidget(self.yaml_button)  # YAMLボタンを追加
        file_layout.addWidget(self.file_name_label, stretch=1)  # stretchを1に設定して余白を埋める

        # ズームコントロールをメソッドに分離
        zoom_widget = self.create_zoom_controls()

        layout.addWidget(menu_bar)
        # layout.addLayout(button_layout)  # 戻る/進むボタンを追加
        layout.addLayout(file_layout)  # ファイル選択部分を追加
        layout.addWidget(zoom_widget)

        # グリッドボタンを追加
        self.grid_button = QPushButton("Toggle Grid")
        self.grid_button.setCheckable(True)  # トグルボタンとして設定
        self.grid_button.setStyleSheet("""
            QPushButton {
                padding: 5px 10px;
                background-color: #f0f0f0;
                border: 1px solid #ccc;
                border-radius: 3px;
            }
            QPushButton:checked {
                background-color: #e0e0e0;
                border: 2px solid #999;
            }
        """)
        file_layout.addWidget(self.grid_button)  # file_layoutにグリッドボタンを追加

        file_layout.addWidget(self.undo_button)
        file_layout.addWidget(self.redo_button)

        self.setLayout(layout)

    def create_zoom_controls(self):
        """ズームコントロールの作成を集約"""
        zoom_widget = QWidget()
        zoom_layout = QHBoxLayout()

        # ズームスライダーの設定
        self.zoom_slider = QSlider(Qt.Orientation.Horizontal)
        self.zoom_slider.setRange(1, 100)
        self.zoom_slider.setValue(50)

        # ズーム率表示用ラベル
        self.zoom_label = QLabel("100%")

        self.zoom_label.setMinimumWidth(50)  # ラベルの最小幅を設定
```

```
1673        self.zoom_label.setAlignment(Qt.AlignmentFlag.AlignRight | Qt.AlignmentFlag.AlignVCenter)
1674
1675        # スライダー値変更時の処理を更新
1676        def update_zoom(value):
1677            zoom_percent = int((value / 50.0) * 100)
1678            self.zoom_label.setText(f"{zoom_percent}%")
1679            self.zoom_value_changed.emit(value)
1680
1681        self.zoom_slider.valueChanged.connect(update_zoom)
1682
1683        reset_button = QPushButton("Reset Zoom")
1684        reset_button.clicked.connect(lambda: self.zoom_slider.setValue(50))
1685
1686        # レイアウトにコンポーネントを追加
1687        zoom_layout.addWidget(self.zoom_slider, stretch=1)  # スライダーを伸縮可能に
1688        zoom_layout.addWidget(self.zoom_label)
1689        zoom_layout.addWidget(reset_button)
1690        zoom_widget.setLayout(zoom_layout)
1691        return zoom_widget
1692
1693    def open_file_dialog(self):
1694        file_name, _ = QFileDialog.getOpenFileName(
1695            self,
1696            "Open PGM File",
1697            "",
1698            "PGM Files (*.pgm);;All Files (*)"
1699        )
1700        if file_name:
1701            # ファイルのベース名（パスを除いた部分）を表示
1702            self.file_name_label.setText(file_name.split('/')[-1])
1703            self.file_selected.emit(file_name)
1704
1705    def open_yaml_dialog(self):
1706        """YAMLファイル選択ダイアログを開く"""
1707        file_name, _ = QFileDialog.getOpenFileName(
1708            self,
1709            "Open YAML File",
1710            "",
1711            "YAML Files (*.yaml *.yml);;All Files (*)"
1712        )
1713        if file_name:
1714            self.yaml_selected.emit(file_name)
1715
1716    def update_undo_redo_actions(self, can_undo, can_redo):
1717        """Undo/Redoボタンの状態を更新"""
1718        self.undo_button.setEnabled(can_undo)
1719        self.redo_button.setEnabled(can_redo)
1720
1721 class LayerControl(QWidget):
1722     def __init__(self, layer, parent=None):
```

```
1723        super().__init__(parent)
1724        self.layer = layer
1725        self.setup_ui()
1726
1727    def setup_ui(self):
1728        layout = QHBoxLayout(self)
1729        layout.setContentsMargins(5, 5, 5, 5)
1730        layout.setSpacing(10)  # ウィジェット間のスペースを設定
1731
1732        self.setStyleSheet("""
1733            QWidget {
1734                background-color: #f0f0f0;
1735                border-radius: 3px;
1736                padding: 5px;
1737            }
1738            QCheckBox {
1739                min-width: 120px;  /* チェックボックスの最小幅を設定 */
1740                max-width: 120px;  /* チェックボックスの最大幅を設定 */
1741            }
1742            QSlider {
1743                min-width: 100px;  /* スライダーの最小幅を設定 */
1744            }
1745        """)
1746
1747        # チェックボックスの設定
1748        self.visibility_cb = QCheckBox(self.layer.name)
1749        self.visibility_cb.setChecked(self.layer.visible)
1750        self.visibility_cb.stateChanged.connect(self._on_visibility_changed)
1751
1752        # 不透明度スライダーの設定
1753        self.opacity_slider = QSlider(Qt.Orientation.Horizontal)
1754        self.opacity_slider.setRange(0, 100)
1755        self.opacity_slider.setValue(int(self.layer.opacity * 100))
1756        self.opacity_slider.valueChanged.connect(self._on_opacity_changed)
1757
1758        # レイアウトに追加
1759        layout.addWidget(self.visibility_cb)
1760        layout.addWidget(self.opacity_slider, stretch=1)  # スライダーを伸縮可能に設定
1761
1762    def _on_visibility_changed(self, state):
1763        """表示/非表示の切り替え"""
1764        self.layer.set_visible(state == Qt.CheckState.Checked.value)
1765
1766    def _on_opacity_changed(self, value):
1767        """不透明度の変更"""
1768        self.layer.set_opacity(value / 100.0)
1769
1770class RightPanel(QWidget):
1771    """右側のパネル"""
1772    waypoint_delete_requested = Signal(int)  # 新しいシグナルを追加
1773    all_waypoints_delete_requested = Signal()  # 新しいシグナル
```

```
    waypoint_reorder_requested = Signal(int, int)  # 順序変更シグナルを追加
    generate_path_requested = Signal()  # パス生成用シグナル
    export_requested = Signal(bool, bool)  # (export_pgm, export_waypoints)
    waypoint_import_requested = Signal(str)  # YAMLファイルパスを送信

    def __init__(self):
        super().__init__()
        self.setup_ui()
        self.waypoint_widgets = {} # ウェイポイントウィジェットを保持する辞書を追加

    def setup_ui(self):
        layout = QVBoxLayout()
        layout.setSpacing(15)
        layout.setContentsMargins(10, 10, 10, 10)
        self.setStyleSheet("QWidget { background-color: #f5f5f5; border-radius: 5px; }")

