Qt5 aplikacija za vizualizaciju podataka

Vujasić, Paula

Qt5 aplikacija za vizualizaciju podataka

Vujasić, Paula

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Paula Vujasić

QT5 APLIKACIJA ZA VIZUALIZACIJU PODATAKA

Diplomski rad

Voditelj rada:
prof. dr. sc. Mladen Jurak

Zagreb, veljača, 2020.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Rad posvećujem obitelji, prijateljima i mentoru koji su vjerovali u njegovu realizaciju i onda kada ja nisam.

Sadržaj

Uvod

Qt5 je najnovija verzija Qt biblioteke, razvojnog paketa pogodnog za izradu aplikacija s grafičkim korisničkim sučeljem. Qt dolazi s vlastitim razvojnim okruženjem pod nazivom Qt Creator. Unutar njega integrirana su dva različita uređivača grafičkog sučelja: Qt Designer i Qt Quick Designer. Prvi se koristi za izradu desktop aplikacija, a drugi kod izrade aplikacija za mobilne i ugrađene platforme.

U radu su opisane ključne značajke Qt5 biblioteke, s naglaskom na Qt Widgets aplikaciju - standardni način dizajniranja grafičkog korisničkog sučelja za desktop aplikacije omogućen pomoću Qt Designera. U sklopu rada konstruirana je desktop aplikacija za vizualizaciju podataka "Harry Plotter". Aplikacija omogućuje prikazivanje grafova funkcija jedne varijable u različitim stilovima.

Rad je podijeljen u dva poglavlja. U prvom poglavlju opisani su ključni dijelovi Qt biblioteke, poput meta-objektnog sustava i mehanizma signala i utora, te su potom povezani s oblikovnim obrascima Composite i Observer. U nastavku poglavlja predstavljeni su dijelovi Qt biblioteke potrebni za razvoj aplikacije te Qt Charts modul pomoću kojeg je u Qt-u već omogućeno crtanje različitih vrsta grafikona.

U drugom poglavlju opisana je aplikacija. Na početku se razmatraju postojeća rješenja za crtanje grafova funkcija. Među njima su biblioteke QCustomPlot i Matplotlib, koje su poslužile kao inspiracija u izradi aplikacije. Potom slijedi detaljan opis funkcionalnosti aplikacije i njene implementacije.

Poglavlje 1

Qt5 biblioteka

Qt je *open-source* biblioteka za izradu grafičkih korisničkih sučelja i višeplatformskih aplikacija. Neke od podržanih platformi su Linux, Windows, macOS, Android i ugrađeni sustavi. Osmislili su ga početkom 90-ih Haavard Nord i Eirik Chamble-Eng s ciljem izrade objektno-orijentiranog softverskog okvira s grafičkim sučeljem.

Qt-ov SDK (*software development kit*) sadrži širok raspon alata i biblioteka koji olakšavaju pisanje koda, te se programeri ne moraju brinuti o tehničkim problemima koji se odnose na određenu platformu. Također, Qt pruža gotovo potpunu zamjenu za `STL`-ove klase i tipove. Dobro je dokumentiran i slijedi skup konvencija o imenovanju.

Qt5 je najnovija verzija Qt-a. Ova verzija donosi mnogo novih značajki, te Qt čini vrlo moćnim i stabilnim razvojnim paketom. Qt5 olakšava rješavanje najnovijih promjena paradigme korisničkog sučelja koje zahtijevaju zasloni osjetljivi na dodir i tableti. Također, u Qt5 je pojednostavljena i prenosivost na više platformi. [6]

Qt je velika biblioteka koja sadrži mnogo manjih biblioteka i modula.
Među popularnijima su:

- Qt Core - sadrži osnovne klase poput `QObject`, `QThread`, `QFile`, `QVariant`
- Qt GUI - sadrži klase za grafičko korisničko sučelje
- Qt Widgets - omogućuje stvaranje klasičnih korisničkih sučelja za desktop aplikacije
- Qt SQL - sadrži klase za komunikaciju s `SQL` bazama podataka
- Qt Network - sadrži klase koje olakšavaju mrežno programiranje
- Qt Charts - sadrži klase za crtanje različitih vrsta grafikona

1.1 Qt Creator

Qt dolazi s vlastitim integriranim razvojnim okruženjem (IDE), pod nazivom Qt Creator. Dizajniran je kako bi olakšao razvoj C++ aplikacija koje koriste Qt. Sastoji se od uređivača koda i dizajnera grafičkog korisničkog sučelja integriranih s drugim Qt alatima, kao što su prevodilac (*compiler*) i program za pronalaženje pogrešaka (*debugger*).

Qt Creator omogućuje upravljanje projektima te koristi projektnu `qmake .pro` datoteku. Sadrži sustav pomoći te omogućuje automatsko testiranje, kao i izradu instalacijskih paketa za mobilne uređaje.

Uređivač koda nudi inteligentno dovršavanje koda, isticanje sintakse, poruke o pogreškama i upozorenjima za vrijeme kodiranja i alate za brzu navigaciju kroz kod. Qt Creator također uključuje i ugrađeni preglednik dokumentacije koji se zove `Qt Assistant`. Pomoću njega moguće je na jednostavan način potražiti objašnjenje za određenu Qt klasu ili funkciju.

Dizajner grafičkog korisničkog sučelja dolazi s dva različite vrste uređivača, a to su `Qt Designer` i `Qt Quick Designer`. Za stvaranje intuitivnih korisničkih sučelja modernog izgleda pogodniji je Qt Quick Designer, dok je za tradicionalno, jasno strukturirano korisničko sučelje pogodniji Qt Designer. Zbog toga se Qt Designer obično koristi kod dizajniranja sučelja za desktop aplikacije, a Qt Quick Designer za mobilne i ugrađene platforme. [1, 6]

Više o Qt Designeru reći ćemo u potpoglavlju 1.6. Razvoj Qt Quick aplikacija omogućuje `Qt Quick modul`, a grafičko korisničko sučelje piše se u QML jeziku.

Qt Quick i QML

Qt Quick modul pruža sve što je potrebno za stvaranje aplikacije s fluidnim i dinamičnim korisničkim sučeljem. Naglasak pri izgradnji korisničkih sučelja je na ponašanju komponenti i načinu na koji se komponente međusobno povezuju. Qt Quick omogućuje glatke animirane prijelaze i elemente koji reagiraju na dodir. Također, nudi i vizualno platno s vlastitim koordinatnim sustavom i mehanizmom za prikaz.

QML je deklarativni jezik za dizajn aplikacija usredotočenih na korisničko sučelje. Koristi sintaksu sličnu JSON-u i omogućuje da se različiti izrazi i metode definiraju kao JavaScript funkcije. Time se programerima i dizajnerima olakšava razvoj korisničkih sučelja i logike aplikacije. Moguće je pisati cijele aplikacije samo u QML-u, ali obično se u QML-u piše samo korisničko sučelje, a ostatak aplikacije se implementira u imperativnom C++ jeziku. [6]

1.2 Meta-objektni sustav

Meta-objektni sustav omogućuje upotrebu jedne od središnjih značajki Qt-a: signale i utore. Također, omogućuje identifikaciju tipova tokom izvršavanja programa (RTTI) i dinamički sustav svojstava objekata. Meta-objektni sustav temelji se na `QObject` klasi, `Q_OBJECT` makrou te meta-objektnom kompajleru.

`QObject` klasa je osnovna klasa za objekte koji mogu iskoristiti navedene prednosti meta-objektnog sustava. Među važim klasama koje ju nasljeđuju su: `QWidget`, `QEvent`, `QApplication`, `QLayout`. U daljnjem tekstu potpoglavlja poistovjetit ćemo klasu koja nasljeđuje `QObject` s klasom `QObject`.

Svaka `QObject` klasa ima statičku varijablu članicu `QMetaObject`, dakle jedna se instanca kreira za svaku `QObject` klasu koja se koristi u aplikaciji. Ta instanca pohranjuje sve meta-informacije za `QObject`, a također održava i tablicu korespodencije između signala i utora. Svakom signalu i svakom utoru pridružen je jedinstveni indeks čime je omogućena optimizacija kompajlera upotrebom `jump` instrukcije. [2, 6]

`Q_OBJECT` makro

Makro `Q_OBJECT` mora se pojaviti u privatnom dijelu definicije klase koja implementira signale i utore ili koristi druge usluge koje nudi Qt-ov meta-objektni sustav. Iako je i bez njega moguće koristiti `QObject` klasu, preporučljivo je uvijek koristiti ovaj makro. Osim što su u suprotnom uskraćene usluge poput mehanizma signala i utora, klasa koja nasljeđuje `QObject` je u meta-objektnom sustavu ekvivalentna najbližem pretku s meta-kodom. Zbog toga, primjerice, funkcija `QMetaObject::className()` neće vratiti ime odgovarajuće klase, već njezinog pretka. [6]

Meta-objektni kompajler

Meta-objektni kompajler (*moc*) opskrbljuje svaku potklasu `QObject`-a potrebnim kodom za implementaciju značajki meta-objekta. Stil kodiranja koji se koristi u većini Qt programa nije "čisti" C++. Primjer toga je `Q_OBJECT` makro. Zadaća *moc*-a je da prije kompilacije iz makroa generira potreban C++ kod.

Meta-objektni kompajler čita izvornu C++ datoteku. Ako pronađe barem jednu deklaraciju klase koja sadrži makro `Q_OBJECT`, stvara novu izvornu C++ datoteku pod imenom `moc_filename.cpp` koja sadrži dodatne funkcije za svaku od klasa s `Q_OBJECT` makroom. Zbog toga se Qt aplikacije i biblioteke mogu kompilirati pomoću bilo kojeg kompatibilnog standardnog C++ kompajlera kao što su *Clang*, *GCC*, *MinGW* itd. [4, 6]

1.3 Signali i utori

Mehanizam signala i utora služi za komunikaciju između objekata. Qt koristi ovaj fleksibilan mehanizam još od samih početaka, kada je izašao na tržište sredinom 90-ih. Koncept signala i utora vrlo je značajan - obrazac i terminologija korišteni su iznova i iznova u raznim implementacijama. Upotrebu signala i utora omogućuje Qt-ov meta-objektni sustav.

Postizanje komunikacije među objektima različitih tipova u drugim se softverskim okvirima uglavnom ostvaruje pomoću povratnih poziva (*callback*) - pokazivača na funkciju. Povratni pozivi nisu idealno rješenje, mogu biti neintuitivni i mogu imati problema s osiguravanjem ispravnosti argumenata.

Qt je uveo mehanizam signala i utora kao alternativu tehnici povratnog poziva. Signal se emitira kada se dogodi određeni događaj. Utor je funkcija koja se poziva kao odgovor na određeni signal.

Sve klase koje nasljeđuju `QObject` ili neku od njenih potklasa mogu koristiti signale i utore. Za Qt-ove `QWidget` potklase postoji mnogo već unaprijed definiranih signala i utora. Na primjer, pritiskom na gumb može se slati signal čija je posljedica izvršavanje odgovarajuće funkcije.

Signali i utori mogu primati bilo koji broj argumenata bilo koje vrste. Međutim, kako bismo mogli spojiti signal i utor, broj, redoslijed i tip argumenata moraju se podudarati. Budući da deklaracije signala i utora moraju biti kompatibilne, kompajler nam može pomoći u otkrivanju neusklađenosti argumenata.

Jedan signal može biti povezan s različitim utorima, a isto tako i jedan utor može biti pozvan emitiranjem različitih signala s kojima je povezan. Veza između signala i utora ostvaruje se pomoću `connect` funkcije:

```
connect(pošiljatelj, &Pošiljatelj::imeSignala,
        primatelj, &Primatelj::imeUtora);
```

Signali

Signal je poruka koju šalje objekt. Predstavljen je u definiciji klase kao deklaracija javne funkcije bez povratnog tipa (void funkcija). Može imati popis argumenata, ali nema tijelo funkcije.

Bilo koji `QObject` koji ima signal može ga emitirati pomoću naredbe `emit`. Kad je signal emitiran, utori povezani na njega obično se odmah izvršavaju, kao kod normalnih poziva funkcija.

Objekt koji emitira signal ne zna koji su utori povezani s njime niti brine je li pozvan ijedan utor.

Utori

Utori su C++ funkcije na koje se mogu povezati signali.

Utor se poziva kada je poslan na njega povezan signal. Budući da su utori normalne funkcije članice, mogu se i direktno pozvati pri čemu se slijede normalna C++ pravila za poziv funkcija. Međutim, signal emitiran od proizvoljnog objekta uvijek poziva utor s kojim je povezan, čak i ako je utor definiran kao privatna funkcija.

Ako je na jedan signal spojeno više utora, oni će se nakon emitiranja signala izvršavati jedan za drugim, redoslijedom kojim su spojeni. Utori također mogu biti i virtualne funkcije.

