Politechnika Warszawska
Wydział Geodezji i Kartografii

Informatyka geodezyjna II

Projekt 2

Wtyczka do QGIS - PyQGIS

Michał Bielecki 319294
Michał Chwałek 319305

Grupa 1
Zajęcia:
poniedziałek 12:15-14:00
Rok akademicki:
2022/23, Semestr 4

Prowądzacy: mgr inż. Andrzej Szeszko

Spis treści

1. Cel ćwiczenia

Celem ćwiczenia jest napisanie wtyczki do programu QGiS umożliwiającej obliczenie różnicy wysokości pomiędzy dwoma punktami wybranymi z warstwy lub obliczenia powierzchni wieloboku utworzonego z conajmniej 3 zaznaczonych punktów.

2. Wykorzystane oprogramowanie oraz moduły

Do utworzenia UserInterface naszej wtyczki został wykorzystany program **QT Designer with QGIS custom widgets**, funkcjonalność wtyczki została napisana z wykorzystaniem języka programowania **python** w wersji **3.9.7**. Do opracowania wtyczki użyliśmy modułów:
— **qgis.PyQt**
— **qgis.core**
— **qgis.utils**
W celu konwersji pliku interfejsu o rozszerzeniu **.ui** do pliku o rozszerzeniu **.py** użyliśmy narzędzia **pyuic5**. W celu utworzenia oraz testowania wtyczki użyliśmy pluginów do QGiS:
— **Plugin Builder**
— **Plugin Reloader**
Do testowania poprawności działania wtyczki użyliśmy stworzonej przez nas warstwy testowej, w której atrybuty geometrii (współrzędne X,Y,Z) znajdowały się w tabeli atrybutów.

3. Przebieg ćwiczenia

Projekt rozpoczeliśmy od utworzenia wtyczki przy pomocy zainstalowanego pluginu do QGiS - **Plugin Builder**. Dzięki temu plugin utworzył nam pliki potrzebne do rozpoczęcia pracy nad User Interface oraz samym działaniem naszej wtyczki. Następnie w programie **QT Designer** rozpoczeliśmy prace nad wyglądem interface'u wtyczki. Dodaliśmy tam elementy, takie jak:
— pushButton
— comboBox
— MapLayerComboBox
— radioButton
— label
Po dodaniu elementóW oraz ich wstępnym rozmieszczeniu, przekonwertowaliśmy plik **.ui** do pliku **.py**. Następnie rozpoczeliśmy pracę nad funkcjonalnością wtyczki. Pierwszym krokiem było utworzenie modułu, który pozwalał na obliczenie różnicy wysokości między dwoma punktami wybranymi z warstwy. W tym celu wykorzystaliśmy punkty z naszej warstwy testowej. W celu poprawnego działania modułu na samym początku dodaliśmy warunek

```
if self.comboBox_obliczenie.currentText() != 'Różnica wysokości':
return
```

Dzięki temu wtyczka oblicza wysokość tylko w przypadku, gdy w comboBox'ie przeznaczonym do określenia typu obliczenia została wybrana opcja ***Różnica wysokości***. Analogiczny warunek powstał przy opcji wybierania obliczania pól powierzchni

W następnym kroku przypisaliśmy wybraną przez użytkownika warstwę do zmiennej **layer** i za pomocą metody **selectedFeatures()** pobieramy zaznaczone przez użytkownika punkty

```
layer = self.mMapLayerComboBox_layers.currentLayer()
selected_points = layer.selectedFeatures()
```

Następnie za pomocą iteracji "wydobywamy"wartości z tabeli atrybutów z konkretnych punktów, w tym celu wykorzystujemy warunek zgodności nazwy.

```
ID = []

for point in selected_points:
idnumber = point['ID']
ID.append(idnumber)
```

Z wydobytych wartości punktów napisaliśmy kod w celu obliczenia różnicy wysokości i powierzchni pola:

• W celu obliczenia powierzchni pola algorytm najpierw sortuje punkty po współrzędnych w kierunku przeciwnym do ruchu wskazówek zegara, a następnie przy pomocy wzorów Gaussa liczy pole powierzchni
• W celu obliczenia różnic wysokości wykorzystaliśmy wzór: $h_2 - h_1$

Zaznaczone przez użytkonika punkty wyświetlają się w czasie rzeczywistym w tabeli. W tym celu stworzyliśmy 3 metody:

```
update_table, update_table_for_surface_area, update_table_for_height_difference
```