        # レイヤーパネルを追加
        self.layer_widget = self.create_layer_panel()
        layout.addWidget(self.layer_widget)

        # ウェイポイントリストパネルを追加
        self.waypoint_widget = self.create_waypoint_panel()
        layout.addWidget(self.waypoint_widget)

        # Format Editor (旧Panel 2)を追加
        self.format_editor = FormatEditorPanel()
        layout.addWidget(self.format_editor)

        # エクスポートパネルを追加
        self.export_widget = self.create_export_panel()
        layout.addWidget(self.export_widget)

        self.setLayout(layout)

    def create_layer_panel(self):
        """レイヤーパネルを作成"""
        widget = QWidget()
        layout = QVBoxLayout(widget)

        title_label = QLabel("Layers")
        title_label.setStyleSheet("""
            QLabel {
                font-size: 14px;
                font-weight: bold;
                padding: 5px;
                background-color: #e0e0e0;
                border-radius: 3px;
            }
        """)

        # スクロールエリアを追加
```

```
        scroll_area = QScrollArea()
        scroll_area.setWidgetResizable(True)
        scroll_area.setStyleSheet("""
            QScrollArea {
                background-color: white;
                border: 1px solid #ccc;
                border-radius: 3px;
            }
        """)

        # レイヤーリストのコンテナ
        self.layer_list = QWidget()
        self.layer_list.setStyleSheet("""
            QWidget {
                background-color: white;
                padding: 5px;
            }
        """)
        self.layer_list_layout = QVBoxLayout(self.layer_list)
        self.layer_list_layout.setSpacing(5)
        self.layer_list_layout.setAlignment(Qt.AlignmentFlag.AlignTop)

        # スクロールエリアにレイヤーリストを設定
        scroll_area.setWidget(self.layer_list)

        # 高さの設定
        scroll_area.setMinimumHeight(150)
        scroll_area.setMaximumHeight(200)

        layout.addWidget(title_label)
        layout.addWidget(scroll_area)
        layout.setSpacing(5)

        return widget

    def create_waypoint_panel(self):
        """ウェイポイントリストパネルを作成"""
        widget = QWidget()
        layout = QVBoxLayout(widget)
        layout.setSpacing(5)

        # ヘッダー部分のレイアウト
        header_layout = QHBoxLayout()

        # タイトル
        title_label = QLabel("Waypoints")
        title_label.setStyleSheet("""
            QLabel {
                font-size: 14px;
                font-weight: bold;
                padding: 5px;
```

```
            background-color: #e0e0e0;
            border-radius: 3px;
        }
    """)

    # パス生成ボタン（トグルボタンに変更）
    self.generate_path_button = QPushButton("Generate Path")
    self.generate_path_button.setCheckable(True) # トグルボタンに設定
    self.generate_path_button.setStyleSheet("""
        QPushButton {
            background-color: #4CAF50;
            color: white;
            border-radius: 3px;
            padding: 5px 10px;
            font-size: 12px;
        }
        QPushButton:checked {
            background-color: #45a049;
        }
        QPushButton:hover {
            background-color: #45a049;
        }
    """)
    self.generate_path_button.clicked.connect(self.handle_path_toggle)

    # 全削除ボタン
    clear_button = QPushButton("×")
    clear_button.setFixedSize(20, 20)
    clear_button.setToolTip("すべてのウェイポイントを削除")
    clear_button.setStyleSheet("""
        QPushButton {
            background-color: #ff6b6b;
            color: white;
            border-radius: 10px;
            font-weight: bold;
            font-size: 12px;
        }
        QPushButton:hover {
            background-color: #ff5252;
        }
    """)
    clear_button.clicked.connect(self.all_waypoints_delete_requested.emit)

    # パス生成ボタンと全削除ボタンの間にインポートボタンを追加
    import_button = QPushButton("Import Waypoints")
    import_button.setToolTip("Import Waypoints from YAML")
    import_button.setStyleSheet("""
        QPushButton {
            background-color: #F44336;
            color: white;
            border-radius: 3px;
```

```
            padding: 5px 10px;
            font-size: 12px;
            min-width: 60px;
        }
        QPushButton:hover {
            background-color: #D32F2F;
        }
    """)
    import_button.clicked.connect(self.handle_import_waypoints)

    header_layout.addWidget(title_label)
    header_layout.addWidget(import_button)  # インポートボタンを追加
    header_layout.addStretch()
    header_layout.addWidget(self.generate_path_button)
    header_layout.addWidget(clear_button)

    # スクロールエリアの作成と設定を更新
    self.scroll_area = QScrollArea()  # インスタンス変数として保存
    self.scroll_area.setWidgetResizable(True)
    self.scroll_area.setStyleSheet("""
        QScrollArea {
            background-color: white;
            border: 1px solid #ccc;
            border-radius: 3px;
        }
    """)

    # ウェイポイントリストのコンテナウィジェット
    self.waypoint_list = QWidget()
    self.waypoint_list.setStyleSheet("""
        QWidget {
            background-color: white;
            padding: 5px;
        }
    """)

    self.waypoint_list_layout = QVBoxLayout(self.waypoint_list)
    self.waypoint_list_layout.setSpacing(2)
    self.waypoint_list_layout.setAlignment(Qt.AlignmentFlag.AlignTop)

    # スクロールエリアにウェイポイントリストを設定
    self.scroll_area.setWidget(self.waypoint_list)

    # 固定の高さを設定
    self.scroll_area.setMinimumHeight(150)
    self.scroll_area.setMaximumHeight(300)

    layout.addLayout(header_layout)
    layout.addWidget(self.scroll_area)

```

```
        return widget

    def create_export_panel(self):
        """エクスポートパネルを作成"""
        widget = QWidget()
        layout = QVBoxLayout(widget)

        # タイトル
        title_label = QLabel("Export")  # タイトルを元に戻す
        title_label.setStyleSheet("""
            QLabel {
                font-size: 14px;
                font-weight: bold;
                padding: 5px;
                background-color: #e0e0e0;
                border-radius: 3px;
            }
        """)

        # コンテンツエリア
        content = QWidget()
        content.setStyleSheet("""
            QWidget {
                background-color: white;
                border: 1px solid #ccc;
                border-radius: 3px;
                padding: 10px;
            }
        """)
        content_layout = QVBoxLayout(content)

        # チェックボックス
        self.export_pgm_cb = QCheckBox("Export PGM with drawings")
        self.export_waypoints_cb = QCheckBox("Export Waypoints YAML")