Baš kao što objekt ne zna odazivaju li se utori na njegove signale, utor ne zna je li s njim spojen ijedan signal. Time je osigurana slaba povezanost signala i utora, odnosno neovisnost Qt-ovih komponenata.
[2, 6, 4]

1.4 Model-View programiranje

MVC

Model-View-Controller (MVC) je obrazac koji se često koristi u aplikacijama s korisničkim sučeljem, a potječe iz programskog jezika `Smalltalk-80`. Cilj obrasca je odvajanje poslovne logike od korisničkog sučelja.

MVC se sastoji od tri cjeline:

- Model (*model*) - podaci i poslovna logika određene aplikacije
- Pogled (*view*) - prezentacija podataka
- Upravitelj (*controller*) - upravljanje korisničkim zahtjevima, ”most” između modela i pogleda

Model može imati više pogleda koji na različite načine prezentiraju podatke. Pogled mora odražavati stanje modela. Kad god se podaci modela promijene, model obavještava poglede koji ovise o njemu te se potom svaki pogled ažurira na odgovarajući način. Ovaj pristup omogućuje dodavanje i uređivanje korisničkih sučelja bez utjecaja na logički sloj.
[3, 5]

Qt-ova realizacija MVC obrasca

Ako se objekti pogleda i upravitelja kombiniraju, rezultat je *model-view* arhitektura. Ovakva arhitektura još uvijek razdvaja način pohrane podataka od načina na koji se oni prezenti-

raju korisniku, ali je jednostavnija od MVC obrasca. Kako bi se omogućilo fleksibilno rukovanje korisničkim unosom, u Qt-u je uveden koncept delegata. Delegati omogućuju prilagodbu prikaza i uređivanje podataka.

U Qt-u se klase *model-view* arhitekture mogu podijeliti u tri skupine: na modele, poglede i delegate. Za svaku od ove tri skupine definirana je apstraktna klasa koje pruža zajedničko sučelje i zadane implementacije nekih značajki. Modeli, pogledi i delegati međusobno komuniciraju koristeći signale i utore. [6]

Velik dio mehanizma upravitelja iz MVC obrasca u Qt-u se može naći u delegatima. Osnovna apstraktna klasa za delegate je `QAbstractItemDelegate`. Pogledi osiguravaju klase delegata koje su prikladne za većinu svrha, međutim moguće je implementirati i prilagođene delegate. Pritom se preporučuje nasljeđivanje klase `QStyledItemDelegate` koja koristi zadani stil pri prikazu podataka. Kôd koji povezuje signale i utore također se može smatrati kodom upravitelja. [4]

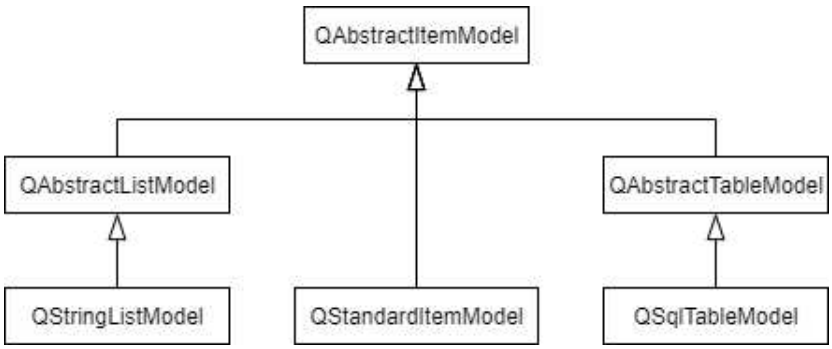
Modeli

Sami podaci ne moraju se pohranjivati u modelu; mogu se primjerice nalaziti u datoteci ili bazi podataka.

Svi su modeli izvedeni iz `QAbstractItemModel` klase. Ova klasa pruža dovoljno fleksibilno sučelje za poglede koji prikazuju podatke u obliku tablica, lista i stabala. Međutim, pri implementaciji novih modela za tablične podatke ili podatke sa strukturom liste, bolje polazišne točke su `QAbstractTableModel` i `QAbstractListModel` u kojima su implementirane odgovarajuće često potrebne funkcije. [6]

U Qt-u postoje i gotovi modeli:

- `QStringListModel` - za spremanje jednostavne liste QString objekata
- `QStandardItemModel` - upravlja složenijim strukturama u obliku stabala koje mogu sadržavati proizvoljne podatke
- `QSqlQueryModel`, `QSqlTableModel`, `QSqlRelationalTableModel` - za pristup bazama podataka
- `QFileSystemModel` - pruža informacije o mapama i datotekama u lokalnom sustavu arhiviranja

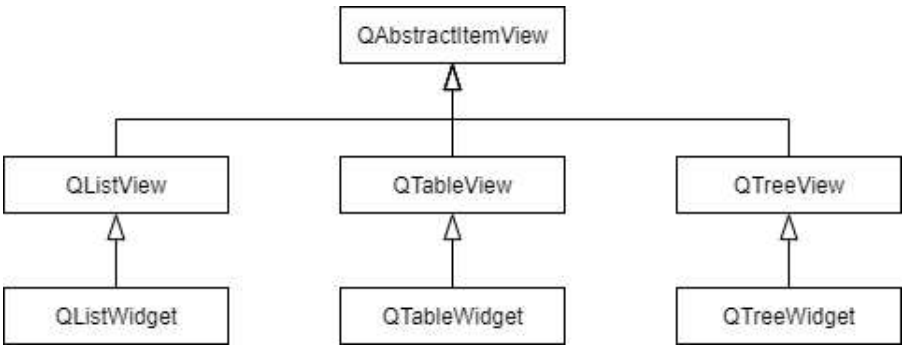


Slika 1.1: Klase modela

Pogledi

U Qt-u postoje gotove implementacije za različite vrste pogleda: `QListView` prikazuje podatke iz modela u listi, `QTableView` podatke prikazuje u tablici, a `QTreeView` u hijerarhijskoj strukturi. Svaka od ovih klasa temelji se na apstraktnoj klasi `QAbstractItemView`. Iako su ove klase spremne za upotrebu, mogu se nasljeđivati kako bi se kreirali prilagođeni pogledi.

Qt nudi klase za prikaz podataka u obliku liste (`QListWidget`), tablice (`QTableWidget`) ili stabla (`QTreeWidget`) u kojima se spremaju podatci. Takve "convenience" klase naslijeđene su iz odgovarajućih klasa pogleda i `QWidget` klase. Ove klase praktičnije su za upotrebu i lakše za razumijevanje, međutim pohranjivanje podataka unutar njih dovodi do snažne ovisnosti između korisničkog sučelja i temeljne strukture podataka. Manje su fleksibilne od klasa pogleda i ne mogu se koristiti s proizvoljnim modelima. [6, 4]



Slika 1.2: Klase pogleda i iz njih izvedene klase

1.5 Oblikovni obrasci u Qt-u

Oblikovni obrasci

Oblikovni obrasci (*design patterns*) elegantna su i učinkovita rješenja problema koji se često javljaju u objektno orijentiranom programiranju. Svaki obrazac predstavlja apstraktno rješenje jednog oblikovnog problema. Oblikovni obrasci olakšavaju ponovno korištenje uspješnog dizajna i arhitekture.

Pri opisivanju oblikovnih obrazaca obično se koriste sljedeći elementi:

- Ime - olakšava komunikaciju među programerima
- Problem - opisuje u kojim situacijama treba primijeniti obrazac
- Rješenje - opisuje elemente koji čine dizajn i njihove međusobne odnose
- Posljedice - rezultati i kompromisi primjene obrasca

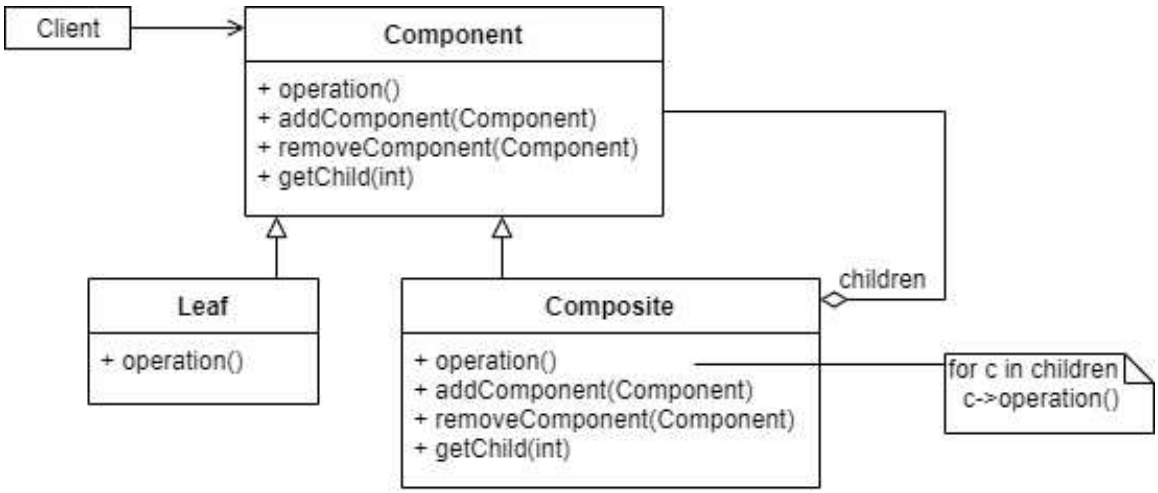
Oblikovni obrasci obično se dijele u 3 grupe, prema ulozi koju imaju:

- Obrasci stvaranja - opisuju proces stvaranja objekata: Factory Method, Builder, Prototype, Singleton...
- Strukturni obrasci - opisuju kako organizirati objekte ili klase i povezati ih: Composite, Façade, Adapter, Bridge...
- Obrasci ponašanja - opisuju načine na koji objekti ili klase međusobno djeluju i raspodjeljuju odgovornost: Observer, Command, Sate, Visitor... [5]

Brojni oblikovni obrasci korišteni su u razvoju Qt-ovih klasa i mehanizama. Osim Composite i Observer obrazaca koje ćemo detaljnije opisati, među korištenima su: Command (klase `QAction`, `QUndoCommand`, `QRunnable`; također, Command je objektno orijentirana zamjena za metodu povratnog poziva), Façade (klasa `QFile`), Monostate (klasa `QSettings`), Serializer (klase `QTextStream`, `QDataStream`), Singleton (klasa `QApplication`). [4]

Composite

Composite obrazac strukturira složeni objekt kao stablo koje predstavlja hijerarhiju sastavnih dijelova. Cilj obrasca je da klijent na isti način može tretirati i individualni objekt i kompoziciju objekata. Rješenje se bazira na tome da ista apstraktna klasa (`Component`) predstavlja i složeni objekt (klasa `Composite`) i elementarni objekt (klasa `Leaf`) nudeći implementaciju koja im je zajednička. [7]



Slika 1.3: Struktura Composite obrasca

Uobičajeni primjer Composite obrasca je sustav arhiviranja na računalu. Mapa (složeni objekt) može sadržavati datoteke (individualni objekti), ali i druge mape. Međutim, rekurzivna kompozicija Composite obrasca pogodna je za više od dokumenata. Može se koristiti za predstavljanje bilo koje hijerarhijske strukture.

Brojne Qt-ove klase koriste Composite obrazac: `QObject`, `QWidget`, `QTreeWidgetItem`, `QDomNode`, `QHelpContentItem`, `QResource`. [4]

QObject i Composite obrazac

Svaki objekt klase `QObject` može imati i roditelja i djecu. Odnos roditelj-dijete između `QObject` objekata predstavlja odnos cjelina-dio u Composite obrascu. U hijerarhijskoj `QObject` strukturi objekti koji imaju djecu mogu se smatrati *Composite* objektima, a objekti bez djece *Leaf* objektima.

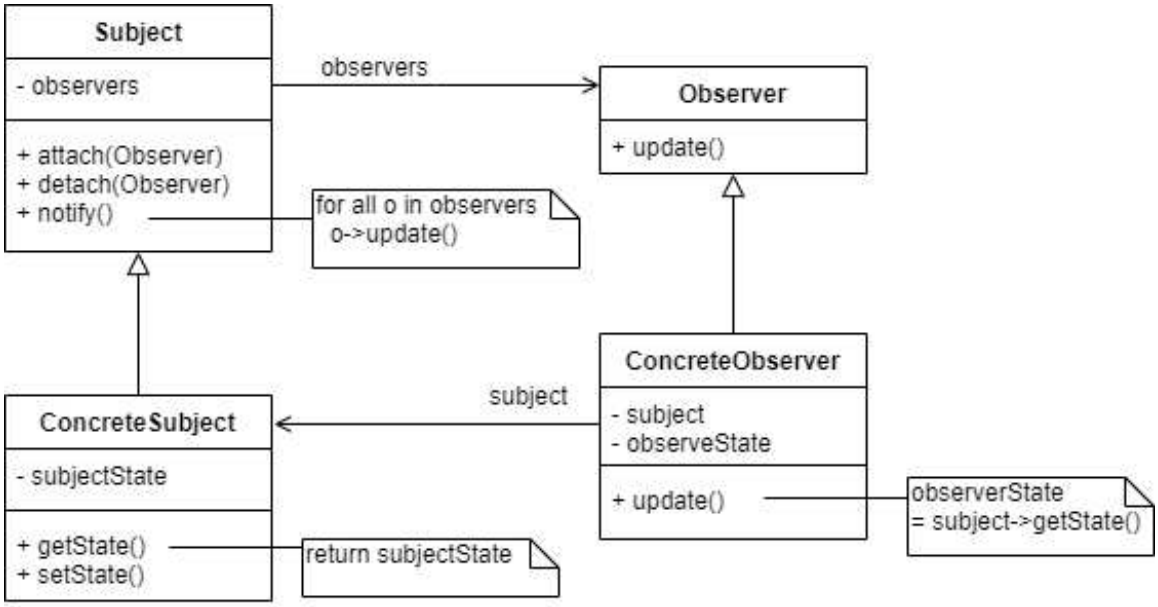
U `QObject` klasi roditelji preuzimaju vlasništvo nad djecom čuvajući njihove pokazivače u listi. Posljedica toga je da mogu brinuti o uništavanju dječjih objekata. Roditelj u destruktoru automatski briše svoju djecu.

