SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

ANĐELA RADOŠ

Z A V R Š N I R A D

QT MULTIPLATFORMSKA APLIKACIJA ZA ISCRTAVANJE GRAFOVA

Split, lipanj 2019.

SVEUČILIŠTE U SPLITU
SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

Predmet: Objektno programiranje

Z A V R Š N I R A D

Kandidat: Anđela Radoš
Naslov rada: Qt multiplatformska aplikacija za iscrtavanje grafova
Mentor: Ljiljana Despalatović, pred.

Split, lipanj 2019.

Sadržaj

Sažetak

U ovome završnom radu opisuje se uporaba pojedinih komponenti razvojnog okvira Qt, preciznije grafičkog sučelja, veza signala i prijemnika te mogućnosti softvera dodatka *Datavisualization* i prikazivanje korisnički unesenih podataka na trodimenzionalnom grafu, što nas dovodi do *shunting yard* algoritma i pretvorbe infiksne u postfiksnu notaciju, koja pojednostavljuje matematičke izračune. Korisnik aplikacije unosi funkciju koju želi vidjeti prikazanu na grafu, nakon što potvrdi unos, djeluje implementirana metoda za popunjavanje grafa, pozivajući pomoćnu pozadinsku metodu „infix-to-postfix“, koja definira sve trojke koordinata za točke unutar određenog raspona. Uz grafički prikaz, u aplikaciju je ugrađena i dodatna funkcionalnost za precizan izračun koordinata pojedine točke te funkcije, mogućnost definiranja raspona za unesenu funkciju te dodavanje dviju površina na graf i njihovo uklanjanje. Postoji i mogućnost spremanja podataka izraza pomoću JSON datoteka i izravan ispis iz njih.

Ključne riječi: JSON, Qt, QtDatavisualization, shunting yard algorithm, signals and slots

Summary

In this thesis the usage of Qt framework features is described, precisely usage of graphic interface, signals – slots relationship along with Datavisualization plug-in functionalities and representing data inserted by user on a three-dimensional graph, which leads us to shunting yard algorithm and infix to postfix conversion, that simplifies mathematics calculations. User inserts the function which they want to see on graph, and when the insert is confirmed, an implemented method for representing data is being executed, calling sub-method „infix-to-postfix“, which defines all coordinate triplets for points that are inside certain range. Along with graphics data representation, the application has extra embedded feature to precisely calculate coordinates of a specific point from that function, as well as ability to define coordinate range for inserted function, adding two surfaces to graph and their removal. There is also a possibility of saving expressions data to JSON files and plotting from them.

Key words: JSON, Qt, QtDatavisualization, shunting yard algorithm, signals and slots

1. Uvod

U ovome poglavlju se opisuje namjena Qt aplikacije za grafički prikaz matematičkih izraza te kako je uopće došlo do te ideje. Zadatak ovog završnog rada bio je implementirati algoritam *shunting yard* za korisnički unesene matematičke izraze i prikazati dobivene podatke na grafu.

Uvijek je velika zainteresiranost za matematičkim samostalnim (engl. *standalone*) aplikacijama i gotovo uvijek je potrebno grafički prikazivati neke podatke. Za dobivanje grafičkih podataka bilo koje vrste, površine, pravca, krivulje i sl, u stvarnom vremenu, pomoću papira i olovke potrebno je izračunati određeni broj točaka koje pripadaju toj površini i ručno iscrtavanje grafa nije precizno. Zato je važno koristiti razvojni okvir koji ima ugrađene mogućnosti za upravljanje trodimenzionalnim prostorom.

Programski jezik u kojemu je napisana aplikacija završnog rada je C++. Korištena verzija razvojnog okvira Qt je 5.12, a verzija uređivača QtCreator 4.8.0. Qt je moderan razvojni okvir te ima jako dobre alate za manipuliranje grafovima i trodimenzionalnim prostorom, koji su brzi, sadrže već označene grafove koji se automatski prilagođavaju veličini iscrtane površine i koriste vektorsko povezivanje točaka, odnosno postoji implementirana metoda za popunjavanje prostora između dvije točke u slijedu (`Q3Dsurface` graf). Uz sve te kompononte bilo je potrebno razumjeti način na koji funkcioniraju i prilagoditi ih aplikaciji, uz implementiran algoritam *shunting yard,* za mehanizam generiranja točaka iz korisničkoga unosa.

Ako bi korisnik unosio iste izraze više puta, umjesto generiranja točki pomoću implementiranih metoda, bilo bi korisno zapamtiti unesene podatke i spremiti ih, kako bi se unosom po drugi put koristili spremljeni podaci. Zato se vrijednosti posljednjeg ispravnog unosa zapisuju u JSON datoteku. Qt ima omogućenu podršku za JSON format i klase za rad s JSON notacijom. Zapis podataka u tu vrstu datoteke je prepoznatljiv i jednostavan, što olakšava kreiranje i čitanje podataka iz JSON datoteke.

Aplikacija funkcionira kao rukovanje događajima (engl. *event-handling*), odnosno

odašiljenjem signala aktiviraju se događaji i pozivaju određene metode. Ti signali su klikovi na botune za dodavanje i uklanjanje površina s grafa, te prije dodavanja unos prolazi provjere valjanosti i korisnika se obavještava u slučaju neispravnog unosa. Svo grafičko sučelje je kreirano u datoteci s nastavkom „ui“.

Napravljena aplikacija može biti korisna jer je upotrebljiva na bilo kakvim matematičkim predavanjima, prezentacijama i analizama te nije potrebno povezivanje na mrežu jer u potpunosti funkcionira neovisno o njoj.

U poglavlju koje slijedi opisuje se funkcioniranje i namjena algoritma *shunting yard* i mogućnosti pretvorbe matematičkih izraza iz jednog oblika u drugi. Zatim slijedi opis razvojnog okvira Qt, uređivača kôda QtCreator i korištenih komponenti, između ostaloga i dodatka QtDataVisualization. Nakon toga se opisuju karakteristike JSON formata podataka, iza čega dolazi implementacija koja objedinjuje sve te komponente.

2. Shunting yard algoritam

Ustanovljena od strane nizozemskog znanstvenika Edsgera Dijkstre, metoda „shunting yard“ koristi se u matematici za pretvaranje izraza napisanih u određenom obliku (infiks, postfiks, prefiks) u drugi oblik, npr. izraza u infiksnoj notaciji u postfiksnu notaciju. Najčešće korišteni oblik je infiksni i primjer takvog oblika su eksplicitne matematičke jednadžbe.

2.1 Formati matematičkih izraza

Kod infiksne notacije, (engl. *in* – unutar), između operanda se nalaze operatori te se računanje krajnjeg rezultata vrši s lijeva nadesno, uzimajući u obzir pravila o prioritetima. Primjer infiksne notacije bi bio izraz oblika 7*6+9, množenje brojeva 7 i 6, te dodavanje broja 9 na dobiveni rezultat.

Uz operatore u infiksnoj notaciji koriste se i zagrade pri davanju prednosti operatorima nižeg prioriteta. One mogu uokvirivati dva operanda i operator koji se nad njima vrši ili višestruke operatore i operande. Uz infiksnu notaciju postoje još i prefiksna i postfiksna.

Kod prefiksne notacije korišteni operator dolazi ispred operanda nad kojima vrši računanje, (engl. *pre* – prije), dok kod postfiksne dolazi nakon operanda, (engl. *post* – poslije). O prefiksnoj notaciji neće biti riječi u nastvaku jer se u ovome radu ne koristi.

Ekvivalentan postfiksni izraz za navedeni primjer bio bi 7 6 * 9 +. Postfiksni format ne koristi zagrade kod zapisa matematičkih izraza, ali ih pri pretvorbi iz drugih oblika može procesirati. Moguća je pretvorba među svim formatima, s tim da je pretvorbu prefiksa u postfiks i obrnutu jednostavnije dobiti posredno, preko infiksa. Izrazi bilo koje kompleksnosti i duljine, ukoliko se mogu napisati u jednome od formata, mogući su u svakome. Tablica 2.1. prikazuje jednostavan matematički izraz u sva tri formata.

Tablica 2.1. Pretvorba iz infiksa u postfiks

INFIX	PREFIKS	POSTFIX
7 + 8	+ 7 8	7 8 +

U svrhu pretvorbe infiksne u postfiksnu notaciju osmišljeno je više algoritama. Proces gdje su svi parovi operator – dva operanda zatvoreni u zasebne zagrade je najjednostaviji način jer omogućuje instantno točan redoslijed operacija, ali postaje sve više nečitljiv što je izraz kompleksniji. Također je bespotrebno ograđivati već prioritizirane operatore zbog postojećih prioriteta i evaluacije s lijeva na desno. Zato se infiksni izraz ne modificira ili prilagođava prije pretvorbe u postfix. Šalje se u izvornom obliku uz korištenje što manje zagrada samo tamo gdje su potrebne. Kod takvog oblika ljudski mozak u vrlo kratkom vremenu dolazi do rezultata, a proces računanja je automatiziran i na računalu. Međutim, ako se u izrazu nalazi nepoznanica, ili više njih, za računalo je potrebno kreirati i dodatne instrukcije za dobivanje rezultata, a za samu pretvorbu u postfiks nije neophodno, jer se nepoznanica tretira kao i broj. Pretvorba u postfiks zasniva se na jednostavnim pravilima:

- Svaki operand ide izravno u novi izraz bez prolaska ikakvih dodatnih provjera.
- Ako je znak prvi operator u izrazu, ide na stog (bez prolaska ikakvih provjera).
- Ako je znak operator, uspoređuje se po prioritetu s posljednjim znakom na stogu, te ako je viši samo se nadovezuje na niz na stogu, a ako je niži ili jednak, izbacuje taj znak sa stoga u novi izraz. Taj proces se ponavlja dok posljednji znak stoga ne bude nižeg prioriteta. Tada se operator samo dodaje na stog.
- Otvorena zagrada infiksnog izraza ide izravno na stog.
- Nailaskom na zatvorenu zagradu sa stoga se u novi izraz izbacuju svi znakovi do otvorene zagrade i ona se poništava.
- Nakon provjere svih znakova infiksnog izraza, znakovi preostali na stogu se prebacuju u novi izraz.
- Kod jednadžbi u pretvorbi sudjeluju samo znakovi nakon znaka jednakosti.

- Na stogu ne mogu biti operatori jednakog prioriteta jedan do drugoga, osim u slučaju operatora najvišeg mogućeg prioriteta, odnosno potencije. Operatori potencije mogu postojati na stogu jedan do drugoga bez obzira na njihov broj.