        # ボタンのレイアウト
        button_layout = QHBoxLayout()

        # エクスポートボタン
        export_button = QPushButton("Export Selected")
        export_button.setStyleSheet("""
            QPushButton {
                background-color: #4CAF50;
                color: white;
                border-radius: 3px;
                padding: 8px;
                font-size: 12px;
                min-width: 100px;
            }

            QPushButton:hover {
```

```
            background-color: #45a049;
        }
    """)
    export_button.clicked.connect(self.handle_export)

    # レイアウトに追加（インポートボタン関連の行を削除）
    content_layout.addWidget(self.export_pgm_cb)
    content_layout.addWidget(self.export_waypoints_cb)
    button_layout.addWidget(export_button)
    content_layout.addLayout(button_layout)

    layout.addWidget(title_label)
    layout.addWidget(content)

    return widget

def handle_import_waypoints(self):
    """Waypointのインポート処理"""
    file_name, _ = QFileDialog.getOpenFileName(
        self,
        "Import Waypoints YAML",
        "",
        "YAML Files (*.yaml);;All Files (*)"
    )
    if file_name:
        self.waypoint_import_requested.emit(file_name)

def handle_export(self):
    """エクスポートボタンクリック時の処理"""
    export_pgm = self.export_pgm_cb.isChecked()
    export_waypoints = self.export_waypoints_cb.isChecked()
    if export_pgm or export_waypoints:
        self.export_requested.emit(export_pgm, export_waypoints)

# スクロールタイマーの設定用メソッドを追加
def start_auto_scroll(self):
    if not hasattr(self, 'scroll_timer'):
        self.scroll_timer = QTimer()
        self.scroll_timer.timeout.connect(self.auto_scroll)
        self.scroll_timer.start(50)  # 50ミリ秒ごとにスクロール

def stop_auto_scroll(self):
    if hasattr(self, 'scroll_timer'):
        self.scroll_timer.stop()
        delattr(self, 'scroll_timer')
        self.scroll_region = None

def auto_scroll(self):
    """自動スクロールの処理 - スクロール速度を動的に調整"""

    if not hasattr(self, 'scroll_region') or not hasattr(self, 'scroll_area'):
```

```
2077            return
2078
2079        scroll_bar = self.scroll_area.verticalScrollBar()
2080        current = scroll_bar.value()
2081
2082        # スクロール速度を計算
2083        # マウス位置に基づいて速度を調整（0.0 〜 1.0の範囲）
2084        cursor_pos = self.scroll_area.mapFromGlobal(QCursor.pos())
2085        viewport_height = self.scroll_area.height()
2086
2087        # スクロール領域のマージン（この範囲でスクロール速度が変化）
2088        margin = 50
2089
2090        if self.scroll_region == 'up':
2091            # 上端からの距離に基づいて速度を計算
2092            distance = max(0, cursor_pos.y())
2093            speed_factor = 1.0 - (distance / margin)
2094        else:  # 'down'
2095            # 下端からの距離に基づいて速度を計算
2096            distance = max(0, viewport_height - cursor_pos.y())
2097            speed_factor = 1.0 - (distance / margin)
2098
2099        # 速度係数を0.0から1.0の範囲に制限
2100        speed_factor = max(0.0, min(1.0, speed_factor))
2101
2102        # 基本スクロール速度と最大スクロール速度
2103        base_speed = 5
2104        max_speed = 30
2105
2106        # 実際のスクロール速度を計算
2107        scroll_speed = int(base_speed + (max_speed - base_speed) * speed_factor)
2108
2109        if self.scroll_region == 'up':
2110            new_value = max(scroll_bar.minimum(), current - scroll_speed)
2111            scroll_bar.setValue(new_value)
2112        elif self.scroll_region == 'down':
2113            new_value = min(scroll_bar.maximum(), current + scroll_speed)
2114            scroll_bar.setValue(new_value)
2115
2116        # スクロールが最端に達したら停止
2117        if (self.scroll_region == 'up' and scroll_bar.value() == scroll_bar.minimum()) or \
2118           (self.scroll_region == 'down' and scroll_bar.value() == scroll_bar.maximum()):
2119            self.stop_auto_scroll()
2120
2121    def handle_path_toggle(self):
2122        """パスの表示/非表示を切り替え"""
2123        if self.generate_path_button.isChecked():
2124            self.generate_path_requested.emit()  # パスを生成
2125        else:
2126            # パスを非表示にする
2127            self.generate_path_requested.emit()  # パスをクリア
```

```
2128
2129    def add_waypoint_to_list(self, waypoint):
2130        """ウェイポイントリストに新しいウェイポイントを追加"""
2131        # 既存のウィジェットを更新または新規作成
2132        if waypoint.number in self.waypoint_widgets:
2133            self.waypoint_widgets[waypoint.number].update_label(waypoint.display_name)
2134            return
2135
2136        # 新しいウェイポイントアイテムを作成
2137        waypoint_item = WaypointListItem(waypoint)
2138        self.waypoint_widgets[waypoint.number] = waypoint_item
2139        # 削除シグナルを親パネルのシグナルに接続
2140        waypoint_item.delete_clicked.connect(self.waypoint_delete_requested.emit)
2141        self.waypoint_list_layout.addWidget(waypoint_item)
2142
2143    def remove_waypoint_from_list(self, number):
2144        """ウェイポイントをリストから削除"""
2145        if number == -1:  # 全削除の場合
2146            self.clear_waypoint_list()
2147            return
2148
2149        if number in self.waypoint_widgets:
2150            # 古いウィジェットを削除
2151            widget = self.waypoint_widgets.pop(number)
2152            self.waypoint_list_layout.removeWidget(widget)
2153            widget.deleteLater()
2154
2155            # 残りのウィジェットを全て削除（再ナンバリングのため）
2156            for widget in self.waypoint_widgets.values():
2157                self.waypoint_list_layout.removeWidget(widget)
2158                widget.deleteLater()
2159            self.waypoint_widgets.clear()
2160
2161    def clear_waypoint_list(self):
2162        """ウェイポイントリストをクリア"""
2163        while self.waypoint_list_layout.count():
2164            item = self.waypoint_list_layout.takeAt(0)
2165            if widget := item.widget():
2166                widget.deleteLater()
2167        self.waypoint_widgets.clear()
2168
2169    def update_layer_list(self, layers):
2170        """レイヤーリストを更新"""
2171        # 既存のウィジェットをクリア
2172        for i in reversed(range(self.layer_list_layout.count())):
2173            widget = self.layer_list_layout.itemAt(i).widget()
2174            if widget:
2175                widget.setParent(None)
2176                widget.deleteLater()
2177
2178        # 各レイヤーのコントロールを追加
```

