ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE

KATEDRA GEOMATIKY

název předmětu

ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS

číslo úlohy	název úlohy				
1	Geometrické vyhledávání bodu				
školní rok	**studijní skup.**	**číslo zadání**	**Zpracovali:**	**datum**	**klasifikace**
2024	C-101	-	Josef Jehlička Antonín Předota	17.3. 2024	

TECHNICKÁ ZPRÁVA

Zadání: Úkolem bylo implementovat Ray Crossing Algorithm nad polygonovou mapou pro geometrické vyhledání incidujícího polygonu obsahujícího zadaný bod q. Nalezený polygon bylo nutno graficky zvýraznit vhodným způsobem, například vyplněním, šrafováním nebo blikáním. Grafické rozhraní je požadováno vytvořit s využitím frameworku QT.

Polygony byly nutno načítat z textového souboru ve zvoleném formátu. Pro datovou reprezentaci jednotlivých polygonů je vhodno použít spaghetový model.

Zpracované bonusové úlohy:

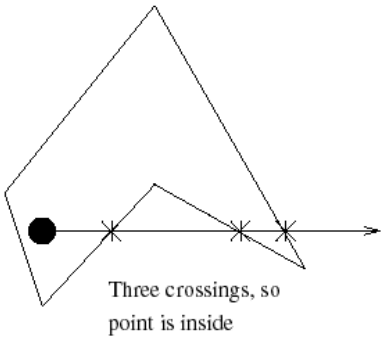
- Analýza polohy bodu (uvnitř/vně) metodou Winding Number Algorithm
- Ošetření singulárního případu ve Winding Number Algorithm: Bod leží na hraně polygonu.
- Ošetření singulárního případu ve Ray Crossing Algorithm: Bod leží na hraně polygonu.

Popis problému: **Point in Polygon Problem (PIP)**
Vstup: jeden či množina polygonů a bod
Výstup: Informace o tom, v jakém polygonu bod leží, či bod leží vně

V oblasti výpočetní geometrie se problém bodu v mnohoúhelníku (PIP – Point-in-Polygon) ptá, zda daný bod v rovině leží uvnitř, vně nebo na hranici mnohoúhelníku. Jedná se o speciální případ problémů umístění bodu a nachází uplatnění v oblastech, které se zabývají zpracováním geometrických dat, jako jsou počítačová grafika, geografické informační systémy (GIS), plánování pohybu a CAD. [1]

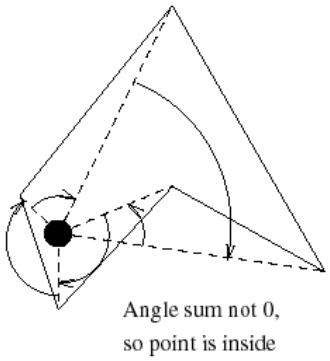
Řešení problému:

Ray crossing (casting) algoritmus – Jednoduchým způsobem, jak zjistit, zda je bod uvnitř nebo vně jednoduchého polygonu, je testovat, kolikrát se paprsek, který začíná v bodě a pokračuje v libovolném pevném směru, protíná hrany polygonu. Pokud dotyčný bod není na hranici polygonu, počet průsečíků je sudý, pokud je bod vně, a lichý, pokud je uvnitř. [2]



Obr.1 : Ray crossing algoritmus [2]

Winding number algoritmus – Algoritmus postupně sčítá úhly mezi vektory (mezi bodem a jednotlivými vrcholy polygonu) v CCW orientaci. Pokud je součet blízký nule, bod je vně; pokud ne, je uvnitř (Obrázek 2). Tato metoda funguje i pro nekonvexní polygony. [2]



Obr.2 : Winding number algoritmus [2]

Výpočet Winding number probíhá následovně [3]:

$$\Omega(q,P) = \frac{1}{2\pi}\sum_{i=1}^{n} \omega(p_i, q, p_{i+1})$$

Kde Ω je suma všech rotací (CCW), které musí průvodič (q, p_i) opsat nad všemi body $p_i \in P$.