Napomenimo još da `QObject` klasa nema konstruktor kopiranjem niti operator pridruživanja kopiranjem zbog čega QObject nikad ne može biti proslijeđen u funkciju po vrijednosti. Makro `Q_DISABLE_COPY(QObject)` eksplicitno osigurava da se nijedan `QObject` ne može kopirati. Razlog tome je upravo odnos roditelja i djece u klasi. Naime, svako dijete može imati samo jednog roditelja. [4]

Observer

Obrazac Observer koristi se za stvaranje objekata koji promatraju promjene u drugom objektu i na odgovarajući način reagiraju na njih. Potreban je kod objekata koji ovise o trenutnom stanju drugog objekta. Primjerice, numerički podaci, koji se mogu prikazivati na različite načine, bit će spremljeni u jedan objekt - subjekt (klasa `Subject`), dok će ih razni drugi objekti - promatrači (klasa `Observer`) prikazivati (pomoću tablice, grafikona itd.). Svaki promatrač registrira se kod subjekta, a subjekt dozvoljava da se kod njega registrira proizvoljno velik broj promatrača.

Klase `ConcreteSubject` i `ConcreteObserver` nasljeđuju apstraktne klase `Subject` i `Observer`, čime se omogućuje neovisno variranje tipova subjekta i promatrača. [7]



Slika 1.4: Struktura Observer obrasca

Ova vrsta interakcije poznata je i pod imenom *publish – subscribe* (objavljivanje-pretplata). Subjekt objavljuje obavijesti ne znajući tko su promatrači. Bilo koji broj promatrača može se pretplatiti na primanje obavijesti. [5]

Qt-ov mehanizam signala i utora generalizirana je implementacija Observer obrasca [8]. Signal je subjekt, a utori promatrači. Sa signalom se može povezati proizvoljan broj utora koji se pozivaju njegovim emitiranjem. Signal ne zna koji su utori povezani s njime.

Također, primjenu Observer obrasca u Qt-u ostvaruje i arhitektura modela i pogleda. Svaki pogled (promatrač) mora odražavati stanje modela (subjekt). Kad god se podaci modela promijene, model obavještava poglede koji o njemu ovise. Kao odgovor, svaki se pogled ažurira na odgovarajući način. Na jedan model moguće je dodati više pogleda koji na različite načine prezentiraju podatke.

1.6 Qt Widgets aplikacija

Jedna od glavnih prednosti korištenja Qt-a za razvoj aplikacija je ta što je vrlo lako dizajnirati grafičko korisničko sučelje (GUI).

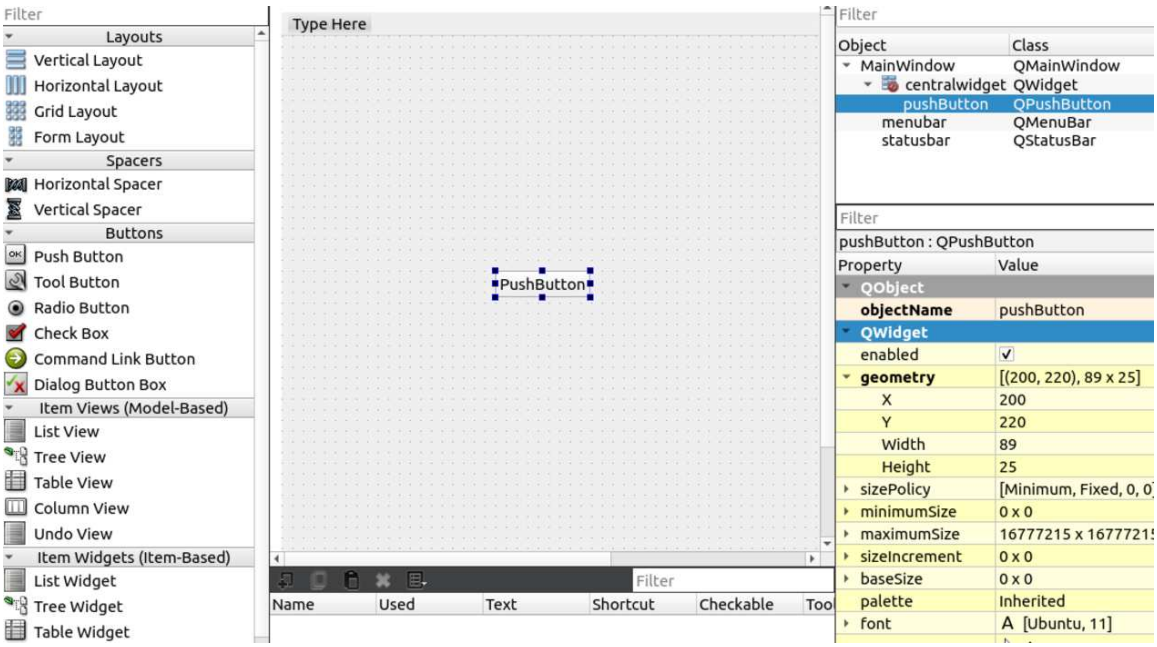
Postoje dvije vrste GUI aplikacija u Qt-u: Qt Quick aplikacija i Qt Widgets aplikacija. Razvoj grafičkog korisničkog sučelja Qt Quick aplikacija omogućen je pomoću Qt Quick Designera, dok se Qt Widgets aplikacije razvijaju pomoću Qt Designera. U ovom potpoglavlju bavimo se Qt Widgets aplikacijama, koje su zasnovane na Qt Widgets modulu i standardni su način dizajniranja grafičkog korisničkog sučelja za desktop aplikacije.

Qt Designer

Qt Designer je Qt-ov alat za izradu grafičkih korisničkih sučelja. Omogućuje jednostavno dodavanje i uređivanje widgeta te funkcionira po *what-you-see-is-what-you-get* (WYSIWYG) principu.

GUI datoteka koju sprema Qt Designer nosi ekstenziju `.ui` te se sprema u XML formatu. Ova datoteka pohranjuje atribute svakog widgeta: poziciju, boju, veličinu, margine itd. Atributi se mogu mijenjati pomoću uređivača svojstava (Property Editor). Pomoću njega se mogu mijenjati i imena widgeta, što je vrlo važno jer ona postaju imena varijabla članica u klasi koju generira User Interface Compiler (*uic*). Zadaća *uic*-a je čitanje datoteke koju generira Qt Designer i generiranje C++ datoteke sa `ui` klasom.

Qt Designer je potpuno integriran u Qt Creator. Sva se svojstva postavljena u njemu mogu dinamički mijenjati unutar koda. Pomoću Qt Designera moguće je dodijeliti ponašanje widgetima koristeći mehanizam signala i utora. [1, 4, 6]

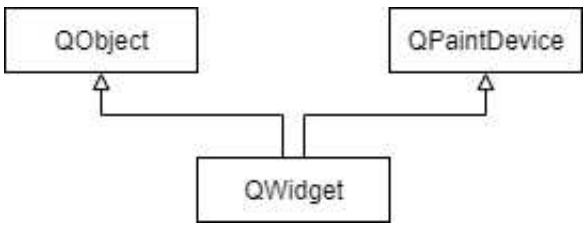


Slika 1.5: Qt Designer

QWidget klasa

Widgeti su osnovni elementi za kreiranje korisničkih sučelja u Qt-u. Widget je objekt izveden iz `QWidget` klase.

`QWidget` klasa pruža mogućnosti prikaza na zaslonu i rukovanja događajima unosa korisnika. To je omogućeno zahvaljujući višestrukom nasljeđivanju. `QWidget` nasljeđuje `QObject` klasu, zbog čega može imati djecu, roditelja, te koristiti mehanizam signala i utora. `QWidget` nasljeđuje i `QPaintDevice` klasu, koja je osnovna klasa za sve objekte koji se mogu prikazati na zaslonu. [4]



Slika 1.6: Klase koje QWidget nasljeđuje

U grafičkom korisničkom sučelju obično su vidljivi odnosi između roditelja i djece - unutar roditeljskih widgeta nalaze se dječji. Dječji widgeti prikazuju se po *layout* pravilima. Widget koji nema roditelja naziva se prozor.

Postoje četiri kategorije osnovnih widgeta:

- Widgeti gumba (*button widgets*) - npr. `QPushButton`, `QCheckBox`, `QRadioButton`, `QToolButton`
- Widgeti unosa (*input widgets*) - služe za unos korisnika, npr. `QLineEdit`, `QSpinBox`, `QComboBox`
- Widgeti prikaza (*display widgets*) - nisu interaktivni, npr. `QLabel`, `QProgressBar`, `QPixmap`
- Widgeti spremnika (*container widgets*) - sadrže druge widgete, npr. `QMainWindow`, `QFrame`, `QToolBar`

Opisani widgeti koriste se pri kreiranju kompleksnijih widgeta, poput dijaloga (`QFileDialog`, `QInputDialog`, `QErrorMessage`) i "pogleda" (`QListView`, `QTreeView`, `QColumnView`, `QTableView`). Također, moguće je izraditi i prilagođene widgete.

Postoje Qt klase koje nemaju grafički prikaz (ne nasljeđuju `QWidget` klasu), ali se koriste u razvoju grafičkog korisničkog sučelja. Među njima su klase za rad s grafičkim objektima, poput `QColor`, `QImage`, `QSize`, te apstraktna klasa `QLayout` i iz nje izvedene klase pomoću kojih se određuje razmještaj elemenata unutar roditeljskih widgeta. [6, 4]

Qt Style Sheets

Qt Widgets aplikacija koristi sustav za oblikovanje pod imenom `Qt Style Sheets`, koji je nastao po uzoru na Cascading Style Sheet (CSS), sustav za oblikovanje web tehnologije. Budući da je Qt Style Sheets inspiriran CSS-om, sintaksa im je vrlo slična. Potrebno je samo napisati opis stila widgeta i Qt će ga prikazati u skladu s tim.

Qt Style sheets sadrži niza pravila oblikovanja. Svako se pravilo sastoji se od selektora i deklaracije. Selektor određuje na koje widgete se odnosi pravilo, a deklaracija opisuje svojstva koja widget treba poprimiti. Primjerice:

```
QLabel {color:  blue;}
```

U navedenom primjeru `QLabel` je selektor, a {color: blue;} deklaracija. Navedeno pravilo određuje da svi `QLabel` objekti trebaju imati plavu boju teksta. Svaka se deklaracija sastoji od atributa i vrijednosti odvojenih dvotočkom.

Oblikovanje widgeta pomoću Style Sheet-a moguće je promijeniti na dva načina: koristeći Property Editor ili C++ kôd direktno.
Svojstva objekta unutar koda možemo promijeniti pozivajući `QObject::setStyleSheet()` funkciju:

```
myLabel->setStyleSheet("{color:  blue;}");
```

Isti efekt možemo postići i mijenjajući `styleSheet` svojstvo u Qt Designeru:

```
QLabel#myLabel{color:  blue;}
```

Qt Style Sheets, poput CSS-a, podržava različite tipove selektora. U prvom primjeru korišten je selektor tipa (*Type Selector*), koji odgovara svim instancama navedene klase i njenih potklasa, dok je u trećem primjeru korišten ID selektor koji odgovara svim instancama navedene klase čije ime je myLabel. [1, 6]

1.7 Graphics View

Softverski okvir Graphics View dio je Qt Widgets modula. Graphics View djeluje poput platna na koji se može nacrtati bilo što pomoću C++ koda, npr. različiti oblici, linije, tekst, čak i slike. Graphics View pruža površinu za upravljanje i interakciju s velikim brojem 2D grafičkih predmeta. Ta je površina prikazana pomoću `QGraphicsView` klase, koja ima podršku za zumiranje i rotaciju. [1]

Međutim, `QGraphicsView` je samo widget koji prikazuje sadržaj scene. U sceni je zapravo pohranjen sadržaj Graphics View-a, a ostvarena je pomoću `QGraphicsScene` klase.