```
        for layer in layers:
            layer_control = LayerControl(layer, self)
            self.layer_list_layout.addWidget(layer_control)

    def handle_waypoint_reorder(self, source_number, target_number):
        """ウェイポイントの順序変更を処理"""
        self.waypoint_reorder_requested.emit(source_number, target_number)

class WaypointListItem(QWidget):
    """ウェイポイントリストの各アイテム用ウィジェット"""
    delete_clicked = Signal(int)

    def __init__(self, waypoint):
        super().__init__()
        self.waypoint_number = waypoint.number
        self.waypoint = waypoint

        self.setAcceptDrops(True)

        # レイアウト設定
        layout = QHBoxLayout(self)
        layout.setContentsMargins(4, 4, 4, 4)
        layout.setSpacing(8)

        # カード風のフレーム
        self.frame = QFrame()
        self.frame.setFrameStyle(QFrame.Shape.StyledPanel)
        self.frame.setStyleSheet("""
            QFrame {
                background-color: white;
                border: 1px solid #e0e0e0;
                border-radius: 4px;
            }
            QFrame:focus {
                border: 1px solid #e0e0e0;
                outline: none;
            }
        """)

        # フレーム内のレイアウト
        frame_layout = QHBoxLayout(self.frame)
        frame_layout.setContentsMargins(8, 4, 8, 4)
        frame_layout.setSpacing(12)

        # ドラッグハンドル
        drag_handle = QLabel(":")
        drag_handle.setStyleSheet("""
            QLabel {
                color: #9e9e9e;
                font-size: 16px;
                padding: 0 2px;
```

```
        }
    """)

    # ウェイポイント番号（青いバッジ風）
    number_badge = QLabel(f"{waypoint.number:02d}")
    number_badge.setStyleSheet("""
        QLabel {
            color: white;
            background-color: #f44336;
            border-radius: 3px;
            padding: 2px 6px;
            font-size: 11px;
            font-weight: bold;
            text-align: center;
        }
    """)
    number_badge.setFixedWidth(40)

    # 座標情報（モノスペースフォントで整列）
    self.coord_label = QLabel(f"({waypoint.x:.2f}, {waypoint.y:.2f})") # インスタンス変数として保存
    self.coord_label.setStyleSheet("""
        QLabel {
            color: #424242;
            font-size: 12px;
        }
        QLabel:focus {
            font-weight: normal;
        }
    """)

    # 角度表示（丸いバッジ風）
    degrees = int(waypoint.angle * 180 / np.pi)
    self.angle_label = QLabel(f"{degrees}°") # インスタンス変数として保存
    self.angle_label.setStyleSheet("""
        QLabel {
            color: #666666;
            background-color: #f5f5f5;
            border-radius: 3px;
            padding: 2px 6px;
            font-size: 11px;
            font-weight: bold;
            min-width: 35px;
            text-align: center;
        }
    """)

    # 削除ボタン
    delete_button = QPushButton("×")
    delete_button.setFixedSize(20, 20)

    delete_button.setStyleSheet("""
```

```
            QPushButton {
                background-color: transparent;
                color: #666666;
                border: none;
                border-radius: 10px;
                font-weight: bold;
                font-size: 14px;
            }
            QPushButton:hover {
                background-color: #ff5252;
                color: white;
            }
        """)
        delete_button.clicked.connect(lambda: self.delete_clicked.emit(self.waypoint_number))

        # フレームにウィジェットを追加
        frame_layout.addWidget(drag_handle)
        frame_layout.addWidget(number_badge)
        frame_layout.addWidget(self.coord_label, 1)
        frame_layout.addWidget(self.angle_label)
        frame_layout.addWidget(delete_button)

        # メインレイアウトにフレームを追加
        layout.addWidget(self.frame)

        # ホバー効果とスペーシングのスタイルを修正
        self.setStyleSheet("""
            WaypointListItem {
                background-color: transparent;
                margin: 1px 0;
            }
            WaypointListItem:hover QFrame {
                border: 1px solid #2196F3;
                background-color: #f8f9fa;
            }
            WaypointListItem:focus {
                outline: none;
            }
        """)

        # フォーカスポリシーを設定
        self.setFocusPolicy(Qt.FocusPolicy.NoFocus)
        self.frame.setFocusPolicy(Qt.FocusPolicy.NoFocus)
        self.coord_label.setFocusPolicy(Qt.FocusPolicy.NoFocus)

    def update_label(self, text):
        """ラベルテキストを更新"""
        # waypoint情報を更新
        if hasattr(self, 'waypoint'):
            degrees = int(self.waypoint.angle * 180 / np.pi)
            self.coord_label.setText(f"({self.waypoint.x:.2f}, {self.waypoint.y:.2f})")
```

```
2331            self.angle_label.setText(f"{degrees}°")
2332
2333        def mousePressEvent(self, event):
2334            if not self.isVisible():
2335                return
2336            if event.button() == Qt.MouseButton.LeftButton:
2337                try:
2338                    # ドラッグ開始時にタイマーをリセット
2339                    right_panel = self.get_right_panel()
2340                    if (right_panel):
2341                        right_panel.stop_auto_scroll()
2342
2343                    drag = QDrag(self)
2344                    mime_data = QMimeData()
2345                    mime_data.setText(str(self.waypoint_number))
2346                    drag.setMimeData(mime_data)
2347
2348                    # ドラッグ中のイベントを監視
2349                    drag.exec(Qt.DropAction.MoveAction)
2350                except RuntimeError:
2351                    pass
2352            # マウスイベントの伝播を停止（super呼び出しを削除）
2353
2354        def mouseMoveEvent(self, event):
2355            # ドラッグ中のマウス位置を取得して自動スクロールの判定
2356            right_panel = self.get_right_panel()
2357            if right_panel and hasattr(right_panel, 'scroll_area'):
2358                scroll_area = right_panel.scroll_area
2359                pos_in_scroll = scroll_area.mapFromGlobal(self.mapToGlobal(event.position().toPoint()))
2360
2361                # スクロール領域の上下端から20ピクセルの範囲を自動スクロール領域とする
2362                scroll_margin = 20
2363
2364                if pos_in_scroll.y() < scroll_margin:
2365                    right_panel.scroll_region = 'up'
2366                    right_panel.start_auto_scroll()
2367                elif pos_in_scroll.y() > scroll_area.height() - scroll_margin:
2368                    right_panel.scroll_region = 'down'
2369                    right_panel.start_auto_scroll()
2370                else:
2371                    right_panel.stop_auto_scroll()
2372
2373            super().mouseMoveEvent(event)
2374
2375        def dragMoveEvent(self, event):
2376            """ドラッグ中の自動スクロール制御を改善"""
2377            right_panel = self.get_right_panel()
2378            if right_panel and hasattr(right_panel, 'scroll_area'):
2379                scroll_area = right_panel.scroll_area