Potom přímka p(p_i, p_{i+1}) dělí σ na σ_l, σ_r (Left/Right Halfplane):

$$\sigma l = \{\, q = [x_q, y_q], t > 0 \,\}$$
$$\sigma p = \{\, q = [x_q, y_q], t < 0 \,\}$$

Kde:

$$t = \begin{matrix} x_{i+1} - x_i & y_{i+1} - y_i \\ x_q - x_i & y_q - y_i \end{matrix}$$

Pro úhly ω(p_i, q, p_{i+1}) orientované:

$$\omega(p_i, \mathrm{q}, p_{i+1}) \begin{cases} + \omega(p_i, \mathrm{q}, p_{i+1}), \mathrm{q} \epsilon\, \sigma l \\ - \omega(p_i, \mathrm{q}, p_{i+1}), \mathrm{q} \epsilon\, \sigma p \end{cases}$$

Body potom polygonu (P) náleží dle:

$$\Omega(q,P) = \begin{cases} 1, q \epsilon P \\ 0, q \notin P \end{cases}$$

Popis algoritmů formálním jazykem [3]:

Algoritmus 1: Ray Crossing Algorithm ($P = \{p_1, \ldots, p_n\}, q$)

1. Inicializuj $k = 0$ //Pocet prueciku
2: Opakuj pro $\forall$ body $p_i \in P$:
3: $x_i' = x_i - x_q$.
4: $y_i' = y_i - y_q$.
5: if $(y_i' > 0)\&\&(y_{i-1}' <= 0)||(y_{i-1}' > 0)\&\&(y_i' <= 0)$. //Vhodny segment
6: $x_m' = (x_i' y_{i-1}' - x_{i-1}' y_i')/(y_i' - y_{i-1}')$. //Vhodny prusecik
7: if $(x_m' > 0)$ pak $k = k + 1$.
8: if $(k\%2) \neq 0$ pak $q \in P$
9: else $q \notin P$

Algoritmus 2: Winding Algorithm

1: Inicializuj $\Omega = 0$, tolerance ε.
2: Opakuj pro $\forall$ trojici (p_i, q, p_{i+1}):
3: Urči polohu q vzhledem k $p = (p_i, p_{i+1})$.
4: Urči úhel $\omega_i = \angle p_i, q, p_{i+1}$.
5: If $q \in \overline{\sigma}_l$, pak $\Omega = \Omega + \omega_i$. //Bod v leve polorovine
6: else $\Omega = \Omega - \omega_i$. //Bod v prave polorovine
7: if $||\Omega| - 2\pi| < \varepsilon$, pak $q \in P$ //Test na odchylku od 2π
8: else $q \notin P$

Problematické situace při řešení:

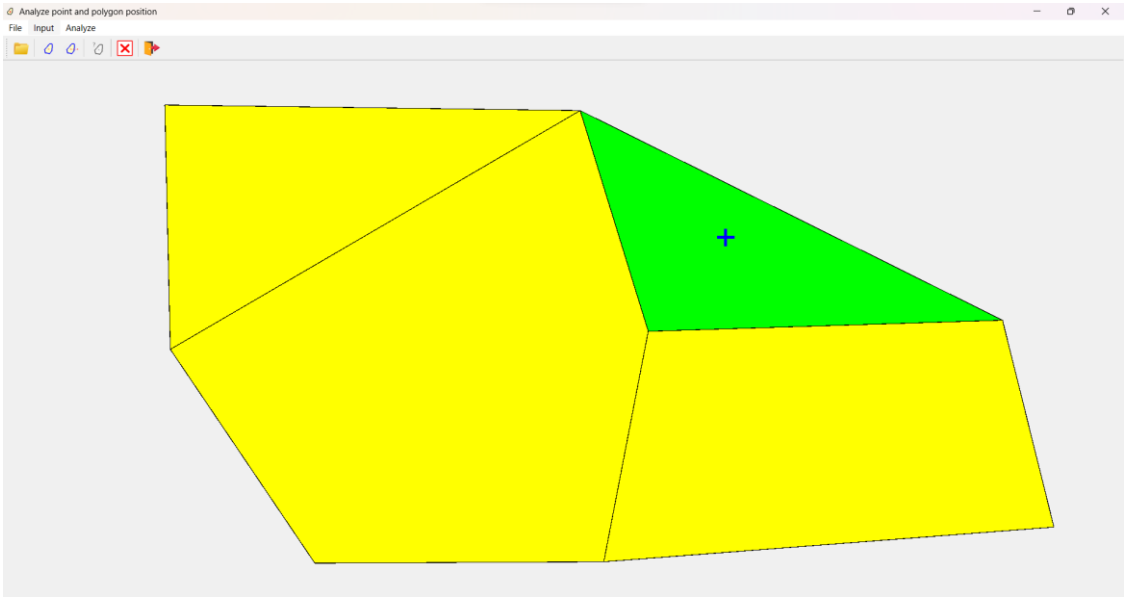
Za nejnáročnější část zpracování je považováno vytvoření metody pro načítání prostorových dat z SHP souborů. Ta probíhá tímto způsobem:

1. Otevře dialogové okno pro uživatele k výběru Shapefile souboru.
2. Načte geometrie z vybraného Shapefile souboru.
3. Vytvoří PyQt polygon z těchto geometrií.
4. Zpracuje body každého polygonu k nalezení minimálních a maximálních souřadnic x a y a uloží všechny souřadnice x a y.
5. Spočítá výšku a šířku ohraničujícího obdélníku, který obsahuje všechny polygony.
6. Spočítá poměry výšky a šířky na základě rozměrů okna aplikace.
7. Spočítá průměr všech souřadnic x a y.
8. Spočítá měřítkový faktor na základě menšího z poměrů výšky a šířky.
9. Spočítá střed ohraničujícího obdélníku a střed okna.
10. Spočítá posunutí ve směrech x a y potřebné k umístění polygonů do středu okna.
11. Změní měřítko a přeloží každý bod každého polygonu na základě vypočteného měřítkového faktoru a posunutí.
12. Kontroluje jedinečné body a ukládá je do nového polygonu.
13. Vrací seznam těchto nových polygonů.

Vstupní data:

Vstupními daty je SHP soubor v souřadnicovém systému WGS84 (EPSG:4326). Soubor obsahuje 4 náhodně vytvořené polygony propojeny v síť. Tato síť se nachází na území Polska. Polygony byly generovány v sw. ArcGIS Pro. Soubor je uložen jako *as.shp* .

Ukázka aplikace:



Aplikace v záložce File obsahuje možnosti Open, která otevře dialogové okno pro výběr SHP souboru, a Exit, která aplikaci ukončí. V záložce input lze přepínat, zda uživatel chce zrovna tlačítkem myši kreslit vrcholy polygonu, či měnit polohu bodu. Ve stejné záložce se nachází tlačítko Clear Data, které vymaže pracovní plochu. V záložce Analyze si může uživatel vybrat, zda právě zadanou situaci v pracovním okně, chce řešit Winding Number algoritmem nebo Ray Crossing Algoritmem.

Dokumentace:

Třída **Ui_MainForm** (MainForm.py)**:**

- Metoda **setupUi**
  Metoda setupUi je zodpovědná za nastavení uživatelského rozhraní hlavního formuláře v aplikaci PyQt6. Je vygenerována v sw. QtDesigner.
  - Nastavuje název objektu, velikost a ikonu okna hlavního formuláře.

    ```
    MainForm.setObjectName("MainForm")
    MainForm.resize(937, 1020)
    MainForm.setWindowIcon(QtGui.QIcon("images/icons/applogo.png"))
    ```

  - Dále načítá ikony a propojuje vytvořená tlačítka s odpovídajícími Metodami

- Metoda **openClick**
  Metoda openClick je zodpovědná za zpracování události, kdy je spuštěna akce "Otevřít". Načítá data ze souboru pomocí třídy IO, nastavuje plátno pro zobrazení načtených polygonů a zakazuje akci "Bod/Polygon".
  - Volá metodu switchYellow objektu Canvas pro změnu barvy polygonu na žlutou.
  - Získá velikost pracovní plochy
  - Načte polygony ze souboru pomocí metody loadData objektu IO.
  - Vycentruje načtená data na pracovní plochu
  - Nastaví kreslení bodu a vypne možnost přepínání Point/polygon

```
def openClick(self):
    # Create object
    io = IO()

    # Run SwitchYellow from draw.py
    self.Canvas.switchYellow()

    size = self.Canvas.size()
    w = size.width()
    h = size.height()
    polygons = io.loadData(w, h)

    # If polygons is not None, then draw polygons
    if polygons != None:
        self.Canvas.setData(polygons)
        self.actionPoint_Polygon.setChecked(True)
        self.actionPoint_Polygon.setEnabled(False)
        self.Canvas.switch2Point()
```

- Metoda **windingNumberClick**
  Metoda windingNumberClick je zodpovědná za analýzu pozice bodu vzhledem k sadě polygonů. Používá algoritmus windingNumber k určení pozice bodu (vnitřní, vnější nebo na hranici) pro každý polygon v sadě.
  - Získá bod a polygony z uživatelského prostředí

    ```
    # Get point and polygon
      q = self.Canvas.getPoint()
      polygons = self.Canvas.getPolygon()
    ```

  - Vytvoří instanci třídy Algorithms

    ```
    a = Algorithms()
    ```

- Definuje slovník position_actions, který mapuje hodnoty pozice vrácené algoritmem

```
position_actions = {
            1:  self.Canvas.greenPolygon,
            2: self.Canvas.greenPolygon,
            3: self.Canvas.greenPolygon
        }
```