Scena

Scena pohranjuje sve grafičke komponente u hijerarhiju zasnovanu na odnosu roditelja i djece. Ona služi kao spremnik za instance `QGraphicsItem` klase, koje se dodaju pozivom funkcije `QGraphicsScene::addItem()`.

Scena ima sljedeće odgovornosti:

- Nudi brzo sučelje za upravljanje velikim brojem grafičkih komponenata.
- Propagira događaje grafičkim komponentama. Primjerice, kada korisnik prijeđe mišem preko komponente na sceni, `QGraphicsView` transformira taj događaj u događaj scene, koja ga potom propagira određenoj komponenti.
- Upravlja stanjima grafičkih komponenata, poput selektiranja komponente i fokusa. Moguće je selektirati komponentu na sceni, kao i dohvatiti sve odabrane komponente. Također, moguće je utvrditi ima li određena komponenta fokus, te pronaći komponentu koja ima fokus.
- Omogućuje prikaz dijelova scene na nekom od uređaja za crtanje (npr. `QImage`) pomoću funkcije `QGraphicsScene::render()`. [6]

QGraphicsItem klasa

`QGraphicsItem` je apstraktna bazna klasa za grafičke komponente na sceni. Ova klasa ne nasljeđuje `QObject`, što znači da se na grafičkim komponentama ne može koristiti mehanizam signala i utora. Takva implementacija odabrana je zbog boljih performansi, kako bi se tisuće grafičkih komponenti mogle brzo prikazati na ekranu.

Međutim, objekti klase `QGraphicsItem` reagiraju na događaje koje im propagira scena, poput klika ili prelaska mišem. Također, vrlo je jednostavno omogućiti vršenje radnji poput premještanja ili selektiranja komponenata - potrebno je samo postaviti određene ”zastave” (npr. `ItemIsMovable`, `ItemIsSelectable`) pozivom funkcije `QGraphicsItem::setFlag()`. Ove zastave inače su onemogućene zbog boljih performansi.

Qt pruža nekoliko standardnih klasa za tipične oblike koje nasljeđuju QGraphicsItem, poput linija (`QGraphicsLineItem`), pravokutnika (`QGraphicsRectItem`), elipsi (`QGraphicsEllipseItem`) i tekstualnih stavki (`QGraphicsTextItem`).

`QGraphicsItem` objekti mogu se grupirati pomoću `QGraphicsItemGroup` klase. Grupiranje grafičkih komponenti vrlo je korisno jer transformacije roditeljskih komponenti nasljeđuju sva njegova djeca. Također, grupirane komponente mogu se zajedno pomicati.

Stil prikazivanja grafičkih komponenti definira se pomoću `QPen` i `QBrush` klasa. `QBrush` obično definira boju pozadine ili ispunjenja komponente, a `QPen` boju obruba. [6]

Koordinatni sustav

Pozicije komponenata na sceni reprezentirane su uređenim parovima cijelih brojeva - x i y koordinatama. Graphics View se temelji na tri koordinatna sustava; razlikujemo koordinate komponenti (*item coordinates*), koordinate scene (*scene coordinates*) i koordinate prikaza (*view coordinates*).

Koordinate prikaza

Ovaj koordinatni sustav izravno je povezan s `QGraphicsView` widgetom - svaka jedinica u prikazu koordinata odgovara jednom pikselu "platna". Na ovaj koordinatni sustav ne utječe scena koju widget prikazuje. Gornji lijevi ugao platna uvijek ima koordinate (0, 0), a donji desni koordinate (*širina platna*, *visina platna*).

Koordinate scene

Koordinate scene predstavljaju osnovni koordinatni sustav za sve u njoj spremljene grafičke komponente. Originalno, koordinate scene podudaraju se s koordinatama prikaza, međutim moguće ih je promijeniti. Primjerice, scena može prikazivati i negativne koordinate.

Svaka grafička komponenta ima informaciju o položaju na sceni te o pravokutniku koji je ograničava u odnosu na scenu (funkcije `QGraphicsItem::scenePos()` i `QGraphicsItem::sceneBoundingRect()`).

Koordinate komponente

Grafičke komponente imaju svoj lokalni koordinatni sustav. Koordinate komponente centrirane su oko središnje točke komponente s koordinatama (0, 0). Ovaj je koordinatni sustav vrlo koristan pri izradi prilagođenih elemenata - potrebno je brinuti samo o koordinatama komponente, a `QGraphicsScene` i `QGraphicsView` izvršavaju sve potrebne transformacije.

Funkcija `QGraphicsItem::pos()` vraća poziciju grafičke komponente. Pozicija je određena koordinatom središnje točke predmeta u koordinatnom sustavu roditelja. Primjerice, ako dječju komponentu smjestimo točno u središnju točku roditelja, tada će koordinatni sustavi dviju komponenti biti identični. Pozicija komponenti koje nemaju roditelja određuje se koordinatom njihove središnje točke u odnosu na koordinate scene.

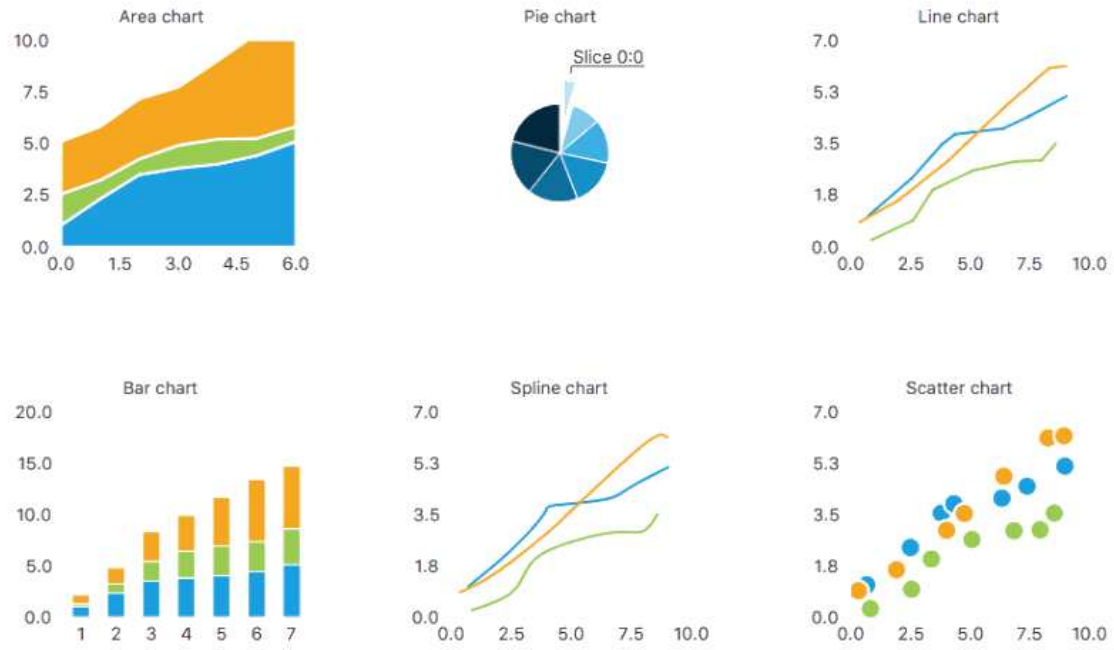
Ostale funkcije `QGraphicsItem` klase djeluju u koordinatama komponente, bez obzira na roditeljske objekte. Na primjer, koordinate ograničavajućeg pravokutnika (`QGraphicsItem::boundingRect()`) uvijek se određuju u koordinatama komponente. [6]

1.8 Qt Charts

Qt Charts modul omogućuje jednostavno crtanje različitih vrsta grafikona. Koristi Qt-ov softverski okvir Graphics View. Qt Charts prethodno je bio samo komercijalni modul, ali od verzije Qt 5.7 postao je dostupan svim *open-source* korisnicima pod `GPLv3` licencom.

Ovaj modul omogućuje crtanje često korištenih vrsta grafikona. Neki od njih su:

- Linijski i splajn grafikoni (*line, spline charts*)
- Stupčasti grafikoni (*bar charts*)
- Tortni grafikoni (*pie charts*)
- Površinski i raspršeni grafikoni (*area, scatter charts*)



Slika 1.7: Vrste grafikona u Qt-u - preuzeto iz [6]

Izgled grafikona može se prilagoditi korištenjem tema, izmjenom boja i svojstava te onemogućavanjem prikaza sastavnih dijelova grafikona. Također, grafikone je moguće animirati.

Qt charts modul sastoji se od niza klasa i funkcija zahvaljujući kojima možemo na elegantan način kreirati grafikone ne znajući kako algoritam djeluje iza kulisa.

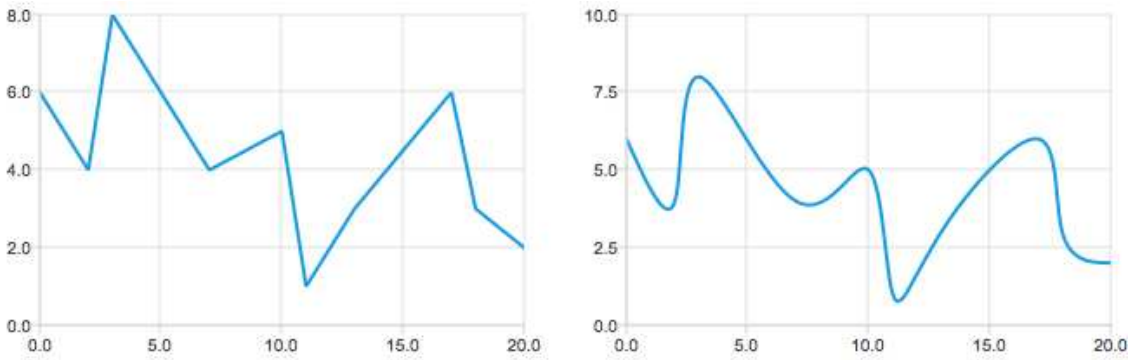
Klasa `QChart` upravlja prikazom različitih vrsta podataka, ali i drugih objekata povezanih s grafikonima, poput legende i koordinatnih osi. `QChart` nasljeđuje `QGraphicsWidget`, klasu čiji se objekti mogu prikazati na sceni. (`QGraphicsWidget` nasljeđuje `QGraphicsItem` pružajući dodatne funkcionalnosti poput odabira stila i fonta.)

Za prikaz `QChart` objekata nije nužno potrebna scena. Grafikon se na jednostavniji način može prikazati pomoću `QChartView` klase. `QChartView` je widget koji rukuje prikazom grafikona.

Podatke koje grafikoni prikazuju spremaju se u klase izvedene iz apstraktne klase `QAbstractSeries` (primjerice: `QLineSeries`, `QBarSeries`, `QPieSeries`). Ti se podatci pridružuju grafikonu pomoću funkcije `QChart::addSeries()`, nakon čega grafikon preuzima vlasništvo nad objektom izvedene klase. S obzirom na vrstu podataka pridruženih grafikonu, prikazuje se odgovarajuća vrstu grafikona. [2, 6]

Linijski i splajn grafikoni

Linijski i splajn grafikoni prikazuju podatke kao niz točaka povezanih linijama. U linijskom grafikonu točke su povezane ravnim linijama, dok su u splajn grafikonu povezane krivuljom.



Slika 1.8: Linijski i splajn grafikoni - preuzeto iz [6]

Linijski grafikon koristi `QLineSeries` klasu za spremanje podataka, a splajn grafikon `QSplineSeries` klasu, koja nasljeđuje `QLineSeries`. `QSplineSeries` sprema točke podataka, ali i kontrolne točke potrebne za crtanje splajna. Kontrolne točke se automatski računaju pri svakoj promjeni podataka. [6]

Poglavlje 2

Aplikacija za vizualizaciju podataka

”Harry Plotter” je desktop aplikacija za crtanje grafova funkcija jedne varijable. Napravljena je kao `Qt Widgets` aplikacija pomoću Qt Creatora te u njega ugrađenog uređivača grafičkog sučelja `Qt Designer`. Pisana je u C++ jeziku te koristi Qt-ove specifičnosti, poput mehanizma signala i utora.

2.1 Postojeća rješenja

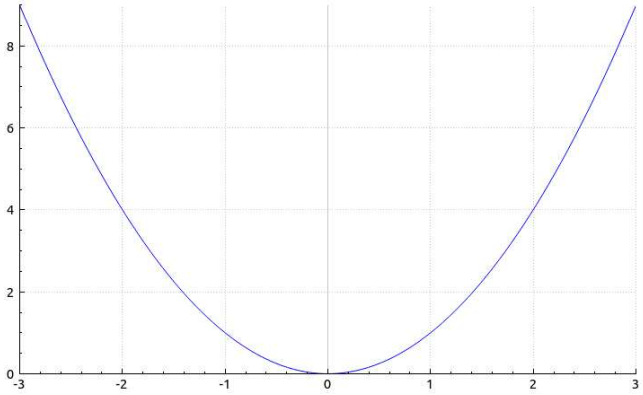
Crtanje grafova funkcija u Qt-u je omogućeno pomoću Qt Charts modula, preciznije, pomoću `QLineSeries`, odnosno iz nje izvedene `QSplineSeries` klase.