2380                pos_in_scroll = scroll_area.mapFromGlobal(QCursor.pos())
```

```
2381
2382            # スクロール領域のマージンを広げる
2383            scroll_margin = 50
2384
2385            if pos_in_scroll.y() < scroll_margin:
2386                right_panel.scroll_region = 'up'
2387                right_panel.start_auto_scroll()
2388            elif pos_in_scroll.y() > scroll_area.height() - scroll_margin:
2389                right_panel.scroll_region = 'down'
2390                right_panel.start_auto_scroll()
2391
2392        event.accept()
2393
2394    def mouseReleaseEvent(self, event):
2395        # ドラッグ終了時に自動スクロールを停止
2396        right_panel = self.get_right_panel()
2397        if right_panel:
2398            right_panel.stop_auto_scroll()
2399        super().mouseReleaseEvent(event)
2400
2401    def get_right_panel(self):
2402        """親のRightPanelウィジェットを取得"""
2403        parent = self.parent()
2404        while parent and not isinstance(parent, RightPanel):
2405            parent = parent.parent()
2406        return parent
2407
2408    def dragEnterEvent(self, event):
2409        if event.mimeData().hasText() and event.source() != self:
2410            event.accept()
2411            # ドラッグ時のスタイル変更を抑制
2412            self.frame.setStyleSheet("""
2413                QFrame {
2414                    background-color: #f8f9fa;
2415                    border: 1px solid #2196F3;
2416                    border-radius: 4px;
2417                }
2418            """)
2419        else:
2420            event.ignore()
2421
2422    def dragLeaveEvent(self, event):
2423        # ドラッグ離脱時のスタイルを元に戻す
2424        self.frame.setStyleSheet("""
2425            QFrame {
2426                background-color: white;
2427                border: 1px solid #e0e0e0;
2428                border-radius: 4px;
2429            }
2430        """)
```

```
2431        super().dragLeaveEvent(event)
2432
2433    def dropEvent(self, event):
2434        source_number = int(event.mimeData().text())
2435        target_number = self.waypoint_number
2436
2437        # 同じ項目へのドロップは無視
2438        if source_number != target_number:
2439            parent = self.parent()
2440            while parent and not isinstance(parent, RightPanel):
2441                parent = parent.parent()
2442            if parent:
2443                # ドロップ位置に基づいて順序を変更
2444                parent.handle_waypoint_reorder(source_number, target_number)
2445
2446        # frameのスタイルを元に戻す
2447        self.frame.setStyleSheet("""
2448            QFrame {
2449                background-color: white;
2450                border: 1px solid #e0e0e0;
2451                border-radius: 4px;
2452            }
2453        """)
2454        event.accept()
2455
2456 class MainWindow(QMainWindow):
2457    """メインウィンドウ
2458    アプリケーションの主要なUIと機能を統合"""
2459
2460    def __init__(self):
2461        super().__init__()
2462        self.setStyleSheet(COMMON_STYLES)
2463        self.setWindowTitle("Map and Waypoint Editor")  # ウィンドウタイトルを日本語に
2464        self.setGeometry(100, 100, 1200, 1000)
2465
2466        # メインウィジェットとレイアウト
2467        main_widget = QWidget()
2468        main_layout = QVBoxLayout()
2469        splitter = QSplitter(Qt.Orientation.Horizontal)
2470
2471        # 左側パネル
2472        left_widget = QWidget()
2473        left_layout = QVBoxLayout()
2474        left_layout.setSpacing(5) # スペースを追加
2475        left_layout.setContentsMargins(5, 5, 5, 5) # マージンを追加
2476
2477        # メニューパネル
2478        self.menu_panel = MenuPanel()
2479
2480        # 画像ビューア
2481        self.image_viewer = ImageViewer()
```

```
2482
2483        # シグナルの接続（シグナルが発生したときにスロットを呼び出す）
2484        self.menu_panel.file_selected.connect(self.load_pgm_file)
2485        self.menu_panel.zoom_value_changed.connect(self.handle_zoom_value_changed)
2486        # ImageViewerからのスケール変更通知を処理
2487        self.image_viewer.scale_changed.connect(self.handle_scale_changed)
2488
2489        # グリッドボタンのシグナルを接続
2490        self.menu_panel.grid_button.clicked.connect(self.image_viewer.toggle_grid)
2491
2492        # YAMLファイル選択時の処理を接続
2493        self.menu_panel.yaml_selected.connect(self.load_yaml_file)
2494
2495        # 戻る/進むボタンのシグナルを接続
2496        self.menu_panel.undo_requested.connect(self.image_viewer.undo)
2497        self.menu_panel.redo_requested.connect(self.image_viewer.redo)
2498
2499        # 履歴状態の変更を監視
2500        self.image_viewer.history_changed.connect(self.update_history_buttons)
2501
2502        # 左側レイアウトの構成
2503        left_layout.addWidget(self.menu_panel)
2504        left_layout.addWidget(self.image_viewer)
2505        left_widget.setLayout(left_layout)
2506
2507        # 右側パネル
2508        self.right_panel = RightPanel()
2509
2510        # スプリッタの設定
2511        splitter.addWidget(left_widget)
2512        splitter.addWidget(self.right_panel)
2513        splitter.setSizes([600, 400])
2514
2515        main_layout.addWidget(splitter)
2516        main_widget.setLayout(main_layout)
2517        self.setCentralWidget(main_widget)
2518
2519        # レイヤー状態変更時の更新処理を接続
2520        self.image_viewer.layer_changed.connect(self.update_layer_panel)
2521
2522        # 初期レイヤーパネルの更新を追加
2523        self.update_layer_panel()  # この行を追加
2524
2525        # ウェイポイント追加時の処理を接続
2526        self.image_viewer.waypoint_added.connect(self.right_panel.add_waypoint_to_list)
2527
2528        # ウェイポイント削除時の処理を接続
2529        self.image_viewer.waypoint_removed.connect(self.right_panel.remove_waypoint_from_list)
2530
2531        # 削除ボタンクリック時の処理を接続（修正版）
2532        self.right_panel.waypoint_delete_requested.connect(self.image_viewer.remove_waypoint)
```