- Iteruje přes všechny polygony. Pokud se bod nachází aspoň v jednom polygonu, nabarví ho na zelenou barvu (*GreenPolygon)*. Pokud se bod nenachází v žádném polygonu, všechny s přebarví na červenou barvu (*paintRED)*

```
# Run analysis
        for pol in polygons:
            position = a.windingNumber(q, pol)
            if position in position_actions:
                self.Canvas.switchYellow()
                position_actions[position](pol)
                positions += 1

        # Show results
        if positions == 0:
            self.Canvas.paintRED(polygons)
        else:
            self.Canvas.switchYellow()
```

- Metoda **RayCrossingClick**
  Metoda RayCrossingClick je zodpovědná za analýzu pozice bodu vzhledem k sadě polygonů. Používá algoritmus Ray Crossing (*getPointPolPosition*) k určení pozice bodu (vnitřní, vnější nebo na hranici) pro každý polygon v sadě.
  - Získá bod a polygony z uživatelského prostředí

```
# Get point and polygon
q = self.Canvas.getPoint()
polygons = self.Canvas.getPolygon()
```

  - Vytvoří instanci třídy Algorithms

```
a = Algorithms()
```

  - Použije funkci getPointPolPosition k určení pozice bodu vzhledem k aktuálnímu polygonu. Pokud je pozice 1, 2 nebo 3 (vnitřek polygonu), zvýrazní polygon na plátně zeleně. Nastaví příznak *position_found* na True, pokud je bod uvnitř jakéhokoli polygonu. Pokud je příznak *position_found* False (bod není uvnitř žádného polygonu), vykreslí všechny polygony červeně.

```
    # Show results
        position_found = False
        # Run analysis
        for pol in polygons:
            position = a.getPointPolPosition(q, pol)
            if position in [1, 2, 3]:
                self.Canvas.switchYellow()
                self.Canvas.greenPolygon(pol)
                position_found = True

        if not position_found:
            self.Canvas.paintRED(polygons)
```

- Metoda **pointPolygonClick**
  Metoda je volána, když je spuštěna akce "Bod/Polygon".
  Volá funkci switchDraw objektu Canvas, která přepíná mezi kreslením bodu a kreslením polygonu.

```
# Switch between point and polygon
self.Canvas.switchDraw()
```

- Metoda **clearAllClick**
  Metoda je zodpovědná za vymazání všech dat v objektu Canvas a obnovení stavu prvků uživatelského rozhraní.
```
self.Canvas.clearData()
self.Canvas.switch2Pols()
self.actionPoint_Polygon.setEnabled(True)
self.actionPoint_Polygon.setChecked(False)
```

- Metoda **retranslateUi**
  Metoda slouží k nastavení textu a nápověd různých prvků uživatelského rozhraní v aplikaci. Je vygenerována v sw. QtDesigner.

Třída **IO** (inpout.py)

- Metoda **loadGeometries**
  Metoda se používá k načítání geometrií z Shapefile. Otevírá Shapefile, projde každý záznam v souboru a převede geometrii každého záznamu na objekt Shapely. Objekty Shapely jsou pak přidány do seznamu a vráceny.
```
# Method to load geometries from a Shapefile
        geometries = []
        # Opening the Shapefile
        with open(fileName) as shapefile:
            # Iterating through each record in the Shapefile
            for record in shapefile:
                geom = shape(record['geometry'])
                geometries.append(geom)
        # Returning list of geometries
        return geometries
```

- Metoda **createPolygon**
  Metoda přijímá seznam Shapely geometrií jako vstup a převede je na polygony v PyQt.
  - Inicializujte prázdný seznam nazvaný polygons.
    `polygons = []`
  - Projde každou geometrii v seznamu geometrií.
    `for pol in geometries:`
  - Vytvořte prázdný objekt QPolygonF nazvaný qpolygon.
    `qpolygon = QPolygonF()`
  - Projde každý bod v externích souřadnicích geometrie.
    `for point in pol.exterior.coords:`
  - Vytvoří objekt QPointF s x a y souřadnicemi bodu, kde je y souřadnice vynásobena -1 a přidá objekt QPointF do qpolygon.
    `qpolygon.append(QPointF(point[0], point[1] * (-1)))`
  - Přidá qpolygon do seznamu polygons
    `polygons.append(qpolygon)`
  - Vrátí seznam polygons.
    `return polygons`

- Metoda **scaleAndTranslatePolygons**
  Mění měřítko a posouvá seznam polygonů pomocí poskytnutého měřítka a hodnot posunu.
```
Data = [QPolygonF([QPointF(point.x() * s - shift_x, point.y() * s - shift_y) for point in pol]) for pol in polygons]

return Data
```