Osim Qt Charts modula, aplikacija je inspirirana bibliotekom `QCustomPlot` i Pythonovom bibliotekom za vizualizaciju `Matplotlib`. Navedene biblioteke bile su inspiracija za izgled prikaza grafa, postavljanje koordinatnih osi, ali i za iscrtavanje funkcija s asimptotskim ponašanjem na zadanom rasponu (npr. funkcija $1/x$).

U navedenim rješenjima, grafovi se crtaju na temelju zadanih točaka. Kod Qt Charts modula, točke se dodaju u `QLineSeries` (`QSplineSeries`) klasu funkcijom `append()`. Koordinate točaka u QCustomPlotu zadaju se pomoću Qt-ovog `QVector`-a, a u Matplotlibu pomoću Pythonovog `array`-a.

QCustomPlot

QCustomPlot je biblioteka za crtanje grafova funkcija i vizualizaciju podataka. Služi za izradu 2D grafikona visoke kvalitete s različitim mogućnostima prilagodbe izgleda figura. QCustomPlot widget može se koristiti kao rješenje za prikaz podataka u stvarnom vremenu unutar različitih aplikacija za vizualizaciju. U grafičko sučelje dodaje se promoviranjem `QWidget` u `QCustomPlot` klasu. [10]



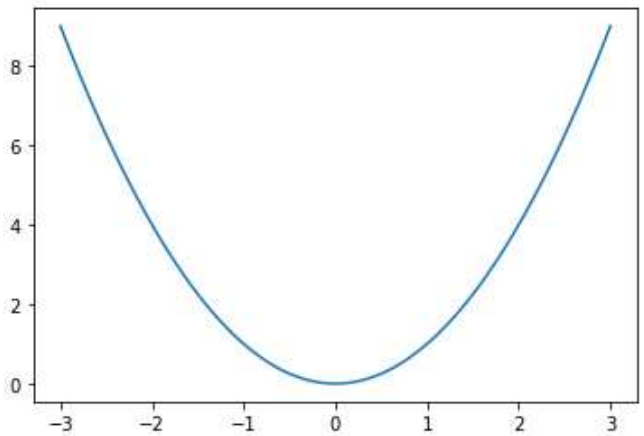

```
QVector<double> x(100), y(100);
x = linspace(-3, 3, 100);
for(int i = 0; i < 100; ++i)
    y[i] = x[i]*x[i];

ui->customPlot->addGraph();
ui->customPlot->graph(0)->setData(x, y);
ui->customPlot->xAxis->setRange(-3, 3);
ui->customPlot->yAxis->setRange(0, 9);
ui->customPlot->replot();
```

Slika 2.1: Primjer QCustomPlot grafa s pripadajućim kodom

Matplotlib

Matplotlib je Pythonova 2D biblioteka za vizualizaciju podataka. Omogućuje jednostavno crtanje različitih vrsta grafikona; u svega nekoliko linija koda moguće je generirati grafove funkcija, histograme, stupčaste dijagrame itd. Matplotlib se može ugraditi u nekoliko alata za izradu grafičkog korisničkog sučelja.

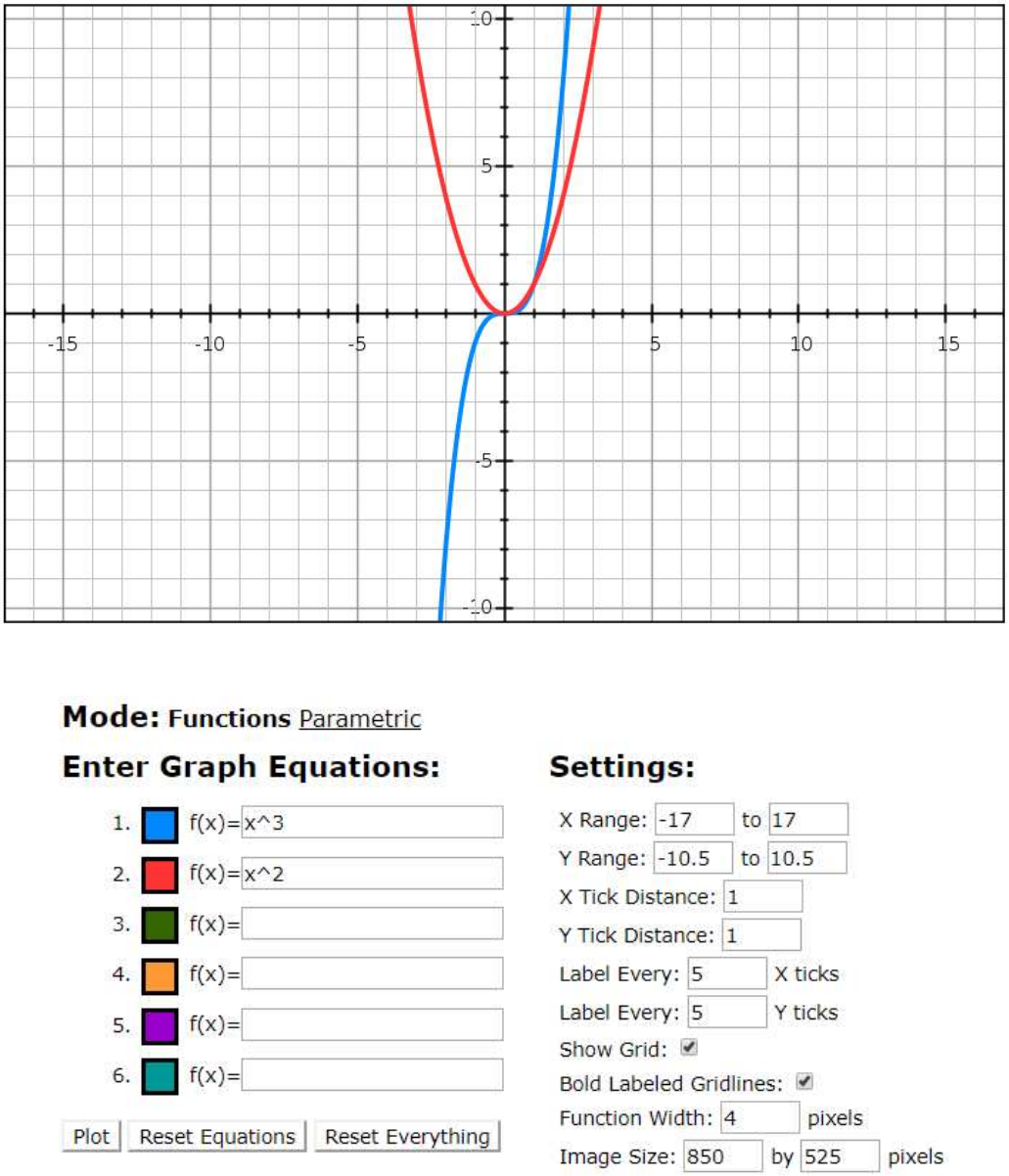
Matplotlib sadrži `pyplot` modul, kolekciju funkcija naredbenog stila kojima je na jednostavan način omogućeno mijenjanje različitih svojstava figura, poput odabira područja crtanja, prikaza koordinatne mreže, izgleda osi, fonta i linija. [11]




```
x = numpy.linspace(-3, 3, 100)
y = numpy.power(x, 2)

plt.plot(x, y)
plt.show()
```

Slika 2.2: Primjer Matplotlib grafa s pripadajućim kodom

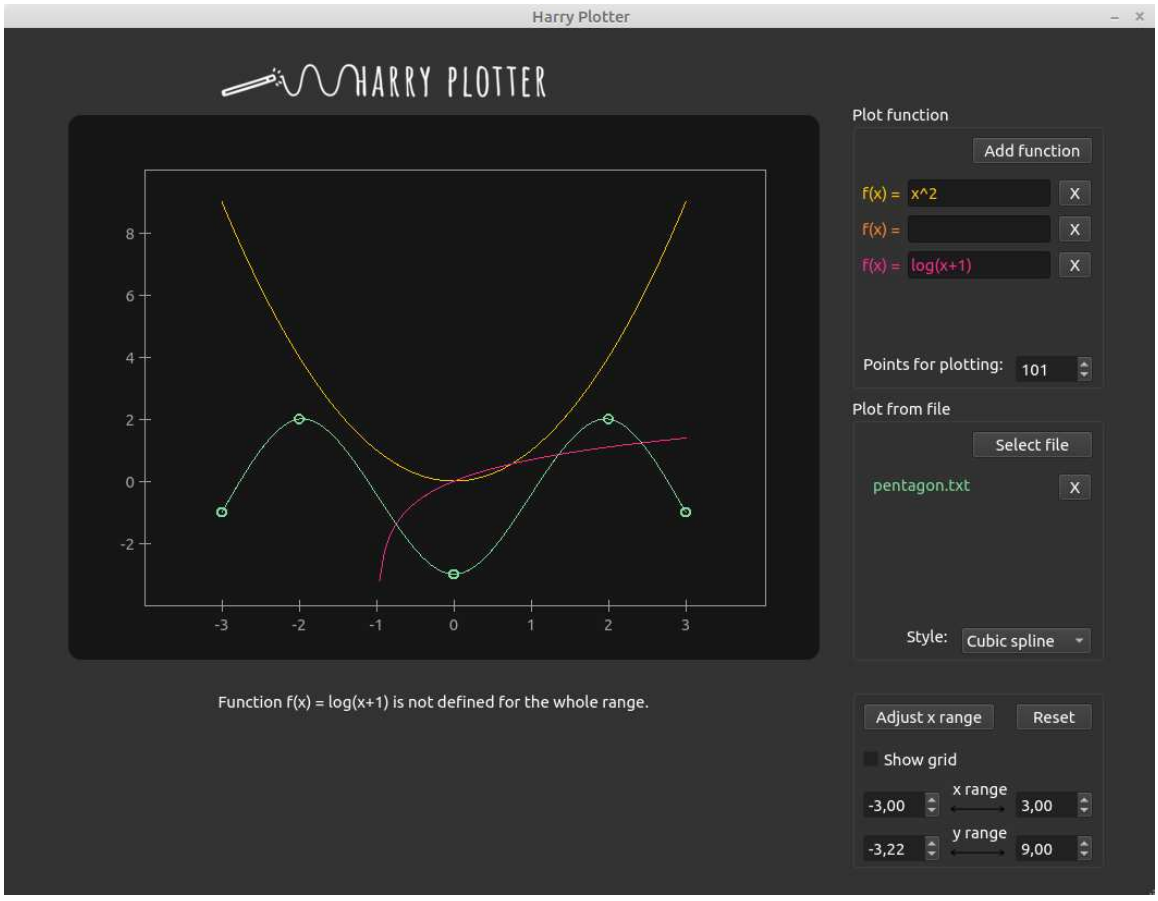
Postoji niz online aplikacija za crtanje grafova funkcija. Neke od njih su: ”FooPlot”, ”Graphing Calculator”, ”GraphSketch”, ”MathGrapher”. Zajednička im je mogućnost istovremenog crtanja grafova više funkcija i odabir x i y raspona.



Slika 2.3: GraphSketch - [9]

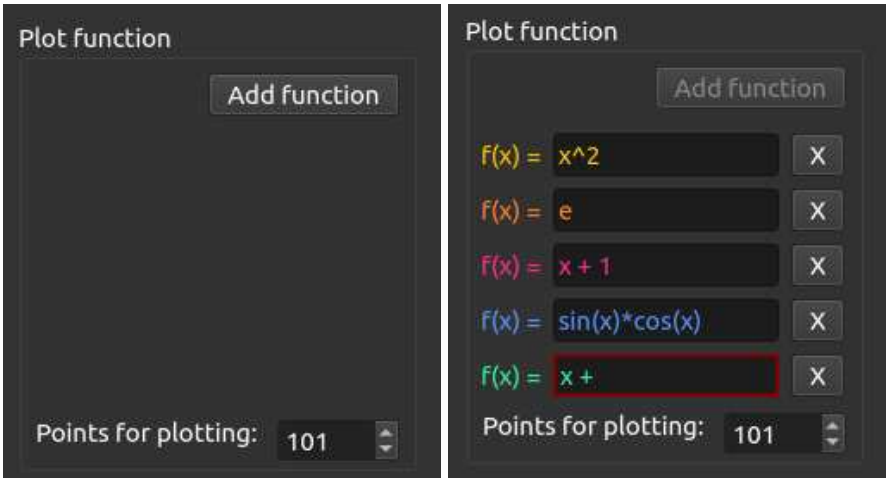
2.2 Funkcionalnost aplikacije

”Harry Plotter” aplikacija omogućuje crtanje grafova funkcija zadanih korisničkim unosom, kao i funkcija definiranih pomoću točaka iz datoteke. Omogućen je izbor x i y raspona, opcionalan prikaz koordinatne mreže, automatsko računanje raspona, odabir broja točaka za aproksimaciju funkcija, odabir stila iscrtavanja točaka iz datoteke te povratak na početne postavke (*reset*).