```
2533
2534        # 全ウェイポイント削除時の処理を接続
2535        self.right_panel.all_waypoints_delete_requested.connect(self.image_viewer.remove_all_waypoints)
2536
2537        # ウェイポイントの順序変更時の処理を接続
2538        self.right_panel.waypoint_reorder_requested.connect(
2539            self.image_viewer.reorder_waypoints)
2540
2541        # パス生成時の処理を接続
2542        self.right_panel.generate_path_requested.connect(
2543            self.image_viewer.generate_path)
2544
2545        # ウェイポイント編集時の処理を接続
2546        self.image_viewer.waypoint_edited.connect(self.right_panel.add_waypoint_to_list)
2547
2548        # エクスポート時の処理を接続
2549        self.right_panel.export_requested.connect(self.handle_export)
2550
2551        # インポート時の処理を接続
2552        self.right_panel.waypoint_import_requested.connect(self.import_waypoints_yaml)
2553
2554    def update_layer_panel(self):
2555        """レイヤーパネルの表示を更新"""
2556        if hasattr(self, 'right_panel') and hasattr(self, 'image_viewer'):
2557            self.right_panel.update_layer_list(self.image_viewer.layers)
2558
2559    def load_pgm_file(self, file_path):
2560        """PGMファイルを読み込む
2561        Parameters:
2562            file_path (str): 読み込むPGMファイルのパス
2563        """
2564        try:
2565            with open(file_path, 'rb') as f:
2566                magic = f.readline().decode('ascii').strip()
2567                if (magic != 'P5'):
2568                    raise ValueError('Not a P5 PGM file')
2569
2570                while True:
2571                    line = f.readline().decode('ascii').strip()
2572                    if not line.startswith('#'):
2573                        break
2574                width, height = map(int, line.split())
2575                max_val = int(f.readline().decode('ascii').strip())
2576
2577                data = f.read()
2578                img_array = np.frombuffer(data, dtype=np.uint8)
2579                img_array = img_array.reshape((height, width))
2580
2581                self.image_viewer.load_image(img_array, width, height)
2582                print(f"Successfully loaded image: {width}x{height}, max value: {255}")
2583
```

```
        except Exception as e:
            print(f"Error loading PGM file: {str(e)}")
            import traceback
            traceback.print_exc()

    def handle_zoom_value_changed(self, value):
        """ズームスライダーの値変更を処理
        Parameters:
            value (int): スライダーの現在値（1-100）
        """
        scale_factor = value / 50.0
        self.image_viewer.scale_factor = scale_factor
        self.image_viewer.update_display()

    def handle_scale_changed(self, scale_factor):
        """ImageViewerからのスケール変更通知を処理"""
        # スライダー値を更新（シグナルの発行を防ぐためにblockSignals使用）
        slider_value = int(scale_factor * 50)
        self.menu_panel.zoom_slider.blockSignals(True)
        self.menu_panel.zoom_slider.setValue(slider_value)
        self.menu_panel.zoom_slider.blockSignals(False)
        # ズーム率表示を更新
        zoom_percent = int(scale_factor * 100)
        self.menu_panel.zoom_label.setText(f"{zoom_percent}%")

    def load_yaml_file(self, file_path):
        """YAMLファイルの読み込みとPGMファイルの自動読み込み"""
        try:
            # YAMLファイルを読み込む
            with open(file_path, 'r') as f:
                yaml_data = yaml.safe_load(f)

            # YAMLファイルのディレクトリパスを取得
            yaml_dir = os.path.dirname(file_path)

            # 画像ファイルのパスを取得し、関連するPGMファイルを読み込む
            if 'image' in yaml_data:
                pgm_filename = yaml_data['image']
                # 相対パスの場合はYAMLファイルのディレクトリを基準に絶対パスを構築
                if not os.path.isabs(pgm_filename):
                    pgm_path = os.path.join(yaml_dir, pgm_filename)
                else:
                    pgm_path = pgm_filename

                # PGMファイルが存在する場合は読み込む
                if os.path.exists(pgm_path):
                    # ファイル名ラベルを更新
                    self.menu_panel.file_name_label.setText(os.path.basename(pgm_path))
                    # PGMファイルを読み込む
                    self.load_pgm_file(pgm_path)
                else:
```

```
                print(f"PGM file not found: {pgm_path}")

            # 原点情報などのYAMLデータを読み込む
            self.image_viewer.load_yaml_file(file_path)

        except Exception as e:
            print(f"Error loading YAML file: {str(e)}")
            import traceback
            traceback.print_exc()

    def handle_export(self, export_pgm, export_waypoints):
        """エクスポート処理"""
        if export_pgm:
            # PGM & YAMLのエクスポート
            self.export_pgm_with_drawings()
        if export_waypoints:
            self.export_waypoints_yaml()

    def export_pgm_with_drawings(self):
        """描画込みのPGMファイルをエクスポート"""
        file_name, _ = QFileDialog.getSaveFileName(
            self,
            "Export PGM with drawings",
            "",
            "PGM Files (*.pgm);;All Files (*)"
        )
        if file_name:
            # ImageViewerの現在の表示内容をPGMとして保存
            pixmap = self.image_viewer.get_combined_pixmap()
            if pixmap:
                image = pixmap.toImage()
                # グレースケールに変換して保存
                gray_image = image.convertToFormat(QImage.Format.Format_Grayscale8)
                gray_image.save(file_name, "PGM")

                # 関連するYAMLファイルを作成
                yaml_file_name = os.path.splitext(file_name)[0] + '.yaml'
                pgm_file_name = os.path.basename(file_name)

                # YAMLデータの作成
                yaml_data = {
                    'image': pgm_file_name,
                    'mode': 'trinary',
                    'resolution': self.image_viewer.resolution,
                    'origin': [0, 0, 0], # デフォルト値
                    'negate': 0,
                    'occupied_thresh': 0.65,
                    'free_thresh': 0.25
                }

```