- Metoda **processPolygonPoints**
  Slouží k zpracování bodů polygonu a k výpočtu extrémních hodnot (minimální a maximální x a y souřadnice) polygonu.
  - Inicializuje extrémní hodnoty (x_min, y_min, x_max, y_max) na kladné a záporné nekonečno.
  - Vytvoří prázdné seznamy (x_crds, y_crds) pro uložení x a y souřadnic polygonu.
  - Projde každý bod v polygonu.
  - Získá x a y souřadnice aktuálního bodu.
  - Přidá x a y souřadnice do příslušných seznamů.
  - Pokud je to nutné, aktualizuje extrémní hodnoty porovnáním s aktuálními x a y souřadnicemi.
  - Vrátí extrémní hodnoty (x_min, y_min, x_max, y_max) a seznamy souřadnic (x_crds, y_crds).

```
x_min = float('inf')
    y_min = float('inf')
    x_max = float('-inf')
    y_max = float('-inf')

    x_crds = []
    y_crds = []

    # Iterating through each point in the polygon
    for point in pol:
        x = point.x()
        y = point.y()

        x_crds.append(x)
        y_crds.append(y)
        # Updating extreme coordinate if necessary
        x_min = min(x_min, x)
        y_min = min(y_min, y)
        x_max = max(x_max, x)
        y_max = max(y_max, y)

    return x_min, y_min, x_max, y_max, x_crds, y_crds
```

- Metoda **loadData**
  Metoda se používá k načtení Shapefile souboru, extrakci geometrií z něj, vytvoření polygonů z geometrií a provedení různých výpočtů a transformací na polygonů.
  - Metoda otevírá dialogové okno pro výběr Shapefile.
  - Načte geometrie z vybraného Shapefile pomocíf Metoda loadGeometries.
  - Vytváří polygony z načtených geometrií pomocí Metoda createPolygons.
  - Prochází každý polygon a extrahuje minimální a maximální x a y souřadnice, stejně jako všechny x a y souřadnice bodů polygonu pomocí Metoda processPolygonPoints.
  - Vypočítává výšku a šířku ohraničujícího obdélníku všech polygonů.
  - Vypočítává výškové a šířkové poměry mezi velikostí okna a velikostí ohraničujícího obdélníku.
  - Vypočítává průměrné x a y souřadnice všech bodů.
  - Vypočítává měřítkový faktor na základě minima výškového a šířkového poměru.
  - Vypočítává středové souřadnice zmenšených a posunutých polygonů.
  - Vypočítává středové souřadnice okna.
  - Vypočítává posunutí ve směrech x a y.
  - Vytváří nový seznam polygonů se zmenšenými a posunutými souřadnicemi.
  - Vrací seznam polygonů.

```
                        if self.dia.exec():

            # Initializing extreme values
            x_pol_min = float('inf')
            y_pol_min = float('inf')
            x_pols_max = float('-inf')
            y_pols_max = float('-inf')

            # Loading geometries from selected Shapefile
            geometries = self.loadGeometries(self.dia.selectedFiles()[0])
            # Creating polygons from loaded geometries
            polygons = self.createPolygons(geometries)

            all_x = []
            all_y = []

            # Iterating through each polygon
            for pol in polygons:
                min_x, min_y, max_x, max_y, points_x, points_y =
self.processPolygonPoints(pol)

                # Appending coordinates to the list
                all_x += points_x
                all_y += points_y

                # Updating extreme coordinates of polygons if necessary
                x_pol_min = min(x_pol_min, min_x)
                y_pol_min = min(y_pol_min, min_y)
                x_pols_max = max(x_pols_max, max_x)
                y_pols_max = max(y_pols_max, max_y)

            # Calculating height and weight of bounding box
            H = y_pols_max - y_pol_min
            W = x_pols_max - x_pol_min

            # Calculating height ratio and width ratio
            ratio_h = h / H
            ratio_w = w / W

            mean_x = mean(all_x)
            mean_y = mean(all_y)

            # Calculating scaling factor
            scale = min(ratio_w, ratio_h) * 0.9

            center_X = mean_x * scale
            center_Y = mean_y * scale

            # Calculating center of the window
            center_x = w / 2
            center_y = h / 2

            # Calculating shift in X-direction and Y-direction
            shift_x = center_X - center_x
            shift_y = center_Y - center_y

            Data = []
            # Iterating through each polygon
            for pol in polygons:
                polygon = QPolygonF()
                points = {}
                # Iterating through each point in the polygon
                for point in pol:
```

```
                # Scaling and shifting coordinates
                x = point.x() * scale - shift_x
                y = point.y() * scale - shift_y

                # Checking if point is unique
                if x not in points:
                    points[x] = y

                    # Storing Y-coordinate corresponding to X-coordinate
                    point2 = QPointF(x, y)
                    polygon.append(point2)

            Data.append(polygon)

        return Data
```

Třída **Draw** (draw.py)

Třída Draw je podtřídou QWidget v knihovně PyQt6. Poskytuje funkcionalitu pro kreslení polygonů a bodů na widgetu.