Iz svega navedenog može se zaključiti da kod pretvorbe operandi ne mijenjaju svoj redoslijed, u novi izraz dolaze istim tokom kao u infiksu.

Tablica 2.2. Pretvorba infiksnog u postfiksni izraz

Znak infiksa „3+5-4*(7+8-3)/2“	Stog	Novi izraz – postfix
3		3
+	+	3
5	+	3 5
-	-	3 5 +
4	-	3 5 + 4
*	- *	3 5 + 4
(	- *(	3 5 + 4
7	- * (	3 5 + 4 7
+	- * (+	3 5 + 4 7
8	- * (+	3 5 + 4 7 8
-	- * (-	3 5 + 4 7 8 +
3	- * (-	3 5 + 4 7 8 + 3
)	- *	3 5 + 4 7 8 + 3 -
/	- /	3 5 + 4 7 8 + 3 - *
2	- /	3 5 + 4 7 8 + 3 - * 2
		3 5 + 4 7 8 + 3 - * 2 / -

Za konkretan infiksni izraz, „3+5-4*(7+8-3)/2“, prikazana je tablica 2.2., gdje svaki red označava jednu radnju. Cijeli postupak bi glasio:

- Prvi znak je broj i ide u novi izraz.

- Idući znak je plus operator i ide na stog.
- Broj 5 ide u novi izraz odmah nakon broja 3.
- Operator minus se uspoređuje po prioritetu s operatorom plus, te s obzirom da im je prioritet jednak, plus prelazi u novi izraz, a minus dolazi na stog.
- Broj 4 ide u novi izraz nakon znaka plus, a operator množenja ide na stog jer ima viši prioritet od znaka na stogu (minus).

- Otvorena zagrada nema po sebi nikakvo značenje, kontrolira joj se samo pozicija na stogu u slučaju nailaska na zatvorenu zagradu u infiksu. Zato otvorena zagrada ide na stog nakon znaka množenja i neposredno poslije nje se na stog može dodati bilo koji operator bez obzira na prioritet.

- Broj 7 ide u novi izraz nakon broja 4, operator plus ide na stog nakon otvorene zagrade, broj 8 u novi izraz, a minus operator na stog s time da izbacuje operator plus u novi izraz. Može se uočiti da, kad ne bi postojala otvorena zagrada na stogu, minus operator nastavio bi izbacivati i ostale operatore sa stoga u novi izraz, ali s obzirom da je ima, proces se obavlja samo do njenog pojavljivanja i onda to staje, a operator se dodaje na stog.

- Broj 3 ide u novi izraz, a pojavljivanjem zatvorene zagrade sa stoga operator minus prelazi u novi izraz, dok se otvorena zagrada samo nestaje sa stoga.

- Operatori dijeljenja i množenja (posljednji na stogu) imaju jednak prioritet i množenje ide sa stoga u novi izraz, a dijeljenje na stog.

- Broj 2 u novi izraz, te preostali znkakovi sa stoga također u novi izraz, po principu skidanja sa stoga, “last in first out“ (posljednji unutra, prvi van).

Postfiksni format ne treba zagrade jer je cijeli algoritam osmišljen na principu redoslijeda i prioriteta, te stog regulira točne pozicije operatora. Tako će nakon bilo kojeg para operanda doći na red odgovarajući operator.

I bez obzira na to nad kojima operandima se vrši koja operacija, u procesu pretvorbe operatori gube svaku vezu s operandima. Operatori se razvrstavaju uz pomoć stoga samo na osnovu prioriteta i redoslijeda. Tek po završetku kreiranja cjelokupnog postfiksnog izraza, ponovno stupaju u vezu s operandima.

Ako je u infiksu neki dio izraza zatvoren u zagrade, na stogu se samo formira skupina operatora unutar zagrada i one u stogu posluže isključivo za obilježavanje početka i kraja te skupine. Također postfiks funkcionira na principu da rezultat dobiven obavljanjem trenutne operacije posluži kao jedan od operanda za iduću operaciju, odnosno uzima se upravo dobiveni rezultat, operator koji slijedi i operand koji slijedi.

3. QT

Qt je razvojni okvir koji omogućuje dodatne funkcionalnosti za C++, korištenjem tzv. "Meta – Object" prevoditelja (eng. *compiler*) nazvanog "MOC". Qt razvojni okvir napisan je u programskom jeziku C++ i moguće ga je koristiti na više platformi/operacijskih sustava, što je testirano pomoću alata Oracle VM za instalaciju virtualnih uređaja. MOC također omogućuje najznačajniju funkcionalnost specifičnu za razvojni okvir Qt, signale i prijemnike, odnosno mehanizam međukomunikacije objekata na način da pokretanje nekog događaja (signal) kao što su klik, nailazak miša preko objekta, povlačenje i otpuštanje objekata, dvoklik i sl. okidaju određenu funkciju (prijemnik), npr. odbrojavanje, kreiranje novoga objekta, promjenu fonta itd.

Qt ima vlastiti izvršni (engl. *make*) alat, zvani qmake, za stvaranje izvršnih datoteka vezanih za platformu na kojoj se pokreće, čitajući datoteke s nastavkom „pro". Ako je u pro datoteci nešto promijenjeno, potrebno je pokrenuti *qmake* za ažuriranje izvršnih datoteka, ako nije, prevoditelj ignorira pro datoteku. Qt razvojni okvir obuhvaća nekoliko biblioteka:

Core framework, Gui framework, SQL framework, Xml framework, Networking framework, OpenGL framework, Multimedia framework, WebKit framework itd.

Postoje komercijalna verzija i slobodni kod Qt-a, ukoliko se koristi komercijalna verzija, nije potrebno dijeliti napisani kôd, a za verziju otvorenog kôda je potrebno.

3.1 QtCreator

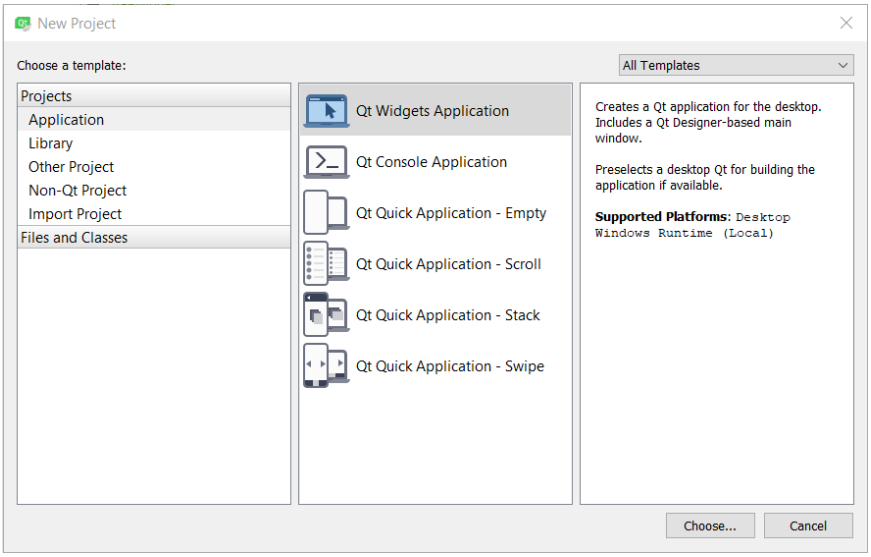
Okruženje za kreiranje Qt programa je jednostavno za korištenje. Sadrži uređivač kôda, dizajner i program za pronalaženje grešaka (engl. *debugger*). Pri pisanju kôda ima mogućnosti formatiranja, nadopunjavanje koda zvano "intellisense", upozoravanje na pogreške i u nekim slučajevima automatsko ispravljanje kôda (. i ->).

QtCreator podržava više programa za pronalaženje grešaka i omogućuje opciju točke zaustavljanja (engl. *breakpoint*). Također ima mnogo mogućnosti u grafičkome sučelju. Za

grafičke aplikacije omogućeno je brzo prebacivanje iz dizajnera u uređivač pomoću trake s lijeve strane ekrana, a za bilo koji izraz sintakse napisana je dokumentacija i pristupa joj se pomoću tipke F1.

Qt ima i mogućnost praćenja curenja memorije (engl. *memory leak*). Instalacijom QtCreatora dobiva se i mnoštvo aplikacija – primjera napisanih u Qt-u, najviše grafičkih aplikacija. Podržava razne formate datoteka i moguće je otvoriti slike, xml, JSON dokumente itd. Nudi i mogućnost interakcije s Git sustavom.

Postoji mnogo priključaka za QtCreator koji se mogu instalirati, a moguće su i detaljne postavke formata, programa za ispravljanje pogreški i uređivača kôda općenito na alatnoj traci. Qt ima podršku za više programskih jezika. Na slici 3.1. prikazan je izbornik pri kreiranju projekata u uređivaču Qt Creator.



Slika 3.1. Forma za odabir vrste projekta u Qt Creatoru

3.2 Klasa QObject

Osim signala i prijemnika, MOC omogućuje još i `QObject`, baznu i hijerarhijski najvišu klasu koju sve ostale klase nasljeđuju. Objekti klase `QObject` uglavnom imaju objekt roditelja, da se pri njihovom kreiranju u konstruktoru navede pokazivač na roditeljski objekt ili mu se u međuvremenu on dodijeli metodom `setParent()`.

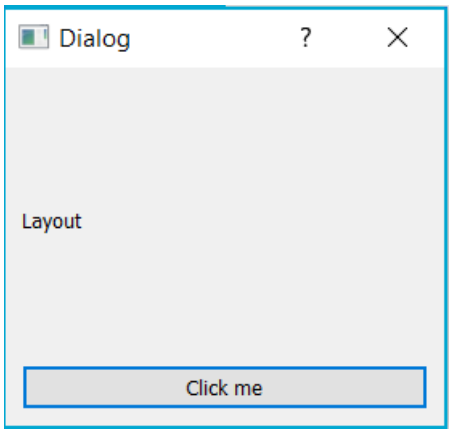
Takav princip najkorisniji je kod kreiranja objekata klase `QWidgets`, odnosno kod rada s grafičkim sučeljem. Npr. glavni prozor, dijalog ili drugi grafički element su roditelj svakoga objekta koji se stvori na njima. Ako na prozoru postoji neka površina (engl. *layout*), na kojemu se još kreira traka za prikaz tijeka (engl. *progress bar*), onda je površina neposredni roditelj za traku, a glavni prozor roditelj za površinu, što znači da neki objekt istovremeno može biti roditelj jednome, a dijete drugome objektu.