```
2685            # 原点情報が存在する場合は更新
2686            if self.image_viewer.origin_point:
2687                origin_x = -self.image_viewer.origin_point[0] * self.image_viewer.resolution
2688                origin_y = -(self.image_viewer.pgm_layer.pixmap.height() -
2689                             self.image_viewer.origin_point[1]) * self.image_viewer.resolution
2690                yaml_data['origin'] = [origin_x, origin_y, 0]
2691
2692            # YAMLファイルを保存
2693            try:
2694                with open(yaml_file_name, 'w') as f:
2695                    yaml.dump(yaml_data, f, default_flow_style=None)
2696            except Exception as e:
2697                QMessageBox.warning(self, "Error", f"Error saving YAML file: {str(e)}")
2698
2699    def export_waypoints_yaml(self):
2700        """ウェイポイントをYAMLファイルとしてエクスポート"""
2701        file_name, _ = QFileDialog.getSaveFileName(
2702            self,
2703            "Export Waypoints YAML",
2704            "",
2705            "YAML Files (*.yaml);;All Files (*)"
2706        )
2707        if file_name:
2708            waypoints_data = []
2709            current_format = format_manager.get_format()
2710
2711            for wp in self.image_viewer.waypoints:
2712                # 基本属性を取得
2713                waypoint_data = {
2714                    key: self.get_waypoint_value(wp, key, type_info)
2715                    for key, type_info in current_format['format'].items()
2716                    if self.get_waypoint_value(wp, key, type_info) is not None
2717                }
2718
2719                # カスタム属性を追加
2720                for key, value in wp.attributes.items():
2721                    if key in current_format['format']:
2722                        try:
2723                            # フォーマットに従って型変換を試みる
2724                            converted_value = self.convert_value(value, current_format['format'][key])
2725                            waypoint_data[key] = converted_value
2726                        except:
2727                            # 変換に失敗した場合は文字列のまま保存
2728                            waypoint_data[key] = value
2729
2730                waypoints_data.append(waypoint_data)
2731
2732            data = {
2733                'format_version': current_format['version'],
2734                'waypoints': waypoints_data
```

```
2735            }
2736
2737            try:
2738                with open(file_name, 'w') as f:
2739                    yaml.dump(data, f, default_flow_style=False, sort_keys=False)
2740            except Exception as e:
2741                print(f"Error saving waypoints YAML: {str(e)}")
2742
2743        def import_waypoints_yaml(self, file_path):
2744            """Waypointの設定をYAMLファイルからインポート"""
2745            try:
2746                with open(file_path, 'r') as f:
2747                    data = yaml.safe_load(f)
2748
2749                current_format = format_manager.get_format()
2750
2751                if 'format_version' in data:
2752                    if data['format_version'] != current_format['version']:
2753                        response = QMessageBox.question(
2754                            self,
2755                            "Version Mismatch",
2756                            f"File format version ({data['format_version']}) differs from current version ({current_format['version']}).
    Continue?",
2757                            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
2758                        )
2759                        if response == QMessageBox.StandardButton.No:
2760                            return
2761
2762                self.image_viewer.import_waypoints_from_yaml(data)
2763
2764            except Exception as e:
2765                QMessageBox.critical(self, "Error", f"Error importing waypoints: {str(e)}")
2766
2767        def update_history_buttons(self, can_undo, can_redo):
2768            """戻る/進むボタンの状態を更新"""
2769            self.menu_panel.update_undo_redo_actions(can_undo, can_redo)
2770
2771    class FormatManager:
2772        def __init__(self):
2773            self._format = {
2774                'version': '1.0',
2775                'format': {
2776                    'number': 'int',
2777                    'x': 'float',
2778                    'y': 'float',
2779                    'angle_degrees': 'float',
2780                    'angle_radians': 'float'
2781                }
2782            }
2783            self._observers = []
2784
```

```
2785        def get_format(self):
2786            return self._format
2787
2788        def set_format(self, new_format):
2789            self._format = new_format
2790            self._notify_observers()
2791
2792        def add_observer(self, observer):
2793            self._observers.append(observer)
2794
2795        def _notify_observers(self):
2796            for observer in self._observers:
2797                observer(self._format)
2798
2799    # FormatManagerのグローバルインスタンス
2800    format_manager = FormatManager()
2801
2802    class FormatEditorPanel(QFrame):
2803        format_updated = Signal(dict)  # フォーマット更新時のシグナル
2804
2805        def __init__(self):
2806            super().__init__()
2807            self.setAutoFillBackground(True)
2808            # デフォルトのフォーマットを保存
2809            self.default_format = {
2810                'version': '1.0',
2811                'format': {
2812                    'number': 'int',
2813                    'x': 'float',
2814                    'y': 'float',
2815                    'angle_degrees': 'float',
2816                    'angle_radians': 'float'
2817                }
2818            }
2819            self.setup_ui()
2820            format_manager.add_observer(self.on_format_changed)
2821
2822        def setup_ui(self):
2823            # パネル自体のスタイルを設定
2824            self.setStyleSheet("""
2825                FormatEditorPanel {
2826                    background-color: #f5f5f5;
2827                    border-radius: 5px;
2828                }
2829                QLabel {
2830                    font-size: 14px;
2831                    font-weight: bold;
2832                    padding: 5px;
2833                    background-color: #e0e0e0;
2834                    border-radius: 3px;
2835                }
```

```
            QWidget#contentWidget {
                background-color: white;
                border: 1px solid #ccc;
                border-radius: 3px;
                padding: 10px;
            }
            QTextEdit {
                font-family: monospace;
                font-size: 15pt;
                border: 1px solid #ccc;
                border-radius: 3px;
                padding: 5px;
                background-color: white;
                min-height: 150px;
            }
        """)

        layout = QVBoxLayout(self)
        layout.setSpacing(5)
        layout.setContentsMargins(10, 10, 10, 10)  # マージンを追加

        # タイトル
        title_label = QLabel("Format Editor")

        # コンテンツエリア
        content_widget = QWidget()
        content_widget.setObjectName("contentWidget")  # スタイルシートで参照するためのID
        content_widget.setMinimumHeight(200)

        content_layout = QVBoxLayout(content_widget)
        content_layout.setContentsMargins(10, 10, 10, 10)

        # エディタ
        self.editor = QTextEdit()
        self.editor.setVerticalScrollBarPolicy(Qt.ScrollBarPolicy.ScrollBarAlwaysOn)
        self.editor.setHorizontalScrollBarPolicy(Qt.ScrollBarPolicy.ScrollBarAsNeeded)

        # ボタンレイアウト
        button_layout = QHBoxLayout()

        # 更新ボタン
        update_button = QPushButton("Update Format")
        update_button.setStyleSheet("""
            QPushButton {
                background-color: #2196F3;
                color: white;
                padding: 5px 10px;
                border-radius: 3px;
                min-width: 100px;
            }
            QPushButton:hover {
```

```
            background-color: #1976D2;
        }
    """)
    update_button.clicked.connect(self.update_format)

    # リセットボタン
    reset_button = QPushButton("Reset to Default")
    reset_button.setStyleSheet("""
        QPushButton {
            background-color: #757575;
            color: white;
            padding: 5px 10px;
            border-radius: 3px;
            min-width: 100px;
        }
        QPushButton:hover {
            background-color: #616161;
        }
    """)
    reset_button.clicked.connect(self.reset_to_default)