- Atribut **self.polygons –** prázdný list pro načtení skupiny polygonů
- Atribut **self.pol** – Definuje objektu polygonu Qt.
- Atribut **self.polygons.append(self.pol) –** Přidává vytvořený polygon do seznamu polygonů
- Atribut **self.q = QPointF(-100, -100)** – Definuje Qt bod mimo pracovní plochu
- Atribut **self.add_vertex –** Definuje příznak, zda uživatel zadává polohu bodu či vrchol polygonu (defaultně True)
- Atribut **self.greenPol** – Definuje výsledný polygon obsahující bod
- Atribut **self.paint_red –** Definuje příznak pro vykreslování polygonů neobsahujících bod (defaultně false). Pokud bod neleží uvnitř žádného polygonu změní se jeho hodnota na True

- Metoda **mousePressEvent**
  Je spuštěna, pokud je stisknuto tlačítko myši nad pracovní plochou.
  - Získá souřadnice kursoru
  - Pokud je příznak add_vertex True, vytvoří nový objekt QPointF s souřadnicemi kurzoru a přidá ho do polygonu pol.
  - Pokud je příznak add_vertex False, nastaví souřadnice bodu q na souřadnice kurzoru.

  Překreslí pracovní plochu.
  self.greenPol = QPolygonF()

```
    # Get cursor position
        x = e.position().x()
        y = e.position().y()

        # Draw polygon
        if self.add_vertex:
            # Create new point
            p = QPointF(x, y)

            # Add point to polygon
            self.pol.append(p)
            self.polygons[0] = self.pol
        else:
            self.q.setX(x)
            self.q.setY(y)

        # Repaint screen
        self.repaint()
```

- Metoda **paintEvent**
  Metoda je zodpovědná za kreslení grafických prvků na widgetu. Používá třídu QPainter k nastavení atributů jako je barva pera a barva výplně, a poté kreslí polygony či bod na základě aktuálního stavu objektu třídy Draw.
  - Vytvoří nový objekt QPainter a začněte kreslit na widgetu.
  - Nastaví barvu pera na černou a barvu štětce na červenou nebo žlutou podle hodnoty příznaku paint_red.
  - Vykreslí všechny polygony uložené v seznamu polygonů pomocí metody drawPolygon objektu QPainter.
  - Nastaví barvu pera na černou a barvu štětce na zelenou
  - Vykreslí zelený polygon uložený v proměnné greenPol pomocí metody drawPolygon.
  - Nastaví barvu pera na modrou a šířku pera na 4.
  - Nakreslí svislou čáru a vodorovnou čáru vycentrovanou na souřadnicích specifikovaných objektem QPointF.
  - Na závěr ukončí vykreslování

```
        # Draw situation
          # Create new object
          qp = QPainter(self)

          # Start drawing
          qp.begin(self)

          # Set atributes
          qp.setPen(Qt.GlobalColor.black)
          if self.paint_red:  # Check the flag here
              qp.setBrush(Qt.GlobalColor.red)
          else:
              qp.setBrush(Qt.GlobalColor.yellow)

          # Draw polygon
          for i in self.polygons:
              qp.drawPolygon(i)

          qp.setPen(Qt.GlobalColor.black)
          qp.setBrush(Qt.GlobalColor.green)

          qp.drawPolygon(self.greenPol)

          # Set attributes
          qp.setPen(QPen(Qt.GlobalColor.blue, 4))

          # Draw "+"
          length = 10
          qp.drawLine(int(self.q.x()), int(self.q.y() -
length), int(self.q.x()),int(self.q.y() + length))
 # Vertical line
          qp.drawLine(int(self.q.x() - length)int(self.q.y()),
int(self.q.x() + length), int(self.q.y()))
  # Horizontal line

          # End drawing
          qp.end()
```

- Metoda **switchDraw**
  Metoda switchDraw se používá k přepínání hodnoty příznaku add_vertex ve třídě Draw.

```
self.add_vertex = not self.add_vertex
```