Slika 2.4: Aplikacija ”Harry Plotter”

Funkcije zadane korisničkim unosom



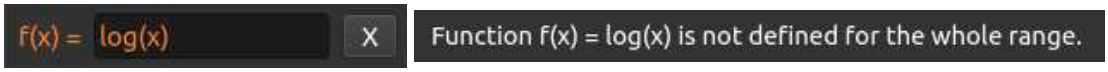
Slika 2.5: Unos funkcija

Nakon pritiska gumba ”Add function” na ekranu se pojavljuje mogućnost unosa funkcije s pripadajućim gumbom za brisanje. Na ovaj je način moguće istovremeno zadati maksimalno pet funkcija, pri čemu je svakoj dodijeljena različita boja. Po završetku unosa funkcije (kod promjene fokusa ili pritiska tipke Enter na tipkovnici) na ekranu se iscrtava graf funkcije u odgovarajućoj boji (zajedno s ranije zadanim funkcijama).

U slučaju neispravno zadane funkcije pojavljuje se crveni okvir i ispod grafa se ispisuje odgovarajuća poruka:

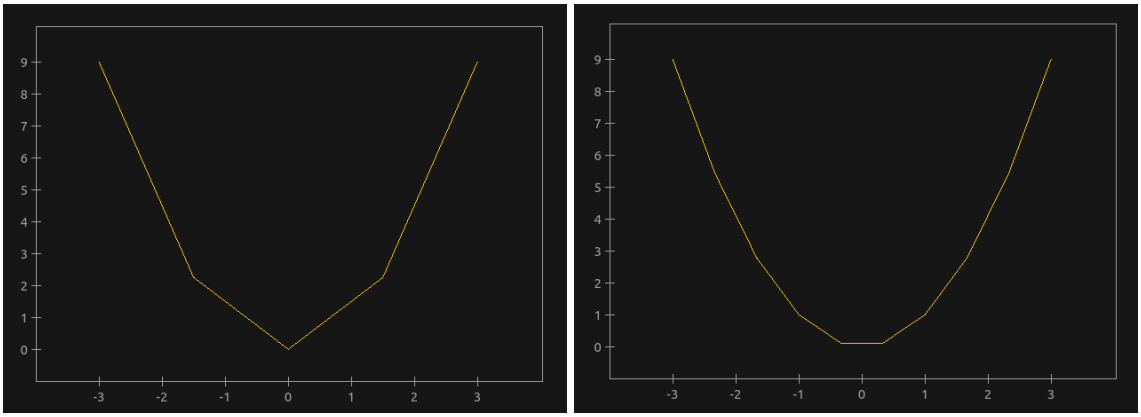
Slika 2.6: Neispravno zadana funkcija

Ukoliko funkcija nije definirana na cijelom prikazanom rasponu, ispod grafa se ispisuje odgovarajuća poruka:



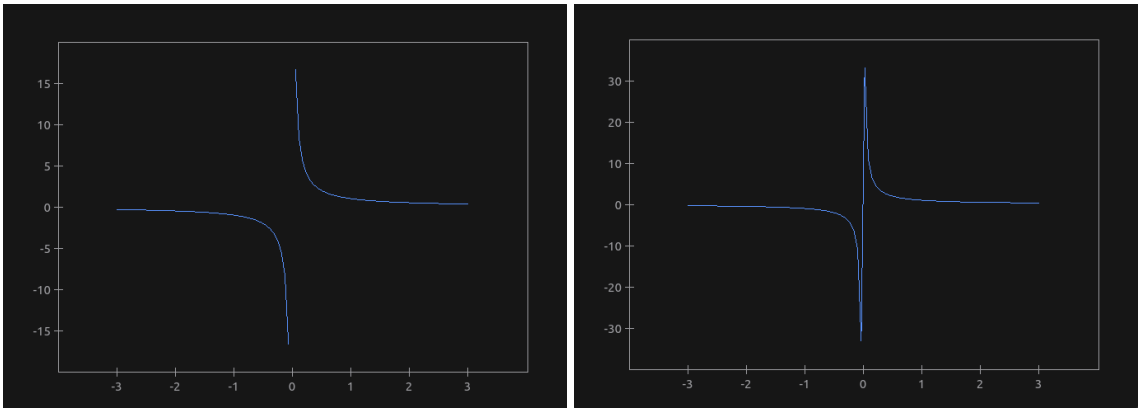
Slika 2.7: Funkcija koja nije definirana na prikazanom rasponu

U aplikaciji je omogućen odabir broja točaka pomoću kojih će se iscrtati graf funkcije. Naime, funkcija se aproksimira pomoću određenog broja ravnomjerno raspoređenih točaka (s obzirom na x os) povezanih ravnim linijama.



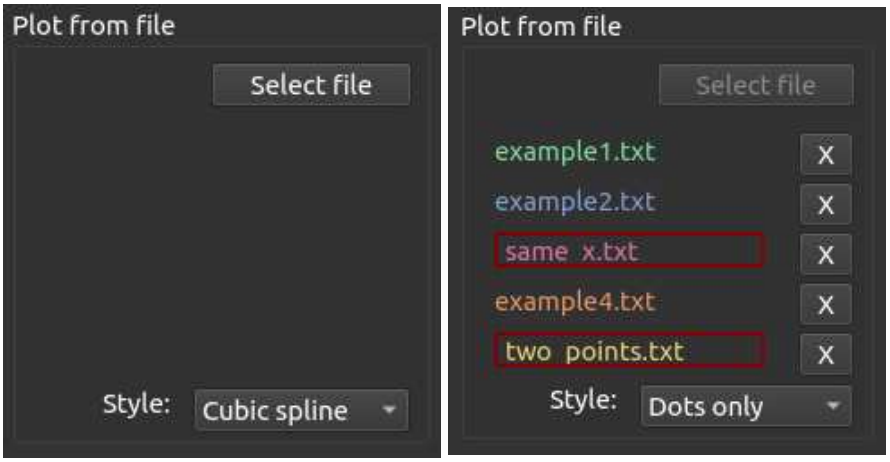
Slika 2.8: Graf funkcije x^2 iscrtan pomoću 5 točaka (lijevo) i 10 točaka (desno)

Zanimljiva je problematika funkcija s asimptotskim ponašanjem na zadanom rasponu. Primjerice, iscrtavanje grafa funkcije $1/x$ uvelike će ovisiti o broju točaka pomoću kojih je aproksimirana. Razlika s obzirom na izabrani broj točaka očituje se u automatski izračunatom y rasponu. Također, u ovisnosti o izabranom parnom ili neparnom broju točaka, vrijednost funkcije će se potencijalno pokušati izračunati za točku u kojoj funkcija nije definirana.
Na primjer, na zadanom rasponu od -3 do 3, kod izabranog neparnog broja točaka za aproksimaciju, nula će se nalaziti među točkama za koje se računa vrijednost funkcije. Kako funkcija $1/x$ nije definirana u nuli, prepoznaje se "problem" i asimptote se ne povezuju ravnom linijom. Kod izabranog parnog broja točaka ne dolazi do računanja vrijednosti funkcije u nuli te se asimptote povezuju ravnom linijom kao da se radi o "običnoj" funkciji. Ovakvo je rješenje napravljeno po uzoru na QCustomPlot i Matplotlib.



Slika 2.9: Graf funkcije $1/x$ iscrtan pomoću 101 točke (lijevo) i 100 točaka (desno)

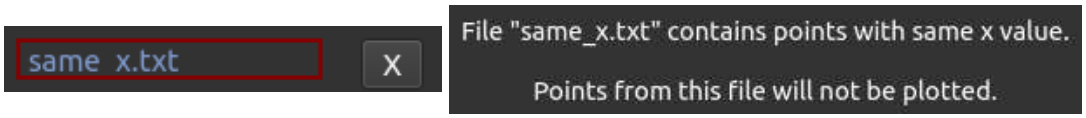
Funkcije zadane točkama iz datoteke



Slika 2.10: Odabir datoteka

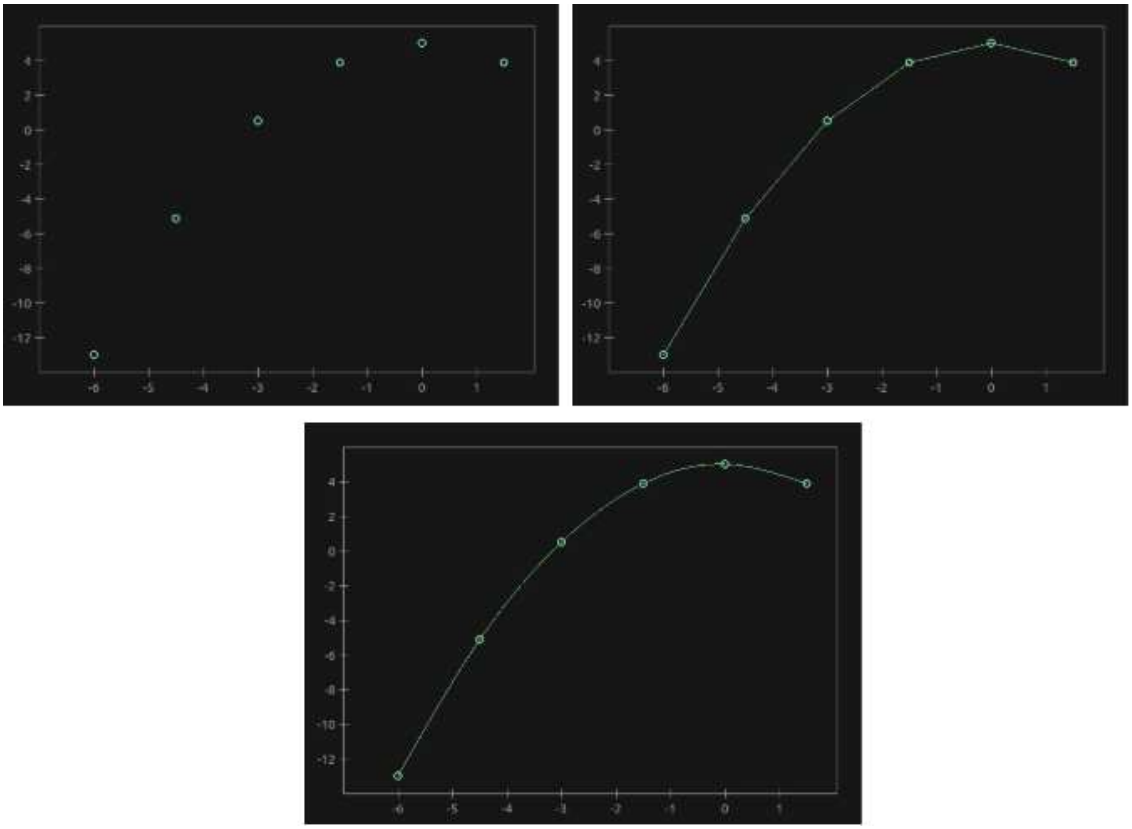
Nakon pritiska gumba ”Select file” otvara se dijaloški okvir pomoću kojega je moguće izabrati datoteku. Nakon izbora datoteke, na ekranu se pojavljuje njeno ime zajedno s pripadajućim gumbom za brisanje. Automatski se iscrtavaju točke iz datoteke u odgovarajućoj boji i zadanom stilu.

Ako datoteka sadrži točke s istom x koordinatom ili manje od tri ispravno zadane točke, oko imena datoteke pojavljuje se crveni okvir i ispod grafa se ispisuje odgovarajuće upozorenje.



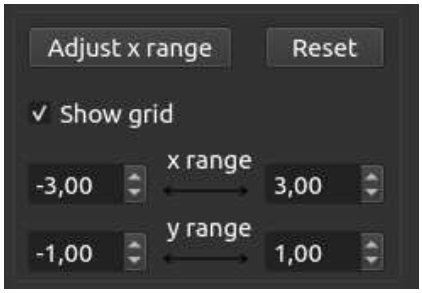
Slika 2.11: Datoteka koja sadrži točke s istim x vrijednostima

Omogućen je odabir stila prikaza točaka iz datoteke. Točke se mogu prikazati onako kako su zadane u datoteci (”Dots only”), ali ponuđene su i opcije linearne interpolacije (”Linear interpolation”) i kubičnog splajna (”Cubic spline”).