    # エクスポートボタン
    export_button = QPushButton("Export Format")
    export_button.setStyleSheet("""
        QPushButton {
            background-color: #4CAF50;
            color: white;
            padding: 5px 10px;
            border-radius: 3px;
            min-width: 100px;
        }
        QPushButton:hover {
            background-color: #388E3C;
        }
    """)
    export_button.clicked.connect(self.export_format)

    # インポートボタン
    import_button = QPushButton("Import Format")
    import_button.setStyleSheet("""
        QPushButton {
            background-color: #F44336;
            color: white;
            padding: 5px 10px;
            border-radius: 3px;
            min-width: 100px;
        }
        QPushButton:hover {
            background-color: #D32F2F;
        }
    """)
```

```
        import_button.clicked.connect(self.import_format)

        # ボタンをレイアウトに追加
        button_layout.addWidget(update_button)
        button_layout.addWidget(reset_button)
        button_layout.addWidget(export_button)
        button_layout.addWidget(import_button)

        # コンテンツレイアウトに要素を追加
        content_layout.addWidget(self.editor)
        content_layout.addLayout(button_layout)

        # メインレイアウトに要素を追加
        layout.addWidget(title_label)
        layout.addWidget(content_widget)

        # 初期フォーマットを表示
        self.show_current_format()

    def reset_to_default(self):
        """フォーマットをデフォルトに戻す"""
        # デフォルトのフォーマットを設定
        format_manager.set_format(self.default_format)
        self.show_current_format()
        QMessageBox.information(self, "Success", "Format reset to default")

    def show_current_format(self):
        format_text = yaml.dump(format_manager.get_format(), default_flow_style=False)
        self.editor.setText(format_text)

    def update_format(self):
        try:
            # テキストをYAMLとしてパース
            new_format = yaml.safe_load(self.editor.toPlainText())
            # 必要なキーの存在チェック
            if 'version' not in new_format or 'format' not in new_format:
                raise ValueError("Format must contain 'version' and 'format' keys")

            # フォーマットを更新
            format_manager.set_format(new_format)
            self.format_updated.emit(new_format)

            # 成功メッセージを表示
            QMessageBox.information(self, "Success", "Format updated successfully")
        except Exception as e:
            QMessageBox.warning(self, "Error", f"Invalid format: {str(e)}")

    def export_format(self):
        """フォーマットをYAMLファイルとしてエクスポート"""
        file_name, _ = QFileDialog.getSaveFileName(
            self,
```

```
            "Export Format",
            "",
            "YAML Files (*.yaml);;All Files (*)"
        )
        if file_name:
            try:
                with open(file_name, 'w') as f:
                    f.write(self.editor.toPlainText())
                QMessageBox.information(self, "Success", "Format exported successfully")
            except Exception as e:
                QMessageBox.warning(self, "Error", f"Error exporting format: {str(e)}")

    def import_format(self):
        """フォーマットをYAMLファイルからインポート"""
        file_name, _ = QFileDialog.getOpenFileName(
            self,
            "Import Format",
            "",
            "YAML Files (*.yaml);;All Files (*)"
        )
        if file_name:
            try:
                with open(file_name, 'r') as f:
                    new_format = yaml.safe_load(f)
                format_manager.set_format(new_format)
                self.show_current_format()
                QMessageBox.information(self, "Success", "Format imported successfully")
            except Exception as e:
                QMessageBox.warning(self, "Error", f"Error importing format: {str(e)}")

    def on_format_changed(self, new_format):
        self.show_current_format()

class AttributeDialog(QDialog):
    def __init__(self, waypoint, format_data, parent=None):
        super().__init__(parent)
        self.waypoint = waypoint
        self.format_data = format_data
        self.setup_ui()
        self.parent_viewer = None
        # 親ウィンドウを遡って ImageViewer を探す
        current_parent = parent
        while current_parent:
            if isinstance(current_parent, ImageViewer):
                self.parent_viewer = current_parent
                break
            current_parent = current_parent.parent()

    def setup_ui(self):

        self.setWindowTitle("Add Actions")
```

```
3039        layout = QVBoxLayout(self)
3040
3041        # 属性リスト
3042        self.attribute_list = QWidget()
3043        self.attribute_layout = QVBoxLayout(self.attribute_list)
3044
3045        # フォーマットから利用可能な属性を取得
3046        available_attrs = [key for key in self.format_data['format'].keys()
3047                           if key not in ['number', 'x', 'y', 'angle_degrees', 'angle_radians']]
3048
3049        # 既存の属性を表示
3050        for key in available_attrs:
3051            self.add_attribute_row(key, self.waypoint.get_attribute(key, ""))
3052
3053        scroll = QScrollArea()
3054        scroll.setWidget(self.attribute_list)
3055        scroll.setWidgetResizable(True)
3056        layout.addWidget(scroll)
3057
3058        # ボタン
3059        button_layout = QHBoxLayout()
3060        ok_button = QPushButton("OK")
3061        ok_button.clicked.connect(self.accept)
3062        cancel_button = QPushButton("Cancel")
3063        cancel_button.clicked.connect(self.reject)
3064        button_layout.addWidget(ok_button)
3065        button_layout.addWidget(cancel_button)
3066        layout.addLayout(button_layout)
3067
3068    def add_attribute_row(self, key, value):
3069        """属性入力行を追加"""
3070        row = QHBoxLayout()
3071
3072        # キーのラベル
3073        key_label = QLabel(key)
3074        key_label.setMinimumWidth(100)
3075
3076        # 値の入力フィールド
3077        value_edit = QLineEdit(str(value))
3078        value_edit.setProperty('key', key)  # キーを保存
3079
3080        row.addWidget(key_label)
3081        row.addWidget(value_edit)
3082        self.attribute_layout.addLayout(row)
3083
3084    def get_attributes(self):
3085        """ダイアログから属性を取得"""
3086        attributes = {}
3087        for i in range(self.attribute_layout.count()):
3088            layout_item = self.attribute_layout.itemAt(i)
3089            if layout_item and isinstance(layout_item, QHBoxLayout):
```

```
                value_edit = layout_item.itemAt(1).widget()
                if value_edit and value_edit.text():  # 空でない値のみ保存
                    key = value_edit.property('key')
                    attributes[key] = value_edit.text()
        return attributes

    def accept(self):
        """OKボタンが押された時の処理"""
        # 属性を更新
        self.waypoint.attributes = self.get_attributes()

        # ImageViewerの表示を更新
        if self.parent_viewer:
            self.parent_viewer.update_display()

        super().accept()

def main():
    """アプリケーションのメインエントリーポイント"""
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())

if __name__ == '__main__':
    main()

```