- Metoda **switch2Point**
  Metoda switch2Point je součástí třídy Draw a slouží k nastavení příznaku s názvem add_vertex na False. Uživatel nyní v pracovní ploše nastavuje polohu bodu.

  ```
  self.add_vertex = False
  ```

- Metoda **switch2Pols**
  Metoda switch2Pols nastavuje příznak, který označuje, že program by měl přepnout na kreslení polygonů místo bodů.

  ```
  self.add_vertex = True
  ```

- Metoda **getPoint**
  Metoda getPoint ve třídě Draw vrací aktuální bod k analýze.

  ```
  return self.q
  ```

- Metoda **getPolygon**
  Metoda getPolygon ve třídě Draw vrací aktuální polygony k analýze.

  ```
  return self.polygons
  ```

- Metoda **clearData**
  Metoda clearData se používá k resetování dat ve třídě Draw. Vymaže souřadnice bodů, data polygonů a nastaví příznak pro vykreslování polygonu na žlutou barvu. Také spouští událost repaint k aktualizaci obrazovky.
  - Metoda volá metodu switchYellow k nastavení příznaku pro malování polygonů na žlutou barvu.
  - Metoda nastaví souřadnice bodu na (-1000, -1000) k jeho vymazání.
  - Metoda vymaže data polygonů voláním metody clear na objektech pol a greenPol.
  - Metoda nastaví seznam polygonů tak, aby obsahoval jediný prázdný objekt QPolygonF.
  - Nakonec metoda volá metodu repaint k aktualizaci obrazovky.

  ```
  # Set the flag to False when clearData is called
          self.switchYellow()
          # Clear point
          self.q.setX(-1000)
          self.q.setY(-1000)

          # Clear polygon
          self.pol.clear()
          self.greenPol.clear()
          self.polygons = [QPolygonF()]

          # Repaint screen
          self.repaint()
  ```

- Metoda **setData**
  Metoda setData ve třídě Draw slouží k nastavení polygonů, které budou vykresleny na obrazovce.

  ```
  self.clearData()
  self.polygons = pols
  self.repaint()
  ```

- Metoda **greenPolygon**
  Metoda greenPolygon ve třídě Draw nastavuje polygon, který má být při volání vykreslen zeleně.

```
self.greenPol = pol
self.repaint()
```

- Metoda **paintRED**
  Metoda paintRED slouží k nastavení příznaku na True a spuštění události repaint, což způsobí, že polygony vykreslí červeně.

```
self.paint_red = True
self.repaint()
```

- Metoda **switchYellow**
  Metoda switchYellow se používá k nastavení příznaku s názvem paint_red na False, když je zavolána. Potom se polygony neobsahující bod vykreslují žlutě.

```
self.paint_red = False
```

Třída **Algorithms** (algorithms.py)

- Metoda **getPointPolPosition**
  Tato metoda určuje polohu bodu vzhledem k polygonu tím, že zkontroluje, zda je bod uvnitř polygonu, na hraně nebo mimo polygon. Využívá Ray Crossing algoritmus.
  - Inicializuje proměnné j, n a vert na hodnotu 0.
  - Projde hrany polygonu.
  - Určí indexy aktuální hrany.
  - Spočítá vzdálenosti mezi bodem a vrcholy hrany.
  - Zkontroluje, zda existuje průsečík mezi hranou a přímkou tvořenou bodem a osou x.
  - Pokud existuje průsečík a x-ová souřadnice průsečíku je větší než 0, zvýší j o 1.
  - Zkontroluje, zda je bod na hraně výpočtem determinantu hrany a přímky tvořené bodem a osou x.
  - Pokud je determinant roven 0, zvětší vert o 1.
  - Na základě hodnot vert a j určí polohu bodu vzhledem k polygonu.
  - Vrátí polohu bodu

```
j = 0
n = len(pol)
vert = 0

for i in range(n):

    if i != (n - 1):
        a = i
        b = i + 1
    else:
        a = i
        b = 0

    x_red = pol[a].x() - q.x()
    y_red = pol[a].y() - q.y()

    xi1_red = pol[b % n].x() - q.x()
    yi1_red = pol[b % n].y() - q.y()
```

```
            if ((yi1_red > 0) and (y_red <= 0)) or ((y_red >
0) and (yi1_red <= 0)):
                x_int = (xi1_red * y_red - x_red * yi1_red)
/ (yi1_red - y_red)
                if x_int > 0:
                    j = j + 1


            det = ((pol[b].x() - pol[a].x()) * (q.y() -
pol[a].y())) - ( (pol[b].y() - pol[a].y()) * (q.x() -
pol[a].x()))
            if det == 0:
                vert = vert + 1


        if vert == 1:
            return 1
        if vert > 1:
            return 2
        if j % 2 == 1:
            return 3
        return 0
```