Roditelj se objektu najčešće dodjeljuje prema grafičkome prikazu, odnosnog onaj koji djeluje kao posjednik (engl. *holder*) i obuhvaća taj objekt. Tako bi za sliku 3.2. vrijedilo da je dijalog roditelj svim objektima na slici, a uz to je površina roditelj labele koja se na njemu nalazi. (površina nije vidljiva, služi za grupiranje drugih grafičkih elemenata).

Brisanjem objekta roditelja briše se i objekt dijete, odnosno cijela hijerarhija ispod toga objekta, ali modificiranje roditelja na neki drugi način, kao što je promjena veličine i sl, ne utječe na dijete, osim ako to nije definirano.

Također varijable klase `QObject` ne moraju biti djeca objekta te klase, tu vezu definira kreator aplikacije. Ali s obzirom da objekt klase `QObject` živi u dretvi u kojoj se kreira, onda i roditelj ili dijete toga objekta mora biti u istoj dretvi jer se inače neće moći uspostaviti roditeljska veza.

U sljedećem primjeru, botun na čiji klik se kreira okvir s porukom (engl. *MessageBox*), je roditelj toga objekta (slika 3.2.).



Slika 3.2. Objekti roditelji i djeca u grafičkom sučelju

Klasa `QObject` omogućuje i meta-informacije o samome objektu, kao što su ime objekta, klase, informacije o tome nasljeđuje li objekt određenu klasu, pronalazak objekta ili liste objekata djece, signal da je objekt uništen itd.

U zaglavlju bilo koje klase koja nasljeđuje `QObject` mora se nalaziti `Q_OBJECT` makro, kako bi prevoditelj, čitajući zaglavlje, mogao kreirati .cpp datoteku s *meta-object* kôdom za klase koje imaju makro u svojoj deklaraciji.

Klasa `QObject` ne mora sadržavati makro kako bi funkcionirala, ali za određena svojstva, između ostalog signale i prijemnike, mora imati makro za kontroliranje toga mehanizma, da je razumljiviji za prevoditelj.

3.3 Signali i prijemnici

Pod *signal* se smatra bilo koji događaj ili promjena stanja nad nekim objektom, odnosno grafičkim elementom, a *prijemnik* (engl. *slot* – prijemnik, utor) je reakcija na emitiranje signala. Za Qt postoje ugrađeni signali i prijemnici, a također korisnici mogu kreirati prilagođeni.

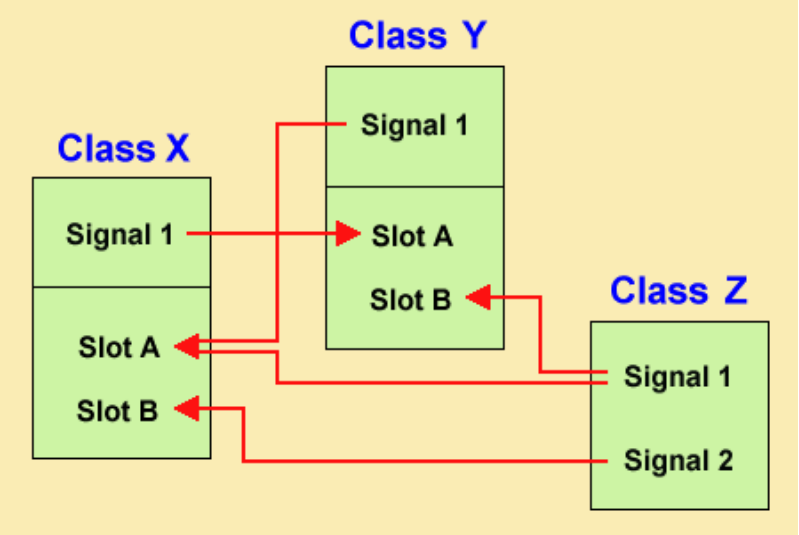
Određeni objekt može sadržavati signal ili prijemnik i kod međusobnog povezivanja se uz signal i prijemnik navode i objekti koji ih sadrže. Povezivanje se obavlja pomoću metode `connect()`, prosljeđujući joj prvi objekt, signal kojega sadrži, drugi objekt i prijemnik kojega on sadrži. Povezivanje je moguće i pozivanjem prijemnika unutar signala. Kako objekti ne moraju biti ovisni jedan o drugome, to vrijedi i za signale i prijemnike.

Ukoliko je neki signal povezan s jednim prijemnikom, može biti povezan i s drugima, a i prijemnik s više signala istovremeno (slika 3.3.). Takav mehanizam se još naziva i „event handling“, jer se reakcije, odnosno prijemnici aktiviraju na neki događaj, a prednost toga principa je taj da se ne mora neprestano provjeravati je li se određeni događaj izvršio, nego se zna uz pomoć odaslanoga signala. Međutim, za povezivanje signala i prijemnika pomoću metode `connect()`, oni moraju biti istovrsni, odnosno primati jednak broj argumenata i to istoga tipa podataka. Npr. signal `click()` može se povezati sa prijemnikom `close()` jer primaju jednak broj argumenata.

Dok postojanje signala uglavnom ukazuje na neke događaje i aktivaciju prijemnika, to ne mora vrijediti i za prijemnike. Oni mogu biti pozvani i bez povezivanja sa signalima kao obične metode ili korišteni kao pomoćne metode itd.

Moguće je i višestruko povezivanje signala međusobno, tako da emitiranje prvoga signala uzrokuje emitiranje drugoga kao reakciju. Za bilo kakve interakcije s grafičkim elementima (engl. *widget*) neophodno je korištenje signala i prijemnika, a pogotovo za interakciju sa složenijim objektima.

Ako u signal uzrokuje promjene samo u nekim slučajevima, potrebno je postaviti uvjete na početku signala. Također je potrebno dodati i ishode u slučaju ispunjavanja i neispunjavanja uvjeta kako bi se signal mogao ponovno odašiljati.



Slika 3.3. Grafički prikaz povezivanja signala i prijemnika

3.4 Qt Data Visualization

Funkcionalnost Data Visualization služi za trodimenzionalni prikaz određenih podataka u koordinatnome sustavu. U zaglavlje ili implementaciju kôda gdje se koristi potrebno je umetnuti "`#include <QtDataVisualization>`" i "`using namespace QtDataVisualization`" (opcionalno, ako se želi izbjeći prefiks `QtDataVisualization::` za svaku komponentu). Može se i preporučljivo je uključiti samo neke klase koje se koriste, a ne sve.

Ova funkcionalnost omogućuje mnogo raznolikih opcija i ima jako korisnih ugrađenih svojstava. Najčešća upotreba je kod prikaza maketa i reljefa zemljopisnih lokacija. Posjeduje predefinirane opcije za bojanje površina i pozadine, odnosno teme, kao i prilagođene mogućnosti da se određeni dijelovi grafa, npr. oni različite visine prikažu u različitim bojama.

Kao temelj koristi se objekt klase `QAbstract3DGraph`. Postoje tri vrste grafova: `Q3DSurface` (za vektorsko crtanje, koordinate su povezane u jednu površinu), `Q3DScatter` (za zaseban prikaz točaka u koordinatnome sustavu, podaci su u obliku disperzije) i `Q3DBars` (iako trodimenzionalan vizualno, češće se koristi u kontekstu

dvodimenzionalnih prikaza, statističkih podataka, rasta i pada s obzirom na vremenski period i sl). U ovome radu korišten je graf vrste `Q3DSurface`, s već ugrađenim izgledom. Za postavljanje podataka na graf koristi se objekt klase `Q3DSurfaceSeries`, ima opcije za označavanje koordinatnih osi, a u konstruktor mu se prosljeđuje objekt klase `QsurfaceDataProxy` koji drži podatke, odnosno figura (engl. *mesh*).

4. JSON

JSON (JavaScript Object Notation) je format podataka koji se uglavnom koristi za kreiranje konfiguracijskih datoteka ili za serijalizaciju objekata pri prijenosu podataka. Koristi tekstualni zapis podataka i može se uređivati u tekstualnim uređivačima. Iako je najsličniji sintaksi programa Javascript, JSON format ne ovisi o programskom jeziku i podržava ga većina biblioteka. Objekti su predstavljeni u ključ-vrijednost (engl. *key-value*) parovima podataka.

Vrste podataka podržane u JSON datotekama su brojevi, stringovi, nizovi, *boolean* vrijednosti i objekti. Objekti su zatvoreni u vitičastim zagradama, a nizovi u uglatim. Parovi su odvojeni dvotočkom i prvi član para (ključ) ide u dvostruke navodnike, a objekti su odvojeni zarezom.

4.1 JSON podrška u QT

Qt ima ugrađenu podršku za raščlanjivanje podataka u JSON formatu. Enkapsulirano je nekoliko klasa za upravljanje JSON podacima. `QJsonArray` služi za nizove vrijednosti ili objekata. U niz se može dodavati i vaditi elemente, na početak ili kraj niza i na bilo kojemu indeksu pomoću uobičajenog C++ iteratora.

Objekt klase `QJsonObject` sadrži listu podataka. Vrijednosti u listi su tipa `QJsonValue`, nepobrojanog tipa podataka (engl. *enum*) koji sadrži sve tipove podataka. U JSON objekt može se umetati podatke, uklanjati, provjeravati veličina i postojanje podataka.