Slika 2.12: Stilovi prikaza točaka iz datoteke

Zajedničke funkcionalnosti

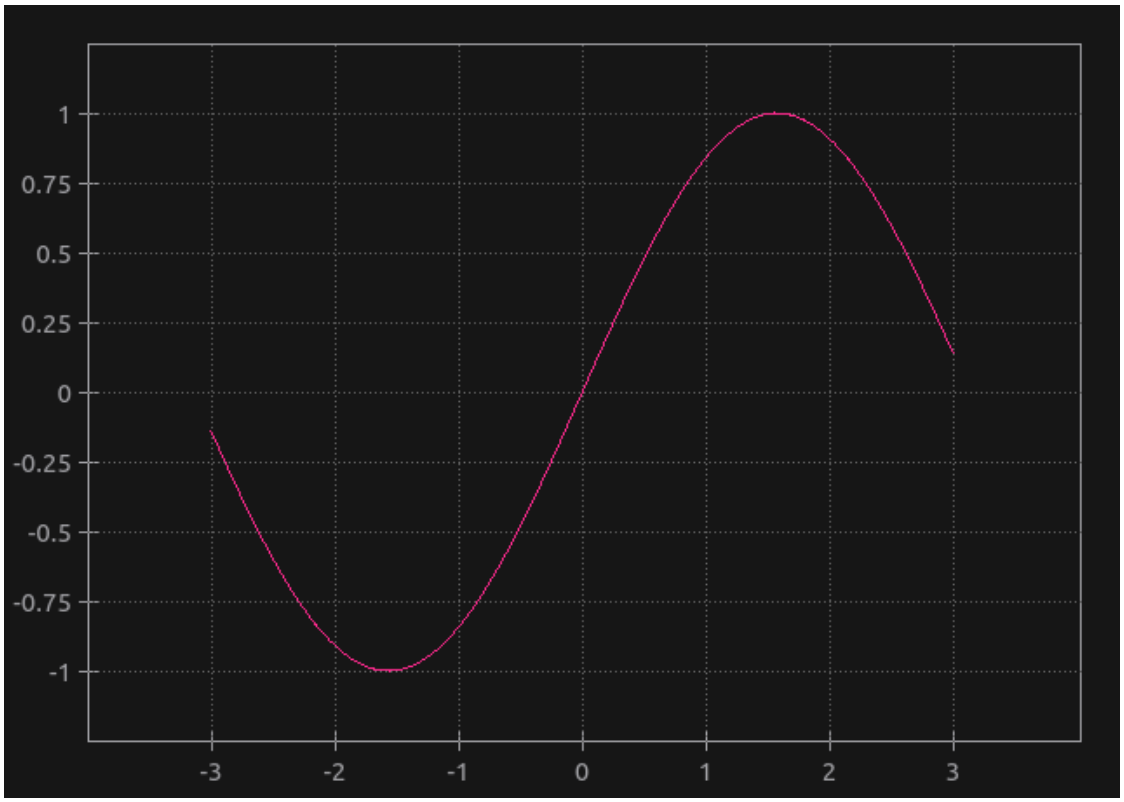


Slika 2.13: Dodatne opcije

U aplikaciji je omogućeno mijenjanje x i y raspona. Prilikom promjene raspona, grafovi se iscrtavaju s novim koordinatnim osima, prema unesenim vrijednostima. Napomenimo da se inače prilikom iscrtavanja grafova, primjerice nakon unosa funkcije, y raspon računa automatski te se ažuriraju njegove vrijednosti prikazane u dodatnim opcijama. Kod x raspona se u obzir uzimaju najveće (odnosno najmanje) vrijednosti x raspona prikazane u opcijama i vrijednosti x koordinata iz datoteka.

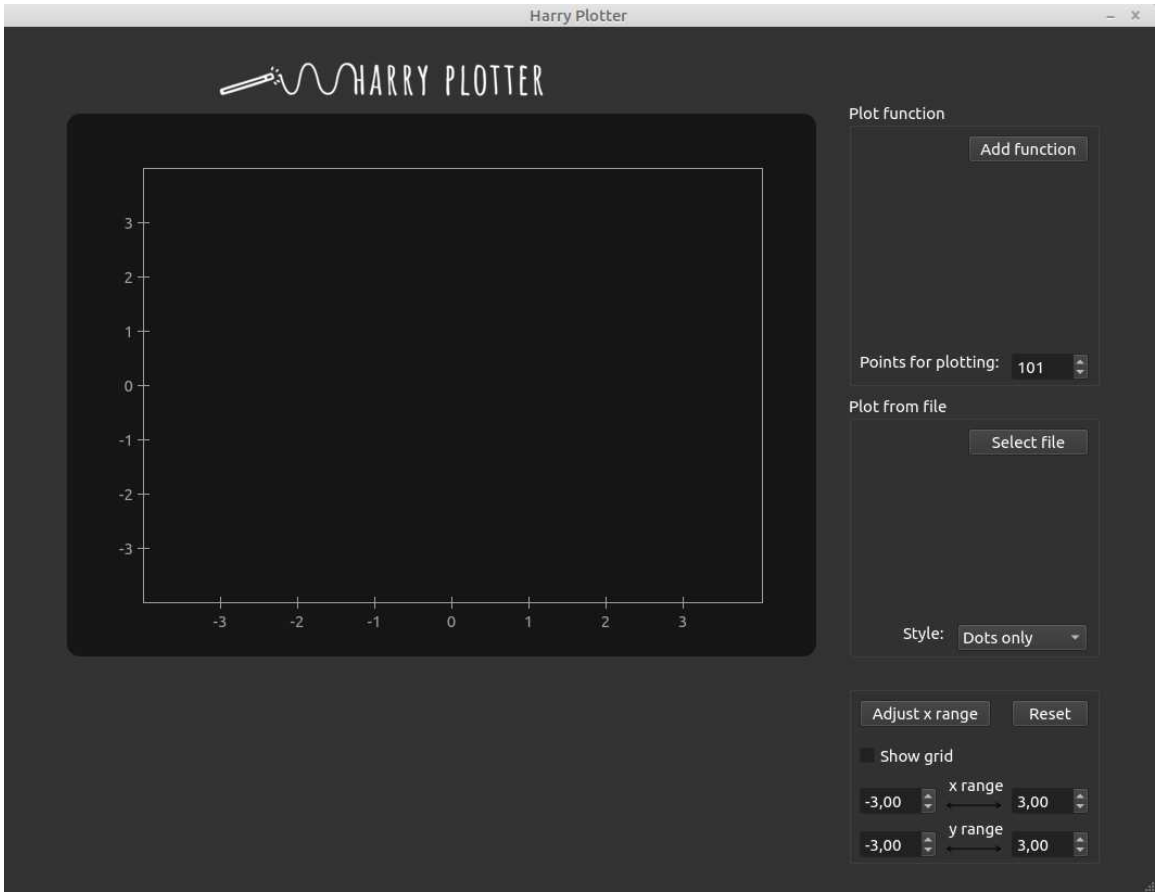
Pritiskom gumba "Adjust x range", x raspon računa se isključivo prema koordinatama točaka iz datoteka, a y raspon računa se automatski (prema koordinatama točaka iz datoteka i vrijednostima funkcija na izračunatom x rasponu).

Omogućen je opcionalan prikaz koordinatne mreže.



Slika 2.14: Graf funkcije s koordinatnom mrežom

Nakon pritiska gumba "Reset" s ekrana se uklanjaju svi grafovi, upozorenja, te zadane funkcije i datoteke. Sve se postavke postavljaju na početne.



Slika 2.15: Početne postavke

Prilikom prelaska pokazivačem miša preko grafa funkcije, iznad figure se u odgovarajućoj boji ispisuje o kojoj funkciji se radi. Ako je kao stil prikaza točaka iz datoteke izabrana linearna interpolacija ili kubični splajn, ispisuje se ime odgovarajuće datoteke.

Prilikom prelaska pokazivačem miša preko točke, ispisuju se koordinate točke.

Slika 2.16: Prikaz imena datoteke prilikom prelaska pokazivačem miša

2.3 Implementacija

Pri izradi aplikacije, korišteni su Qt-ovi moduli Qt Core, Qt GUI i Qt Widgets.
Za elemente grafičkog sučelja korišteni su widgeti sljedećih klasa naslijeđenih iz `QWidget`:

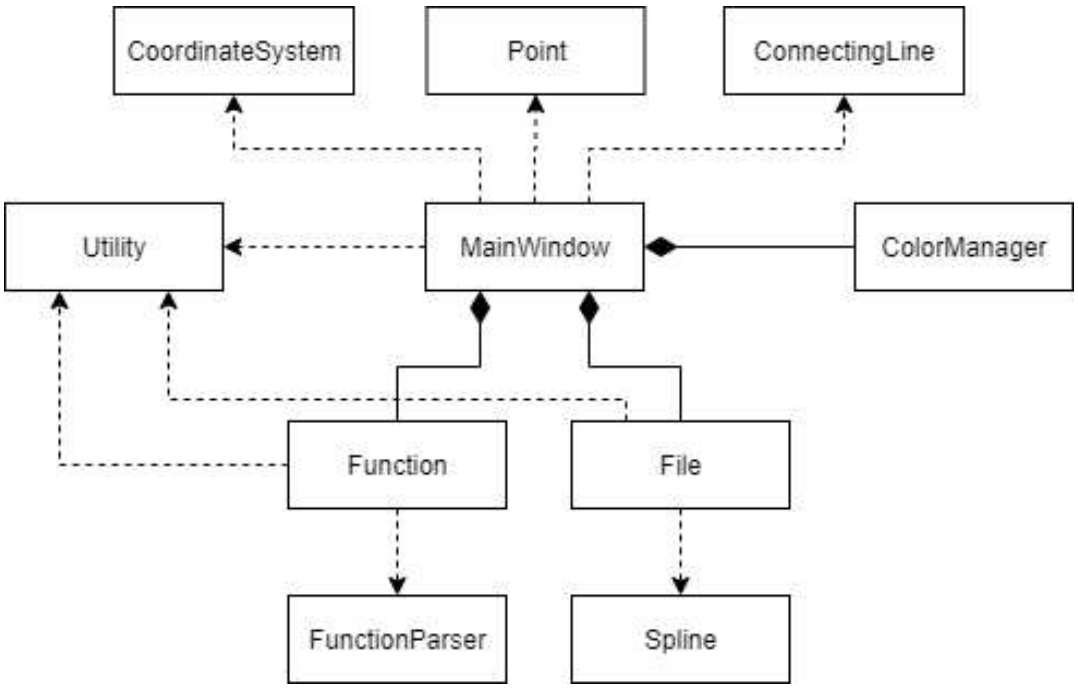
- `QGraphicsView` - platno za crtanje grafova
- `QPushButton` - gumbi za dodavanje funkcija, odabir datoteka, brisanje itd.
- `QSpinBox` - odabir broja točaka za aproksimiranje funkcije
- `QLabel` - labele s imenima odabranih datoteka
- `QCheckBox` - opcionalni izbor iscrtavanja grida
- `QDoubleSpinBox` - izbor x i y raspona
- `QComboBox` - izbor stila iscrtavanja točaka iz datoteke
- `QGroupBox` - grupiranje funkcija zadanih korisničkim unosom, funkcija zadanih točkama iz datoteke i zajedničkih funkcionalnosti

Također, korištene su i Qt-ove klase odgovorne za izgled grafičkog sučelja koje ne nasljeđuju `QWidget`: `QVBoxLayout` - za vertikalno programatsko poredavanje funkcija i datoteka, te `QSpacerItem` - okomiti *spacer* koji ”gura” dodane funkcije i datoteke na vrh, prisiljavajući ih da zauzmu najmanji mogući prostor.

Za dodavanje funkcija, točaka, koordinatnog sustava i mreže na scenu (i posljedičnog prikaza na ekranu pomoću `QGraphicsView` objekta) korišteni su objekti klase `QGraphicsItem` i iz nje naslijeđene klase `QGraphicsItemGroup`.

U aplikaciji su izrađene klase: **MainWindow**, **Function**, **File**, **CoordinateSystem**, **Point**, **ConnectingLine**, **ColorManager**, **Utility**.

Uz navedene, u aplikaciji se koriste klase **FunctionParser** i **Spline** za parsiranje funkcija i računanje vrijednosti kubičnog splajna. Klasa FunctionParser dio je ”fparser” biblioteke dostupne pod LGPL licencom [13]. Klasa Spline implementirana je u *open-source* ”spline.h” datoteci zaglavlja te dolazi s GPLv2 licencom [12].