- Metoda **windingNumber**
  Metoda windingNumber vypočítává winding number bodu vzhledem k polygonu. Algoritmus postupně sčítá úhly mezi vektory a dle toho určuje polohu bodu vůči polygonu.
  - Inicializuje proměnné w, n a vert
  - Projde hrany polygonu.
  - Spočítá vzdálenosti mezi bodem a vrcholy aktuální hrany.
  - Spočítá délky dvou úseček tvořených bodem a vrcholy aktuální hrany.
  - Spočítá kosinovou hodnotu pomocí délek úseček.
  - Zkontroluje speciální případy, kdy jsou délky nulové nebo kosinová hodnota leží mimo rozsah [-1, 1].
  - Spočítá úhel mezi dvěma úsečkami pomocí arkuskosinu kosinové hodnoty.
  - Spočítá determinant dvou úseček a bodu.
  - Aktualizuje winding number na základě znaménka determinantu a úhlu.
  - Aktualizuje počet vrcholů, pokud je determinant roven nule.
  - Určí polohu bodu vzhledem k polygonu na základě počtu vrcholů a winding number.

```
# Initialize variables
        w = 0
        n = len(pol)
        vert = 0

        # Iterate through polygon edges
        for i in range(n):
            a = i
            b = (i + 1) % n

            # Calculate distances between point and vertices
            x_qi_del = pol[a].x() - q.x()
            y_qi_del = pol[a].y() - q.y()

            l_qi = sqrt(x_qi_del * x_qi_del + y_qi_del *
            y_qi_del)

            x_qi1_del = pol[b].x() - q.x()
            y_qi1_del = pol[b].y() - q.y()
```

```
                l_qi1 = sqrt(x_qi1_del * x_qi1_del + y_qi1_del *
                y_qi1_del)

                x_ii1_del = pol[a].x() - pol[b].x()
                y_ii1_del = pol[a].y() - pol[b].y()

                l_ii1 = sqrt(x_ii1_del * x_ii1_del + y_ii1_del *
                y_ii1_del)

                cosine_value = (l_qi * l_qi + l_qi1 * l_qi1 -
                l_ii1 * l_ii1) / (2 * l_qi * l_qi1)

                # Check for special cases
                if l_qi == 0 or l_qi1 == 0 or cosine_value > 1
or cosine_value < -1:
                    vert = 5
                else:
                    w2 = acos(cosine_value)

                # Determine winding number
                det = ((pol[b].x() - pol[a].x()) * (q.y() -
pol[a].y())) - ( (pol[b].y() - pol[a].y()) * (q.x() -
pol[a].x()))

                if det > 0:
                    w = w + w2
                elif det < 0:
                    w = w - w2

                if det == 0:
                    vert = vert + 1

        # Determine point position relative to polygon
        if vert > 1:
            return 1
        if vert == 1:
            return 2
        if abs(abs(w) - 2 * pi) < 0.01:
            return 3
        return 0
```

Závěr:

Výsledná aplikace umožňuje uživateli kreslit polygon, měnit polohu bodu, nahrát polygony z SHP souboru a analyzovat polohu bodu dvěma různými algoritmy.

Za nedostatky jsou považovány dva jevy. Některé polygonové sítě načteny z SHP souboru jsou z malé části zobrazeny mimo pracovní plochu. Také v pár případech po nabarvení polygonů na červeno se po vyčištění pracovní plochy a kresbě nového polygonu zobrazí nový polygon také červeně, i přesto že neproběhla nová analýza polohy bodu.

Seznam literatury:

[1] D. Kularathne and L. Jayarathne, "Point in Polygon Determination Algorithm for 2-D Vector Graphics Applications," *2018 National Information Technology Conference (NITC)*, Colombo, Sri Lanka, 2018, pp. 1-5, doi: 10.1109/NITC.2018.8550057.

[2] Haines, Eric, "Point in Polygon Strategies," Graphics Gems IV, ed. Paul Heckbert, Academic Press, p. 24-46, 1994.

[3] BAYER, Tomáš. Point Location Problem.: Konvexní a nekonvexní oblasti. Ray Algorithm. Winding Number Algorithm. Vysokoškolská prezentace. Katedra aplikované geoinformatiky a kartografie. Přírodovedecká fakulta UK.

V Praze, 22.3.2024

Josef Jehlička
Antonín Předota