`QJsonDocument` predstavlja dokument za čitanje ili pisanje podataka. Metoda `toJson()` pretvara objekt klase `QJsonDocument` u tekstualni oblik i koristi se kod zapisivanja u datoteku, a za pretvaranje iz tekstualnog oblika natrag u `QJsonDocument` je

metoda `fromJson()`. Metode `read()` i `write()` primaju kao argument niz bajtova, odnosno objekt klase `QByteArray`, pa je bitno kod pisanja u taj objekt spremiti tekstualni oblik, a kod čitanja object tipa `QJsonDocument`. U dokumentu se može obavljati pretraga objekata i nizova, provjeravati ispravnost dokumenta i dohvaćati objekte. Ispis 4.1. prikazuje spremljene podatke u JSON datoteci.

```
{
"advanced": "",
"coord1_max": "3",
"coord1_min": "1",
"coord2_max": "3",
"coord2_min": "1",
"equation_id": "z=x*(y-8)+6",
"points_on_graph": [
      {
      "x": 1,
      "y": 1,
      "z": -1
      },
      {
      "x": 1,
      "y": 2,
      "z": 0
      },
      {
      "x": 2,
      "y": 1,
      "z": -8
      },
      {
      "x": 2,
      "y": 2,
      "z"; -6
      }
      ],
"rate": "1"
}
```


Ispis 4.1. JSON format generiran generiran unosom podataka

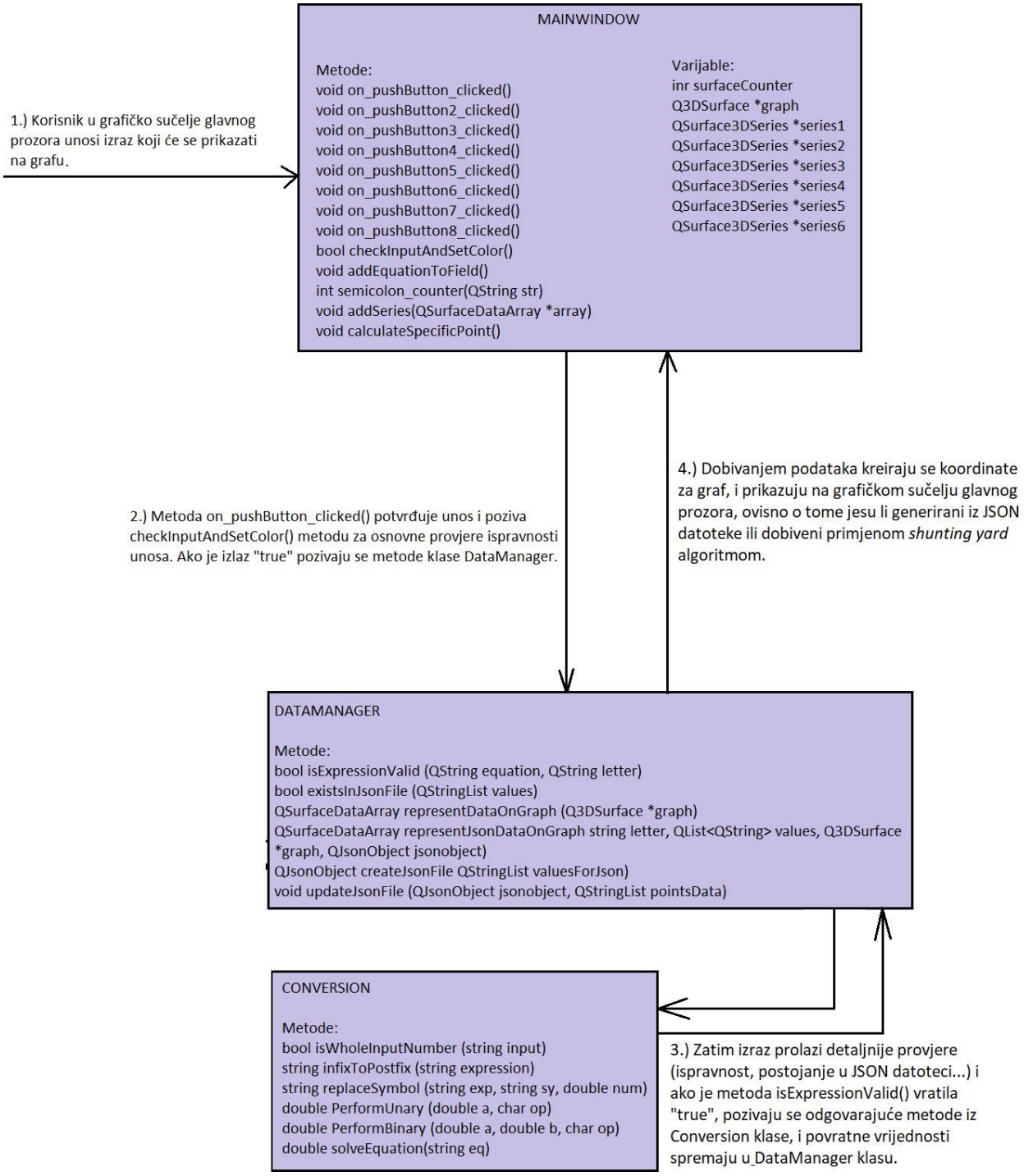
5. Implementacija

Projekt sadrži glavni prozor u kojemu je svo grafičko sučelje i dvije klase:

- `Conversion` u kojoj se nalaze sve procedure vezane za izračune i matematičke operacije i rad s unesenim podacima,

- `DataManager` u kojoj su definirane metode za provjeru ispravnosti unosa i za postavljanje podataka na graf.

U datoteku *qmake* (nastavak .pro) dodani su "`QT+= QtCore`" i "`QT += QtDataVisualization`". Te informacije su potrebne konfiguracije kod izgradnje projekta i izradi izvršne datoteke.
`QtCore` sadrži osnovne Qt klase. Za uključeni `QtCore` i `QtDatavisualization` u datotekama implementacije kôda, potrebno je obaviti tzv. linkanje, odnosno generiranje objekata i stvaranje neke izvršne datoteke od strane prevoditelja u .pro datoteci.
Slika 5.1. prikazuje tok i način funkcioniranja aplikacije.



Slika 5.1. Dijagram toka aplikacije

5.1 Klasa Conversion

Jedina svrha postojanja ove klase je sakupljanje svih matematičkih procesa na jedno mjesto, tako da klasa glavnog prozora bude rasterećena od metoda i da istovrsne stavke budu u istoj klasi. Klasa se sastoji samo od spomenutih metoda i ne posjeduje nikakve varijable članove, kao ni *gettere* i *settere*. Sve metode su čisti C++ bez dodataka i razvojnog okvira Qt. Uključuju se biblioteke:

- `<algorithm>` za korištenje predefiniranih matematičkih funkcija, trigonometrijskih, potencija itd.
- `<stack>` korištenje stoga zbog potrebe za rasporedom gdje se element s vrha vadi prvi, kod dodavanja operatora (engl. *last-in-first-out*).
- `<map>` za mapiranje operatora i njihovih prioriteta i provjere je li neki znak operator, obzirom da je unos tipa `QString`.
- `<string>` za std string umjesto niza char znakova.

Prva metoda je `replaceSymbol()` i neophodna je za cjeloviti izračun. Za pretvorbu u postfix nije neophodna jer se i nepoznanice mogu upotrebljavati kao operandi. Poziva se obavezno dvaput (za 3D prostor najveći mogući broj nepoznanica), ako su u izrazu nepoznanice, da se zamijenu za odgovarajuće brojeve. U kôdu 5.2. prikazana je funkcionalnost metode te parametri i povratna vrijednost.

```
/**
     * @brief Conversion::replaceSymbol checks expression and replaces
    passed
     * symbol (x,y or z) with passed number
     * @param exp expression to check
     * @param sy symbol which will be replaced
     * @param num number which will replace the symbol
     * @return transformed expression string type
*/
string Conversion::replaceSymbol(string exp, char sy, double num)
string newString = "";
for (unsigned i = 0; i < exp.length();
i++){ std::string s(1,exp[i]);
if (s == sy) {
      if (num >= 0) {
      newString += to_string(num);
      }
      else {
      newString += "(" + to_string(num) + ")";
      }
}
else{
      newString += s;
      }
}
return newString;
```

Kôd 5.2. Zaglavlje i implementacija metode `replaceSymbol()`

Iduća metoda je `infixToPostfix()` i kao argument prima string (infiksni matematički izraz). Već opisana na početku, transformira infiks u postfiks pomoću stoga te takav novi (postfiksni) izraz joj je povratna vrijednost tipa `std string`. U slučaju nemogućnosti dolaska do rezultata vraća prazan `string`. Za razvrstavanje operatora koristi se stog s `char` vrijednostima, a `<char, int>` mapa sadrži sve operatore i njihov prioritet (kôd 5.3.) te se kod provjere operatora koristi usporedba s vrijednostima iz mape. Upisivanje

infiksa se vrši bez pisanja razmaka, ali dodavanjem u novi `string` se dodaju i razmaci kako bi se lakše razlikovali operator i predznak minus.

```
/**
     * @brief Conversion::infixToPostfix converts infix string to postfix
     * @param expression entered expression in infix form
     * @return string transformed to postfix
*/
string Conversion::infixToPostfix(string expression)


stack<char> stack1;
map<char, int> precedence = { {'+',1}, {'-',1}, {'*',2}, {'/',2},
{'^',3}, {'S',4}, {'C',4}, {'T',4}, {'K',4}, {'L',4}, {'Q',4} };
```

Kôd 5.3. Zaglavlje metode `infixToPostfix()` i ulomak kôda s stogom i mapom operatora

Kroz sveukupni string izraza se prolazi `for` petljom i za svaki simbol određuje kamo će ići (kôd 5.5.), a potrebno je provjeriti i je li upisano neko slovo koje ne postoji u mapi (kôd 5.4.).

```
if(isalpha(expression[i]) &&
precedence.find(expression[i]) == precedence.end()
&& expression[i] != 'x' && expression[i] != 'y' &&
expression[i] != 'z')
{
return "";
}
```

Kôd 5.4. Ulomak kôda za provjeru postojanja slova u izrazu koja ne označavaju operatore

```
if (isdigit(expression[i]) || (expression[i] == '.') ||
(expression[i] =='-' && (i == 0 || expression[i - 1]
== '(')) || (expression[i] =='x')
|| (expression[i]=='y')  || (expression[i]=='z'))


{
postfix += expression[i];
} else if (expression[i] == '(') {
     stack1.push(expression[i]);
}else if (expression[i] == ')') {
postfix += " ";
while (!stack1.empty())
{
if (stack1.top() == '(')
{
stack1.pop();
break;
}else{
postfix += stack1.top(); stack1.pop();
}
}
} else if (!(precedence.find(expression[i])
 == precedence.end())) {
postfix += " ";
while (!stack1.empty())
{
```

Kôd 5.5. Pregledavanje svakoga znaka posebno i premještanje na stog ili u novi izraz

Nakon izvršavanja `for` petlje, sve sa stoga se `while` petljom prebacuje u novi izraz (kôd 5.6.).

```
while (!stack1.empty())

{

postfix = postfix + " " +
stack1.top();
stack1.pop();

}
```

Kôd 5.6. Dodavanje vrijednosti sa stoga u novodobiveni izraz

Metode `performUnary()` i `performBinary()` koriste se za dobivanje krajnjeg rezultata (pozivaju se za svaki par operand – operator(i)), iz raščlanjenog izraza razaznaju operande i operatore te im omogućuju djelovanje (kôd 5.7.).

```
/**

     * @brief Conversion::PerformBinary performs binary operation
     * based on sent char operator
     * @param a first operand
     * @param b second operand
     * @param op operator to perform
     * @return result of performed operation type double
*/

double Conversion::PerformBinary(double a, double b, char op)


if (op == '+'){

return (double)(a + b);

}else  if  (op  ==  '-'){
return (double)(a- b);

}else  if  (op  ==  '*'){
return (double)(a* b);

}else  if  (op  ==  '/'){
return (double)(a/ b);

}else if (op == '^'){

return (double)(pow(a, b));

}
```

Kôd 5.7. Zaglavlje i implementacija metode `performBinary()`

Metoda `solveEquation()` je ključna i njena povratna vrijednost je najvažnija za cijelu aplikaciju. Ta povratna vrijednost je generirana treća koordinata točke. Prije pozivanja ove metode poznate su samo dvije koordinate svake točke. One se generiraju uz pomoć raspona i stope, dok se treća izračunava. Kao i sve metode za računanje, vraća tip podataka `double` zbog što veće preciznosti. Za argument se uzima `string` i to je prethodno dobiveni postfiksni izraz (kôd 5.8.). Način na koji radi ova metoda je da se postfiksni izraz (rezultat metode `infixToPostfix()`) ponovno raščlanjuje i po određenim pravilima se računa rezultat. Ovoga puta operandi idu izravno na stog, a operatori se redom provjeravaju u primljenom izrazu, jer je metodom `infixToPostfix()` dobiven njihov točan redoslijed. Prvi znak u postfiksnom izrazu je uvijek operand tako da u svakom slučaju prvi znak ide na stog (eventualno i drugi, ovisno o tome da li je operator binarni ili unarni). Ostatak postfiksa može varirati.