Slika 2.17: Struktura aplikacije

MainWindow klasa

U MainWindow klasi implementirani su utori unaprijed definiranih signala koje emitiraju korišteni widgeti:

Isječak koda 2.1: Utori unaprijed definiranih signala

```
1     void on_selectFile_clicked();
2     void on_gridBox_stateChanged(int);
3     void on_addFunction_clicked();
4     void on_resetButton_clicked();
5     void on_rangeButton_clicked();
6     void on_styleBox_currentIndexChanged(int);
7     void on_xFrom_valueChanged(double);
8     void on_xTo_valueChanged(double);
9     void on_yFrom_valueChanged(double);
10    void on_yTo_valueChanged(double);
11    void on_pointsNumber_valueChanged(int);
```

Isječak koda 2.2: Utor koji se poziva nakon pritiska gumba "Add function"

```
void MainWindow::on_addFunction_clicked()
{
    Function* function = new Function(colorManager.getFunctionColor());
    functions.append(function);
    ui->functionLayout->addWidget(function);

    connect(function, &Function::deleteFunctionSignal, this,
            &MainWindow::deleteFunction);
    connect(function, &Function::plotSignal, this, &MainWindow::plot);

    if(int(functions.size()) == 5)
        ui->addFunction->setEnabled(false);
}
```

Također, implementirana su i tri utora za signale definirane u Function i File klasama:

Isječak koda 2.3: Utori signala iz klasa Function i File

```
    void deleteFunction(Function *function);
    void deleteFile(File *file);
    void plot(bool automatic);
```

Ostale funkcije članice MainWindow klase su:

Isječak koda 2.4: Funkcije članice MainWindow klase

```
    QPointF transform(QPointF point);
    void createLines();
    void createPoints();
    void yBoundaries(double* yStart, double* yEnd, double xStart,
                     double xEnd);
    bool pointInSystem(QPointF point);
```

Funkcija `plot()` središnja je funkcija u aplikaciji. Služi za iscrtavanje grafova i poziva se pri unosu funkcije, odabiru datoteke, brisanju funkcije ili datoteke, promjeni stila iscrtavanja točaka, promjeni broja točaka za aproksimaciju funkcije, izboru x i y raspona, odabiru prikaza koordinatne mreže te pritisku gumba za automatsko računanje raspona i "Reset" gumba.

Argument `automatic` služi kako bi se znalo treba li se prikazani raspon izračunati automatski ili se uzimaju vrijednosti iz `QDoubleSpinBox` objekata.

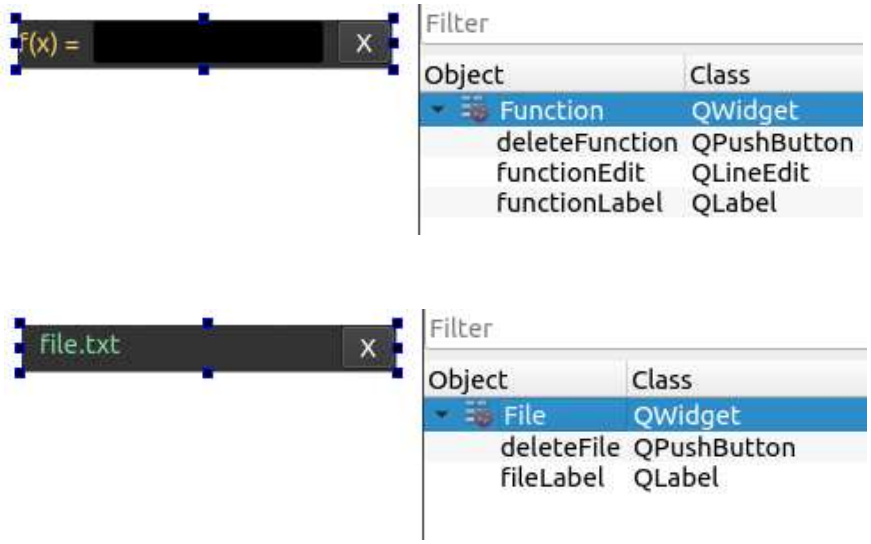
Isječak koda 2.5: Funkcija plot()

```
void MainWindow::plot(bool automatic)
{
    scene->clear();
    ui->warningLabel->setText("");
    QString warningText = "";
    double xStart, xEnd, yStart, yEnd;
    xStart = double(ui->xFrom->value());
    xEnd = double(ui->xTo->value());
    if(xStart >= xEnd)
    {
        ui->warningLabel->setText("Incorrect x range!");
        return;
    }
    if(automatic)
        //pronalazak novih xStart i xEnd na temelju x koordinata iz
         //datoteka i trenutnih xStart i xEnd vrijednosti
    if(!functions.empty())
        for(auto i = functions.begin(); i != functions.end(); ++i)
            warningText +=
            (*i)->generatePoints(xStart, xEnd, ui->pointsNumber->value());

    if(ui->styleBox->currentIndex() == 2)
        if(!files.empty())
            for(auto i = files.begin(); i != files.end(); ++i)
                (*i)->generateSplinePoints(xStart, xEnd);

    ui->warningLabel->setText(warningText);
    if(automatic)
    {
        yBoundaries(&yStart, &yEnd, xStart, xEnd);
        //postavljanje novih vrijednosti raspona u x i y SpinBox
    }
    else
    {
        yStart = ui->yFrom->value();
        yEnd = ui->yTo->value();
    }
    //prilagodba xStart, xEnd, yStart i yEnd kako bi se izbjeglo
     //crtanje uz sam rub koordinatnog sustava
    coordinateSystem* system = new coordinateSystem(scene);
    auto[system_Info,grid] = system->initialize(scene,xStart,xEnd,yStart,yEnd);
    createPoints();
    createLines();
    if(!ui->gridBox->isChecked())
       grid->hide();
}
```

Klase Function i File

U klasama Function i File nalaze se koordinate točaka potrebne za iscrtavanje grafa. Svaka od najviše pet istovremeno zadanih funkcija instanca je Function klase i svaka odabrana datoteka instanca je File klase. Svaka funkcija, kao i svaka datoteka, ima svoju boju i ime.

Klase Function i File imaju posebne datoteke za grafičko sučelje (odvojene su od `mainwindow.ui` datoteke). Qt Creator pruža automatski alat za generiranje klase i pridružene *forme* - pri dodavanju klase potrebno je izabrati "Qt Designer Form Class" opciju.



Slika 2.18: Forme function.ui i file.ui

Obje klase imaju utore za pritisak gumba za brisanje u kojima se emitira signal spojen s utorom MainWindow klase. Brisanje je implementirano u MainWindow klasi.

Isječak koda 2.6: Signali i utori klase File

```
1 signals:
2     void deleteFileSignal(File *file);
3 private slots:
4     void on_deleteFile_clicked();
```

Klasa Function ima dodatni signal i utore, potrebne kako bi aplikacija reagirala na promjenu unosa funkcije.

Isječak koda 2.7: Signali i utori klase Function

```
signals:
    void deleteFunctionSignal(Function *function);
    void plotSignal(bool automatic);

private slots:
    void on_functionEdit_textEdited(const QString &arg1);
    void on_functionEdit_editingFinished();
    void on_deleteFunction_clicked();
```

Obje klase imaju funkcije članice za generiranje točaka.
Kod funkcija, točke se generiraju na temelju x raspona i zadanog broja točaka za aproksimaciju.

Isječak koda 2.8: Definicija funkcije za generiranje točaka u klasi Function

```
    QString generatePoints(double xStart, double xEnd, int numberPoints);
```

Kod datoteka, točke se generiraju na temelju tekstualnog *stream*-a iz datoteke.
Također, klasa File ima i funkciju članicu za generiranje točaka pomoću kojih se iscrtava kubični splajn.

Isječak koda 2.9: Definicija funkcija za generiranje točaka u klasi File

```
    QString generatePoints(QTextStream* in);
    void generateSplinePoints(double xStart, double xEnd);
```

Klase CoordinateSystem, Point i ConnectingLine

Klase CoordinateSystem, Point i ConnectingLine služe za iscrtavanje linija, točaka i koordinatnog sustava na platno.

Nasljeđuju klase izvedene iz `QGraphicsItem` klase: Point nasljeđuje `QGraphicsEllipseItem`, ConnectingLine `QGraphicsLineItem`, a CoordinateSystem `QGraphicsItemGroup` klasu.

U klasama Point i ConnectingLine implementirani su Qt-ovi događaji (*events*) za ulazak i izlazak pokazivača miša iz područja grafičke komponente. Time je omogućen ispis točke, funkcije ili datoteke prilikom prelaska pokazivačem miša.

Isječak koda 2.10: Funkcije kojima je omogućena reakcija na događaj prelaska mišem

```
1    void hoverEnterEvent(QGraphicsSceneHoverEvent*);
2    void hoverLeaveEvent(QGraphicsSceneHoverEvent*);
```

Klasa CoordinateSystem ima funkciju članicu `initialize()` kojom se, na temelju x i y raspona, određuje način iscrtavanja koordinatnog sustava.

Klase ColorManager i Utility

ColorManager i Utility pomoćne su klase.

Klasa ColorManager služi za određivanje boja funkcija i datoteka.

Isječak koda 2.11: Funkcije članice klase ColorManager

```
1    QString getFileColor();
2    QString getFunctionColor();
3    void releaseFileColor(QString color);
4    void releaseFunctionColor(QString color);
5    void reset();
```

U Utility klasi implementirane su statičke pomoćne funkcije koje koriste klase MainWindow, Function i File:

Isječak koda 2.12: Funkcije članice klase Utility

```
1    static bool compareDoubles(double a, double b);
2    static QVector<double> linspace(double start, double end, int num);
3    static bool sameX(QVector<QPointF> points);
4    static bool isUndefined(QPointF point);
```

Bibliografija

[1] L. Eng, *Hands-On GUI Programming with C++ and Qt5*. Packt Publishing Ltd., 2018.

[2] G. Lazar and R. Penea, *Mastering Qt5*. Packt Publishing Ltd., 2016.

[3] N. Sherriff, *Learn Qt5*. Packt Publishing Ltd., 2018.

[4] A. Ezust and P. Ezust, *An Introduction to Design Patterns in C++ with Qt*. Prentice Hall, 2012.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Elements of Reusable Object-Oriented Software*. Addison Wesley, 2002.

[6] "Qt 5.14." `https://doc.qt.io/qt-5/`, preuzeto: siječanj, 2020.

[7] M. Jurak, "Principi objektnog programiranja i oblikovni obrasci." `https://web.math.pmf.unizg.hr/nastava/opepp/Slides/Predavanja/html-noslides/slides-6-1.html`, preuzeto: siječanj, 2020.

[8] S. Collins, "A deeper look at signals and slots." `http://elpauer.org/stuff/a_deeper_look_at_signals_and_slots.pdf`, preuzeto: siječanj, 2020.

[9] "GraphSketch.com." `https://graphsketch.com/`, preuzeto: siječanj, 2020.

[10] "QCustomPlot." `https://www.qcustomplot.com/`, preuzeto: studeni, 2019.

[11] "Matplotlib." `https://matplotlib.org/`, preuzeto: studeni, 2019.

[12] "Cubic Spline interpolation in C++." `https://kluge.in-chemnitz.de/opensource/spline/`, preuzeto: prosinac, 2019.

[13] "Function Parser for C++ v4.5.2." `http://warp.povusers.org/FunctionParser/`, preuzeto: prosinac, 2019.

Sažetak

U radu je predstavljena Qt5 biblioteka, razvojni paket pogodan za izradu aplikacija s grafičkim korisničkim sučeljem. Također, konstruirana je aplikacija ”Harry Plotter” za vizualizaciju podataka kojom je omogućeno crtanje grafova funkcija jedne varijable.
U prvom dijelu rada opisuju se ključni dijelovi Qt5 biblioteke, poput meta-object sustava, mehanizma signala i utora, arhitekture modela i pogleda, te ih se povezuje s oblikovnim obrascima Composite i Observer. Slijedi opis dijela Qt biblioteke potrebnog za izradu aplikacije: Qt Widgets aplikacija i softverski okvir Graphics View. Također, opisan je Qt Charts modul za crtanje različitih vrsta grafikona. U završnom dijelu rada predstavljena je sama aplikacija. Predstavljene su biblioteke kojima je inspirirana, QCustomPlot i Matplotlib. Detaljno su opisane funkcionalnost aplikacije i njena implementacija.

Summary

This master thesis introduces the Qt5 library, a development package suitable for creating applications with a graphical user interface. ”Harry Plotter”, a data visualization application, has also been developed that enables the functions of a single variable to be plotted. At the beginning of the thesis the essential parts of the Qt5 library - Meta-Object system, the signals and slots mechanism, model-view architecture - are described and affiliated with the Composite and Observer design patterns. The second part describes the components of the Qt library required to develop the application: the Qt Widgets application and the Graphics View framework. A Qt Charts module for drawing various types of charts has also been introduced. Finally, the application ”Harry Plotter” itself is presented. Other solutions for plotting are described, such as QCustomPlot and Matplotlib libraries. The functionality of the application and its implementation are described.

Životopis

Paula Vujasić rođena je 4. siječnja 1996. u Zagrebu. Školovanje započinje u Osnovnoj školi Šestine. Potom upisuje V. gimnaziju u Zagrebu i završava je 2014. godine. Iste godine upisuje Preddiplomski sveučilišni studij Matematika na Prirodoslovno - matematičkom fakultetu.

Nakon završenog preddiplomskog studija, 2017. upisuje Diplomski sveučilišni studij Računarstvo i matematika na istom fakultetu. Tijekom srednjoškolskog i fakultetskog obrazovanja natjecateljski se bavi sportskim plesom te s vremenom ulazi u hrvatsku i slovensku reprezentaciju u latinoameričkim plesovima.