Metoda **UpdateTable** definiuje ogólny widok tabeli w zależności od wybranej opcji obiczeń, następnie odwołuje się do kolejnej metody w celu pobrania konkretnych wartości punktów.

```
def update_table(self):
selected_option = self.comboBox_obliczenie.currentText()
self.tableWidget_wybrane.clearContents()
self.tableWidget_wybrane.setRowCount(0)

if selected_option == 'Pole powierzchni':
self.tableWidget_wybrane.setColumnCount(3)
self.tableWidget_wybrane.setHorizontalHeaderLabels(['Nr', 'X', 'Y'])
self.update_table_for_surface_area()
else:
self.tableWidget_wybrane.setColumnCount(2)
self.tableWidget_wybrane.setHorizontalHeaderLabels(['Nr', 'h'])
self.update_table_for_height_difference()
```

Metody **UpdateTableForSurfaceArea** oraz**UpdateTableForHeigtDifferences** iterują po posczególnych elemntach i wybierają pożądane wartości, a następnie przypisują je do odpowiednich kolumn w tabeli:

```
self.tableWidget_wybrane.setRowCount(len(ID))

for row, data in enumerate(zip(ID, Xcoord, Ycoord)):
for col, value in enumerate(data):
item = QtWidgets.QTableWidgetItem(str(value))
self.tableWidget_wybrane.setItem(row, col, item)
```

Metoda **CompareArea** iteruje po tabeli atrybutów w celu wydobycia pola powierzchni z wbudowanych w QGIS'ie narzędzi geometrii. Użytkownik następnie ma możliwość wybór jednostki wyświetlanej wartości:

```
 def compare_area(self):
layer = self.mMapLayerComboBox_layers.currentLayer()
selected_feature = layer.selectedFeatures()
layer.startEditing()
for i in selected_feature:
area = i.geometry().area()
layer.changeAttributeValue(i.id(), layer.fields().indexFromName('Powierzchnia'), area)
layer.commitChanges()
if self.radioButton_m2.isChecked():
self.label_porow.setText(f"Wynik to: {area} metrów kwadratowych")
elif self.radioButton_ar.isChecked():
self.label_porow.setText(f"Wynik to: {area / 100} arów")
elif self.radioButton_ha.isChecked():
self.label_porow.setText(f"Wynik to: {area / 10000} hektarów")
```

Metoda **create_polygon** sortuje wybrane przez użytkownika punkty zgodnie z ruchem wskazówek zegara, następnie pobiera parametry warstwy na której znajdują się punkty, a następnie tworzy nową warstwę na podstawie wcześniej pobranych parametrów, na której znajduje się poligon utworzony z wcześniej posortowanych punktów.

```
    def create_polygon(self):
canvas = iface.mapCanvas()
layer = self.mMapLayerComboBox_layers.currentLayer()
crs = layer.crs()
crs_authid = crs.authid()
selected_features = layer.selectedFeatures()
if len(selected_features) < 3:
iface.messageBar().pushMessage("Błąd",
"Wybierz co najmniej 3 punkty do narysowania poligonu.",
level=Qgis.Warning)
return
points = [feature.geometry().asPoint() for feature in selected_features]
centroid = QgsPoint(sum(point.x() for point in points) / len(points),
sum(point.y() for point in points) / len(points))
points.sort(key=lambda point: -math.atan2(point.y() - centroid.y(), point.x() - centroid.x()))
new_layer = QgsVectorLayer("Polygon?crs=" + crs_authid, "Poligon", "memory")
provider = new_layer.dataProvider()
new_layer.startEditing()
poly_feature = QgsFeature()
polygon = QgsGeometry.fromPolygonXY([points])
poly_feature.setGeometry(polygon)
provider.addFeature(poly_feature)
new_layer.commitChanges()
QgsProject.instance().addMapLayer(new_layer)
canvas.refresh()
```

4. Wnioski

Dzięki pracy przy projekcie mieliśmy możliwość rozwinąć swoje umiejętności w pracy z GitHub'em, QGiS'em, QT Designerem oraz pythonem. Niestety, przez niewystarczającą ilość czasu oraz nieubłagalnie zbliżającą się sesję nie daliśmy rady wykonać wszystkich dodatkowych opcji. Poniżej znajduje się link do zdalnego repozytorium.

github.com/misiek0n/Projekt2