Ovoj metodi u startu može biti proslijeđen `string` koji sadrži samo operand odnosno neku konstantu i u tom slučaju je povratna vrijednost jednaka primljenoj. Kroz svaki znak primljenog izraza se iterira samo jedanput i provjerava se je li on operand. Ako je, ide na stog, a ako je operator, provjeri mu se binarnost i uzima sa stoga jedan ili dva najnedavnija operanda nad kojima se taj operator provodi. Dobivena vrijednost se zatim postavi na stog i nastavlja se sa provjerom idućeg znaka primljenog izraza (kôd 5.9. i 5.10.).

```
/**
     * @brief Conversion::SolveEquation takes postfix string as argument,
     * calculates result of whole equation by calling side methods
     * @param eq sent postfix equation
     * @return result of equation type string
*/
double Conversion::SolveEquation(string eq)



stack<double>
stack1;     string
buff = ""; double
x1;

double x2;

double result = 0;
double tmp;

bool operator_found = false;

map<char, int> operator_type = { {'S',1}, {'C',1}, {'T',1},
{'K',1},{'L',1}, {'+',2}, {'-',2}, {'*',2}, {'/',2}, {'^',2} };
```

Kôd 5.8. Zaglavlje i kreirane instance u metodi `solveEquation()`

```
for (unsigned i = 0; i < eq.length(); i++) {
if (isdigit(eq[i]) || eq[i] == '.' || (eq[i] == '-' &&
isdigit(eq[i+1]))) {
      buff += eq[i];
}else {
      if (!buff.empty()){
            stack1.push(stod(buff));
            buff = "";
      }
      if (eq[i] != ' ' && (!(operator_type.find(eq[i])
       == operator_type.end())))
      {
            operator_found = true;
            if (operator_type.at(eq[i]) == 1) {
                  x1 = stack1.top();
                  stack1.pop();
                  tmp = PerformUnary(x1, eq[i]);
                  stack1.push(tmp);
            }
            else {
                  x1 = stack1.top();
                  stack1.pop();
                  x2 = stack1.top();
                  stack1.pop();
                  tmp = PerformBinary(x2, x1, eq[i]);
                  stack1.push(tmp);
            }
      }
}}
```

Kôd 5.9. Provođenje metode `solveEquation()`

```
if (operator_found == true)
{
result = stack1.top();
}else {
result = stod(eq);
```

Kôd 5.10. Rezultat metode, vrijednost sa stoga ili konstanta ovisno o postojanju operatora

Metoda `isWholeInputNumber()` provjerava za unesene vrijednosti točaka ili raspona jesu li ispravno napisani brojevi (kôd 5.11.). Nije moguće koristiti metodu `stod()` (*string to double*) jer je unos u ova polja tipa `QString`, a ne `string`. Ovom metodom se provjeravaju samo raspon, korak i opcionalna polja za izračun specifične točke, ali ne i polje za unos jednadžbe jer u njemu ne mora biti konstantna vrijednost, već može sadržavati operatore.

```
/**
 * @brief Conversion::isWholeInputNumber checks if inputs
 * like range and rate are valid numbers,
 * if there are only digits or x,y,z variables
 * and if it contains a dot, checks if only one.
 * @param input entered expression, may be a number or equation
 * @return true if input number, false if equation
*/
bool Conversion::isWholeInputNumber(string input)



int point_counter = 0;
for(unsigned i = 0; i < input.size(); i++)
{     if(input[i] == '.') {
point_counter++;
}

if(isalpha(input[i]) && input[i] != 'x' &&
input[i] != 'y' && input[i] != 'z'){
return false;
}
```

Kôd 5.11. Provjera da li je cijeli unos konstanta metodom `isWholeInputNumber()`

5.2 Funkcija main

```
#include "mainwindow.h"
#include "conversion.h"
#include <QApplication>
#include <QtDataVisualization>
int main(int argc, char *argv[])
{
QApplication a(argc, argv);
MainWindow w;
w.showMaximized();
return a.exec();
}
```

Kôd 5.12. Funkcija `main()`

Funkcija `main()` je rasterećena. U njoj se kreira objekt klase `QApplication`, grafički element (u ovome slučaju glavni prozor) i prikaže ga se. Klasa `QApplication` se koristi kod grafičkih aplikacija i dodaje se samo jedna bez obzira koliko prozora, dijaloga i drugih grafičkih elemenata ima u aplikaciji. Objektima klase `QApplication` pristupa se metodom `instance()` koja vraća pokazivač na objekt klase `QApplication` (ili `QApp`). Prema [9], klasa `QApplication` upravlja grafičkim aplikacijama, postavkama i tokom kontrola. Objekt klase `QApplication` se u funkciji `main()` inicijalizira prvi i zadužen je za inicijalizaciju svih drugih grafičkih elemenata (kôd 5.12.).

5.3 Mainwindow

U datoteci zaglavlja klasa Mainwindow je definirana unutar `Ui` modula (engl. *namespace*). Modul `Ui` grupira sve automatski generirane prozore. Služi za razlikovanje klasa korisničkog sučelja u dizajneru za kreiranje sučelja i implementacije klase s funkcionalnostima. Klasa `Mainwindow` nasljeđuje klasu `QObject`. Ima implementirane

metode za provjeru valjanosti unosa, brojač znakova „;“ u tekstnim poljima, dodavanje serija podataka na graf i za izračun specifičnih točaka.

Pošto je u aplikaciji moguće maksimalno prikazati 6 površina istovremeno (sasvim dovoljno za analiziranje zajedničkih elemenata, točaka, pravaca i sl.), klasa `MainWindow` sadrži objekte članove, 6 objekata klase `QSurface3DSeries` (sadrže redove vektora s površinom, dijelove oblika površine). Klasa sadrži i broj koji prati koliko je površina dodano i uklonjeno, broj odjeljivača (;) za slučaj unosa dvodijelnih jednadžbi i prekinutosti funkcija. Uključen je i graf koji se kreira odmah u konstruktoru prozora i ne uklanja se, čak ni kad se uklone sve površine s njega (prema kôdu 5.13.).

Osim toga, u klasi postoje još samo `click()` događaji.

```
int surfaceCounter = 0;
QtDataVisualization::Q3DSurface
*graph = new

QtDataVisualization::Q3DSurface();
QtDataVisualization::QSurface3DSeries *series1 = new

QtDataVisualization::QSurface3DSeries();
QtDataVisualization::QSurface3DSeries *series2 = new

QtDataVisualization::QSurface3DSeries();
QtDataVisualization::QSurface3DSeries *series3 = new

QtDataVisualization::QSurface3DSeries();
QtDataVisualization::QSurface3DSeries *series4 = new

QtDataVisualization::QSurface3DSeries();
QtDataVisualization::QSurface3DSeries *series5 = new

QtDataVisualization::QSurface3DSeries();
QtDataVisualization::QSurface3DSeries *series6 = new

QtDataVisualization::QSurface3DSeries();
```

Kôd 5.13. Definiranje grafa i serija podataka u zaglavlju klase `Mainwindow`

U klasu `Mainwindow` je uključena klasa `QFile` za upravljanje datotekama, otvaranje, čitanje i pisanje, preciznije za zapisa JSON objekata i nizova u datoteku s nastavkom „.json“. Klasa `QFile` radi s binarnim formatom datoteka. Prema [10], u klasi `QFile` se obično u konstruktoru navodi ime datoteke ili se naknadno doda metodom

`setFileName()` (kôd 5.14.). Ako nije navedeno drukčije, podaci se u `QFile` zapisuju i čitaju pomoću objekta klase `QTextStream` ili metodama `read()`, `readAll()`, `write()` itd. Također korisne metode za `QFile` su `atEnd()`, koja vraća `bool` ako je dostignut kraj datoteke i `size()`, koja vraća veličinu datoteke.

U konstruktoru glavnog prozora navodi se pokazivač na roditeljski objekt, a to je objekt klase `QWidget`. Metoda `setupUi()` kreira stvarni objekt/grafički element sa svim ostalim elementima na njemu, za razliku od grafičkih elemenata kreiranih u grafičkom sučelju koji se samo spremaju u datoteku s nastavkom „.xml“. U konstruktoru se ujedno i postavljaju predefinirane vrijednosti za raspone i korak, koje korisnik može promijeniti.

Nakon postavljanja grafičkih elemenata, u konstruktoru se također padajući izbornik (engl. *combobox*) popuni podacima koordinata. I kako bi graf bio dodan na površinu (engl. *layout*), kreira se pokazivač na objekt klase `QWidget` koja služi kao omotač (engl. *wrapper*) za graf i koja postaje neposredni objekt roditelj umjesto glavnog prozora. Prikazivanje grafa u omotaču ne utječe na performanse aplikacije.

```
MainWindow::MainWindow(QWidget *parent) :
QMainWindow(parent), ui(new Ui::MainWindow)


{
ui->setupUi(this);
ui->comboBox->addItems(QStringList() << "z=" << "y=" << "x=");

QWidget *container = QWidget::createWindowContainer(graph);

ui->verticalLayout->addWidget(container);
ui->textEdit_11->setText("-10");
ui->textEdit_12->setText("10");
ui->textEdit_13->setText("-10");
ui->textEdit_14->setText("10");
ui->textEdit_15->setText("1");


}
```

Kôd 5.14. Konstruktor klase `Mainwindow`

Klik na botun s tekstom „Add“ je središnji događaj u cijeloj aplikaciji. U polje za unos je obavezno napisati izraz, a sve ostalo je opcionalno i pokriveno zadanim vrijednostima. Odmah na početku provjerava se jesu li svi unosi ispravni i ako jesu, kreiraju se objekti klasa `DataManager` i `Conversion` za pozivanje metoda iz tih klasa. Provjerava se i postoje li identični uneseni podaci u JSON datoteci. Ako neki opcionalni podaci nisu uneseni, u JSON je za vrijednost spremljen prazan string. Samo ako su svi podaci jednaki JSON vrijednostima, iz JSON datoteke se graf popunjava nizom točaka, a u svim ostalim slučajevima točke se generiraju pozivom metode `representDataOnGraph()` (kôd 5.15.).

I da bi se svakim unosom podaci spremili u prvo slobodno tekstno polje, potrebno je prilikom svakoga klika provjeriti polja redom jesu li prazna. Za serije podataka na grafu, odnosno prikazane površine koriste se varijable članovi. Za prikazivanje tih serija podataka

se poziva metoda `addSeries()` koja određuje vrijednost varijable `surfaceCounter`. Vrijednost te varijable je jednaka rednom broju prvog praznog polja za unos i koristi se kako bi se znalo koju točno seriju treba ukloniti ili dodati.

Pošto se u tekstna polja osim unosa dodaje i odabir iz padajućeg izbornika, ako korisnik ništa ne upiše, osigurano je samo da se graf neće iscrtati, ali u tekstna polja će se upisati string iz izbornika. To se lako može ukloniti klikom na botun pored toga polja, koji osim serije podataka briše i tekst iz odgovarajućeg polja. Prilikom klika na botun „Add“ računa se i treća koordinata neke specifične točke ako su unesene preostale dvije. Ova funkcionalnost se koristi ako je potrebna veća preciznost i ako koordinate sadrže velik broj decimala, ili ako korisnik ne može procijeniti s obzirom na uneseni raspon i korak da li se i gdje točka s tim koordinatama nalazi.

Metoda `calculateSpecificPoint()` (kôd 5.17.) poziva pomoćnu metodu `solveEquation()`, ali samo jedanput i nakon završetka metode postavlja rezultat u tekstno polje 10.

```
/**
    * @brief MainWindow::on_pushButton_clicked
    * when clicked, first entered
    * values are saved to variables
    * which are sent to background methods
    * to calculate with them,
    * if some of the inputs are invalid,
    *  text in that field is colored red.
*/
void MainWindow::on_pushButton_clicked()
bool inputOK = checkInputAndSetColor();
 if(inputOK){
Conversion c;
DataManager dm;
QStringList values;
calculateSpecificPoint();
QSurfaceDataArray *array = new QSurfaceDataArray();
QString letter = ui->comboBox->currentText();

QString expression = ui->textEdit->toPlainText();
QString rate = ui->textEdit_15->toPlainText();

values << letter + expression << min1 << max1
<< min2 << max2 << rate << two_part_range;

bool existsInJson = dm.existsInJsonFile(values);
if(existsInJson){
*array = dm.representJsonDataOnGraph(graph);
```

Kôd 5.15. Potvrda unosa, provjera ispravnosti, postojanja u JSON datoteci i iscrtavanje

Klikom na bilo koji botun s natpisom „Remove“ s grafa se metodom `removeSeries()` uklanja odgovarajuća serija podataka i briše tekst iz polja s istim rednim brojem (kôd 5.16.).

```
void MainWindow::on_pushButton_7_clicked(){
ui->textEdit_7->setText("");
graph->removeSeries(series6);
}
```

Kôd 5.16. Uklanjanje površine s grafa i brisanje iz tekstnog polja

U klasi Mainwindow mnogo funkcija služi kao pomoćne drugim funkcijama i pozivaju se unutar njih. Npr. metoda `calculateSpecificPoint()` poziva metodu `solveEquation()`, koja poziva `infixToPostfix()`.

```
/**
    * @brief MainWindow::calculateSpecificPoint if other two
      coordinates are
    * entered,
    * this will calculate third coordinate of that point and set the
      result
    * to corresponding field.
*/
void MainWindow::calculateSpecificPoint()
Conversion c;
string equation = ui->textEdit->toPlainText().toStdString();
string letter = ui->comboBox->currentText().toStdString();
string changed_equation;
string replace1, replace2; double result;
ui->textEdit_10->setText("");
if(letter == "z="){
replace1 = "x"; replace2 = "y";
} else if(letter == "y="){ replace1 = "x"; replace2 = "z";
} else{
replace1 = "y"; replace2 = "z";
}
if(ui->textEdit_8 -> toPlainText() != "" && ui->textEdit_8
->toPlainText() != ""){
changed_equation = c.replaceSymbol(equation,replace1,
ui-> textEdit_8->toPlainText().toDouble());
changed_equation = c.replaceSymbol(changed_equation, replace2 ,
ui -> textEdit_9->toPlainText().toDouble());
result = c.SolveEquation(c.infixToPostfix(changed_equation));
ui->textEdit_10->setText(QString::number(result));
}
```

Kôd 5.17. Izračunavanje specifične točke ako su unesene koordinate

Također su napravljene i metode koje služe kao brojači ili indikatori, npr. `semicolon_counter()` (kod 5.18.) i `isWholeInputNumber()` (kod 5.11.). Mogu se implementirati u nekoj drugoj metodi, ali su izdvojene zbog čitljivosti kôda, odnosno obavljeno je refaktoriranje.

```
/**
    * @brief MainWindow::semicolon_counter checks how many
      semicolons are in
    * some input
    * called to check if two-parts range have 1 semicolon
      more than entered
    * expression
    * @param str entered expression
    * @return number of semicolons
*/
int MainWindow::semicolon_counter(QString str)



int count = 0;
for(unsigned i = 0; i < str.size(); i++)
{
if(str[i] == ';'){count++;}
}
return count;
```

Kôd 5.18. Pomoćna metoda `semicolon_counter()`

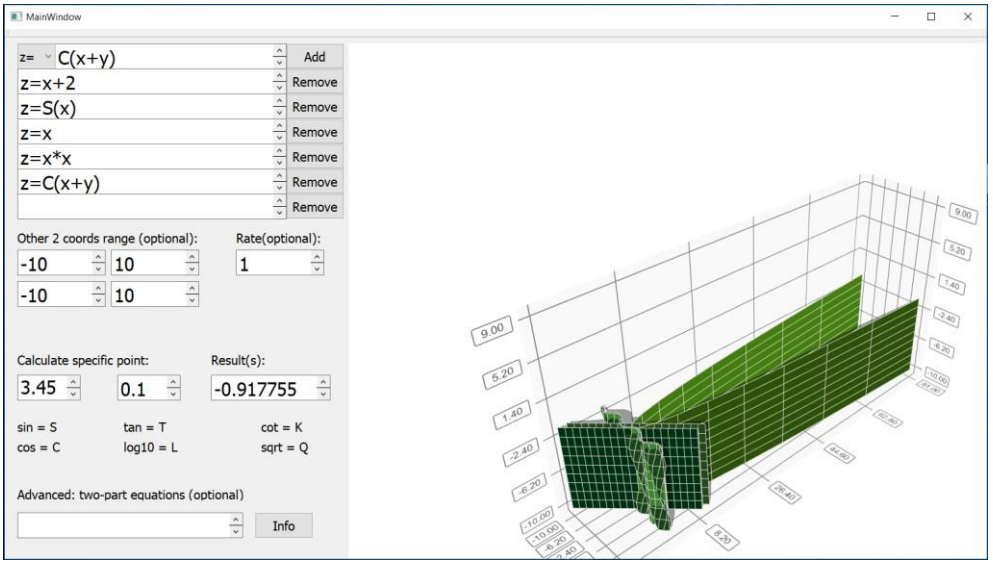
Pošto korisnik može unijeti bilo koji raspon za izračun točaka, za kreiranje objekta klase `QSurfaceDataRow` mora se znati koliko će točaka biti u nizu, pa se uzima razlika maksimuma i minimuma svake koordinate, podijeljeno sa korakom (kôd 5.19.).

```
QSurfaceDataRow(ceil((abs(max2)+abs(min2))/rate));
```

Kôd 5.19. Veličina niza podataka i broj točaka

Objekti klasa `Conversion` i `DataManager` se instanciraju samo za potrebe poziva metoda iz te klase. Ako korisnik unese neispravan izraz i ako nema podataka za računanje, neće doći do kreiranja tih objekata.

Slika 5.20. prikazuje iscrtavanje nekoliko površina i izračun jedne specifične točke, što znači da su svi podaci u tom slučaju ispravno uneseni.



Slika 5.20. Iscrtavanje više površina i prolagođavanje oblika grafa

5.4 Klasa DataManager

U ovoj klasi nalaze se metode za iscrtavanje `QtSurface` grafa, odnosno predstavljanje podataka na grafu.

Ako je unos ispravan, za prikaz podataka poziva se metoda `representDataOnGraph()` ili `representJsonDataOnGraph()`. Kod prve metode važno je poslati koordinatu iz padajućeg izbornika, kako bi se znalo koje dvije koordinate idu u rasponu a koja se računa iz njih. Raspon je opcionalan i zadane vrijednosti su od -10 do 10 za obje preostale koordinate, koje se uzimaju abecednim redom. Ukupni broj točaka na grafu će biti jednak zbroju raspona i eventualno podijeljen sa korakom ako je on

različit od 1. Taj broj se šalje kao parameter objektu klase QSurfaceDataRow.

Točke se spremaju u objekt klase `QVector3D`, koja omogućuje prikaz vektora u 3D prostoru. Svaka točka na nekom poligonu je tzv. tjeme (engl. *vertex*) i mora imati 3 koordinate. A vektori su postavljeni slijedom u objekt klase `QSurfaceDataRow` ili `QSurfaceDataArray`. U `QSurfaceDataRow` vektori se pozicioniraju metodom `setPosition()`, na navedeni indeks se postavi tjeme, a taj indeks se uglavnom određuje pomoću raspona druge koordinate (kôd 5.21.). Svakim povećavanjem prve koordinate postavi se na nulu i iteriranjem kroz unutrašnju petlju se povećava. Spremanjem koordinata u točku također se ta točka sprema u JSON datoteku jer tu ide posljednji unos. Vrijednosti iz tekstnih polja, padajućeg izbornika i ostalih grafičkih elemenata se u JSON upisuju prije generiranja točaka.

```
if(letter == "z=") {

changedExpression = c.replaceSymbol(expression, "x", i);
changedExpression = c.replaceSymbol(changedExpression, "y", j);

double result =
c.SolveEquation(c.infixToPostfix(changedExpression));
pointsData << QString::number(i) + ";" + QString::number(j) +

";" + QString::number(result) + ";" +
QString::number(position);
(*row)[position].setPosition(QVector3D(i, j, result));

}
```

Kôd 5.21. Pozicioniranje vektora u 3D grafu u metodi `representDataOnGraph()`

Metoda `representDataOnGraph()` isto tako prima `QJsonObject` objekt koji je rezultat metode `createJsonFile()`. U toj metodi se otvara datoteka „parseddata.json“ za čitanje i formira se `QJsonObject` koji sadrži samo unesene podatke. Nakon generiranja točaka se poziva metoda `updateJsonFile()` koja dodaje koordinate točaka u datoteku.

```
QSurfaceDataArray
DataManager::representDataOnGraph(std::string letter,
QList<QString> values, Q3DSurface *graph, QJsonObject
jsonobject)
```

Kôd 5.22. Argumenti metode `representDataOnGraph()`, JSON objekt, vrijednosti i graf

Na taj način se graf može iscrtavati vađenjem podataka iz JSON datoteke, sa spremljenim generiranim podacima i pozicijom. Otvara se JSON datoteka za čitanje i dohvati se niz podataka spremljen pod nazivom „points_on_graph“, koji sadrži točke. Za dohvaćanje niza koristi se metoda `value()` i spremaju u objekt `klase QJasonValue`, zatim se taj objekt pretvara u niz (kôd 5.23).

Prema kôdu 5.22., za poziv metode `representDataOnGraph()` potrebno je znati odabir padajućeg izbornika (varijabla `letter`), listu podataka, odnosno unesene vrijednosti u tekstna polja (varijabla `values`), graf i objekt klase `QJsonObject` koji sadrži posljednje spremljene podatke u JSON datoteci. Vrijednosti iz tekstnih polja se uspoređuju s tim podacima.

```
QJsonValue value =
jsonobject.value("points_on_graph";

QJsonArray pointsArray =
value.toArray();
```

Kôd 5.23. Metode za dohvaćanje JSON niza iz datoteke

Svaki objekt unutar JSON niza je tipa `QJsonValue` i iz takvog tipa podataka mogu se dohvaćati svojstva, u ovom slučaju koordinate i pozicija od kojih se kreira `QSurfaceDataRow` niz. Kod svakog objekta unutar niza, ako ima poziciju 0, to znači da

treba inicijalizirati novi `QSurfaceDataRow` niz, jer u jednom nizu ima onoliko točaka koliko i različitih brojeva pozicija (kôd 5.24.). Obje metode za prikazivanje površina na grafu vraćaju pokazivač na objekt klase `QSurfaceDataArray`, koji se kasnije koristi kao argument za dodavanje serija podataka u glavnom prozoru.

```
Q_FOREACH(const QJsonValue &v, pointsArray) {
position =
v.toObject().value("position").toInt();
if(position == 0){
row = new QSurfaceDataRow(ceil((abs(min2) + abs(max2))/rate));
}

(*row)[position].setPosition(QVector3D(v.toObject()
.value("x").toInt(),
v. toObject().value("y").toInt(),v.toObject().value("z").toInt()));
*array << row;
}
```

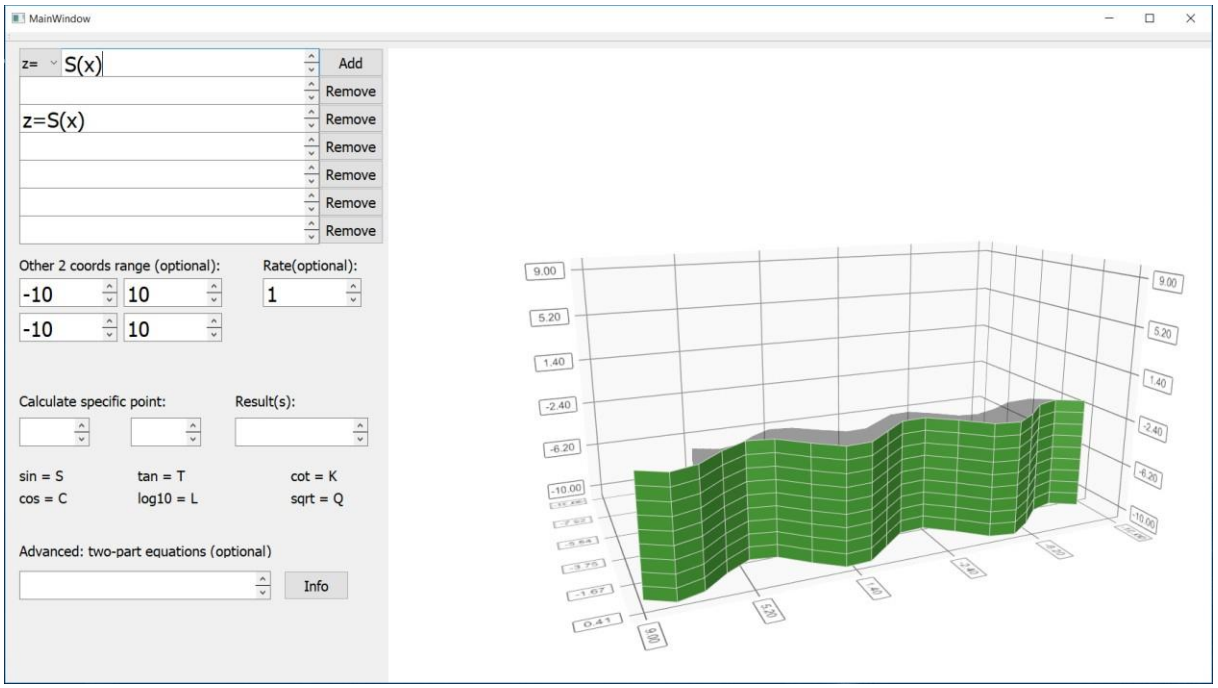
Kôd 5.24. Pozicioniranje vektora u metodi `representJsonDataOnGraph()`

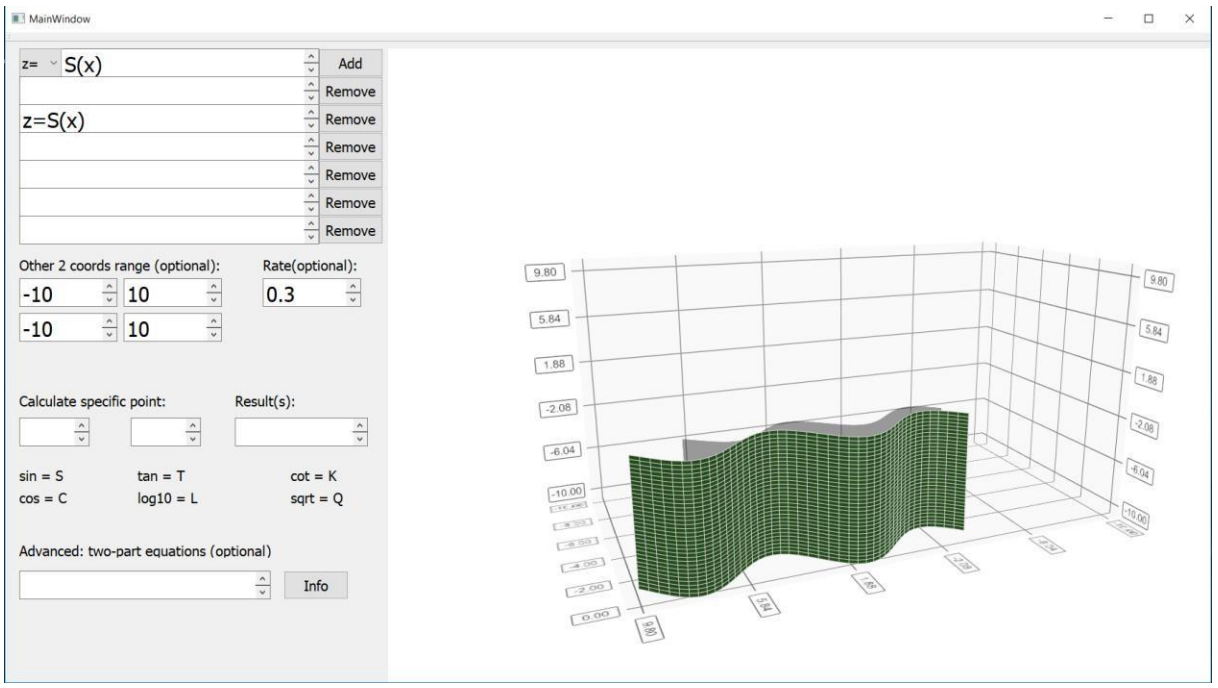
U metodi `existsInJsonFile()` se čita JSON datoteka i uspoređuju vrijednosti s unesenim, povratna vrijednost je tipa `bool` te služi za određivanje hoće li se podaci prikazivati iz json datoteke ili generiranjem točaka (kôd 5.25.).

```
QFile jsonfile("parseddata.json");

QJsonDocument jsondocument;

QJsonObject jsonobject;

QByteArray bytearray;

bool exists = false;

if(jsonfile.open(QIODevice::ReadOnly)){

bytearray = jsonfile.readAll();

jsondocument = jsondocument.fromJson(bytearray);

jsonobject = jsondocument.object();


if(jsonobject["equation_id"] == values[0] &&
jsonobject["coord1_min"]

== values[1] && jsonobject["coord1_max"] == values[2] &&
jsonobject["coord2_min"] == values[3] &&
jsonobject["coord2_max"] == values[4] && jsonobject["rate"] ==
values[5] && jsonobject["advanced"] == values[6]){

exists = true;

}


return exists;
```

Kôd 5.25. Provjera postoji li izraz u JSON datoteci

Slike 5.26. i 5.27. prikazuju iscrtavanje funkcije sinus, odnosno razliku u izgledu površine s korakom 1 i korakom 10.





Slika 5.26. i 5.27. Površina za funkciju sinus s korakom 1 i 0.3

5.5 Grafičko sučelje

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
<property name="geometry">
<rect>
<x>0</x>
<y>0</y>
<width>1638</width>
<height>986</height>
</rect>
</property>
<property name="font">
<font>
<pointsize>12</pointsize>
</font>
</property>
<property name="windowTitle">
<string>MainWindow</string>
</property>
<widget class="QWidget" name="centralWidget">
<widget class="QTextEdit" name="textEdit">
<property name="geometry">
```

Kôd 5.28. ulomak iz xml datoteke s podacima o widgetima

Objekt roditelj svih komponenti na grafičkom sučelju je glavni prozor. U lijevome dijelu prozora su widgeti s kojima korisnik može interaktirati. Gornju traku čine padajući izbornik, polje za unos teksta i botun za potvrdu unosa. Kôd 5.28. prikazuje spremanje podataka o glavnom prozoru.

Unos je u obliku eksplicitne matematičke jednadžbe gdje se jedna nepoznanica (koordinata) izražava preko preostale dvije. U padajućem izborniku stavke su koordinate uz znak jednakosti, a u tekstno polje se unosi desna strana eksplicitne jednadžbe i samo se taj dio jednadžbe računa a ostatak sprema u odabranu varijablu. Ispod se nalazi još 6 tekstnih polja, što znači da ukupno može biti najviše 6 jednadžbi istovremeno prikazano na grafu, u svrhu uspoređivanja površina, traženja zajedničkih točaka, nagiba i sl. Polje u koje se unosi jednadžba ne služi za spremanje tih podataka nego samo za prikaz, a u ostala polja se ti podaci spremaju zajedno sa izrazom iz padajućeg izbornika na početku, odnosno sprema se cijela jednadžba, klikom na prvi botun s tekstom „Add“.

Prije odluke u koje polje će se zapisati jednadžba, na svako okidanje `click()` događaja provjerava se postojanje podataka u tekstnom polju, počevši od prvog prema zadnjem. U prvo slobodno polje se zapiše jednadžba, a ako nema slobodnog polja, ne događa se ništa.

Tu počinje poveznica s algoritmom *shunting yard* i pretvorbom u postfiks. U polje za unos upisuje se isključivo infiksni oblik jednadžbe, nad bilo kojim drugim izrazima neće se provoditi postupak i klikom na dodavanje takve strukture korisnik će vidjeti upozorenje, boja teksta se mijenja u crvenu, te ako nakon toga korisnik unese ispravan izraz i potvrdi ga, tekst ponovno mijenja boju u crnu što je osigurano u metodi `checkInputAndSetColor()`. Ona također vraća `bool` vrijednost o ispravnosti svih podataka, pozivajući `isWholeInputNumber()` iz klase `Conversion` i `isExpressionValid()` iz klase `DataManager`.

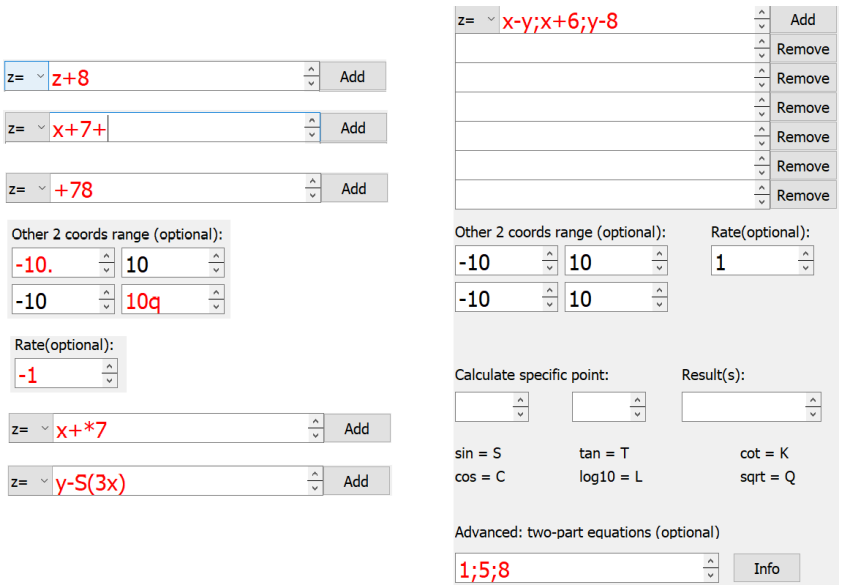
Tip unesenih podataka je `QString`, a za pozivanje metoda podaci se transformiraju u `std string` pomoću ugrađene metode `toStdString()`, jer se nisu sve metode koje vrijede za `string` kompatibilne sa tipom podataka `QString`.

Graf se u ui dizajneru nalazi u zasebnom layout-u zbog poravnanja i da je razdvojen od forme za unos. U slučaju sljedećih neispravnih unosa tekst mijenja boju u crvenu:

- Unos dvaju binarnih operatora za redom (unos binarnog pa unarnog operatora je valjan, npr. x+C(x)).

- Operator na kraju izraza.
- Binarni operator na početku izraza, a da taj operator nije minus.
- Unos slova i znakova kojima nisu kodirani operatori (ovdje se svrstava i unos s izostavljenim zagradama tamo gdje su potrebne, npr. za izraz „3*x“ ne smije se pisati „3x“).
- Vrijednosti za raspon, korak i koordinate specifične točke nisu brojevne, odnosno metoda `isWholeInputNumber()` vraća vrijednost `false`.
- Ako je unesen raspon za dvodijelne jednadžbe i ne sadrži znak „;“ za 1 više nego polje za unos jednadžbe.
- Ako uneseni izraz sadrži vrijednost iz padajućeg izbornika, npr „z=z+6“.
- Za negativan korak ili u vrijednosti 0.

Na slici 5.29. prikazani su svi oblici pogrešnog unosa podataka gdje neće doći do izračuna rezultata i iscrtavanja površina. Ako je unos ispravan, izvršava se kôd 5.30.



Slika 5.29. Primjeri pogrešnih unosa podataka

```
switch (surfaceCounter)
{
case 1:
series1->dataProxy()->resetArray(array);
graph->addSeries(series1);
break;



case 2:
series2->dataProxy()->resetArray(array);
graph->addSeries(series2);
break;
}
```

Kôd 5.30. Dodavanje serija podataka na graf i stvaranje novog niza pomoću metode `resetArray()`

6. Zaključak

Iz svega opisanoga može se zaključiti da je razvojni okvir Qt jako pogodan za razne matematičke procese, zbog ugrađenih grafova, serija podataka i mnogo opcija za grafičke aplikacije. Ovaj završni rad, preciznije aplikacijski dio je jedno od rješenja pri pokušajima automatizacije računanja i očitavanja podataka.

Automatizacija ima prednosti i mane u odnosu na ručnu provedbu tih procesa, za neke operacije je računalo brže od čovjeka, ali za neke sporije te nekada je potrebno dodatno izgubiti vrijeme na neke procese i postupno obavljanje, koje su ljudskom mozgu očigledne pa je moguće čak i preskakanje pojedinih koraka.

Ali za iscrtavanje grafa je isplativije koristiti aplikaciju nego računati dvije točke, što traje duže i veća je vjerojatnost pogreške, stoga je ovom aplikacijom rješen problem brzine i točnosti računanja podataka i omogućeno ono što je samo na računalu i moguće, točan i vjerodostojan prikaz podataka, trodimenzionalno crtanje i omogućeno posmatranje ravnina iz više kutova. Također *shunting yard* i slični algoritmi dodatno olakšavaju automatiziranje i precizno transformiraju izraze da je krajnji rezultat posve točan poredak operacija.

Veza između signala i prijemnika je svakako jedna od najboljih karakteristika cijelog razvojnog okvira te mogućnost neovisnog povezivanja na više signala/prijemnika, kao što je to u primjeru dodavanja (potvrde unosa) jednadžbe, čije emitiranje poziva i upis u tekstna polja i postavljanje na graf te računanje bilo koje treće koordinate s ostale dvije poznate, sve to istovremeno.

Korišteni JSON format je koristan za pamćenje posljednjeg unesenog izraza, ali za spremanje više izraza je nepogodno, jer bi iteriranje kroz datoteku trajalo dugo ovisno o broju spremljenih objekata.

Također svakim upisom novih podataka u JSON datoteku metodom write(), brišu se postojeći podaci i ne postoji append metoda koja samo nadograđuje datoteku. Jedno od rješenja bilo bi sve iz datoteke prebaciti u listu JSON objekata, te izračunom novih podataka u datoteku dodati sveukupne podatke, ali to je dugotrajan proces i pozivao bi se na svaku potvrdu unosa.

Aplikacija, u odnosu na neke druge aplikacije ovoga tipa ima prednosti i mane, dodatne mogućnosti i propuste, ali osnovna namjena je ispunjena i moguće je lako nadograditi funkcionalnost.

U uređivaču QtCreator većina grafičkih aplikacija koje uključuju grafove su aplikacije u programskom jeziku QML koje podatke zapisuju pomoću nizova u koje ručno upisuju koordinate jednu po jednu ili implementiraju određenu funkciju i prikažu je. Ova aplikacija proširuje te mogućnosti na bilo koje unesene funkcije. I omogućeno je korisničko definiranje rate, odnosno koraka između koordinata, što utječe na glatkoću teksture površine i gustoću točaka.

Ovisno o daljnjim potrebama korisnika, aplikaciju je moguće i nadograditi.

7. Literatura

[1] https://doc.qt.io/qt-5/ Qt dokumentacija, travanj 2019.

[2] https://en.wikipedia.org/wiki/Shunting-yard_algorithm Shunting yard algorithm wikipedia, prosinac 2018.

[3] https://wiki.qt.io/About_Qt What is QT?, prosinac 2018.

[4] https://repozitorij.etfos.hr/islandora/object/etfos:1460/preview Robert Veseli, Izrada aplikacije koristeći QT skup programskih alata, travanj 2019.

[5] https://doc.qt.io/qt-5/qtdatavisualization-index.html Qt Data Visualization, siječanj 2019.

[6] https://www.mathblog.dk/tools/infix-postfix-converter/ Pretvarač infiksa u postfiks, siječanj 2019.

[7] https://www.json.org/ Introducing JSON, travanj 2019.

[8] https://doc.qt.io/qt-5/json.html JSON podrška u Qt-u, travanj 2019.

[9] https://doc.qt.io/qt-5/qapplication.html QApplication klasa, siječanj 2019.

[10] https://doc.qt.io/qt-5/qfile.html QFile klasa, travanj 2019.