Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i Data	Vårsemesteret, 2021 Åpen
Forfattere: Daniel Bjørge Anders Thengs Kristensen	Daniel Bjørge ..... (signatur forfatter) Anders Thengs Kristensen ..... (signatur forfatter)
Fagansvarlig: Veiledere: Jarle Urdal og Trygve Eftestøl	
Tittel på bacheloroppgaven: Timeline-representation: Et verktøy for visualisering av tidslinjer i Python	
Studiepoeng: 15	
Emneord: Python, tidslinjer	Sidetall: 23 Stavanger, 15/05/2021

Sammendrag

Gruppen fikk tildelt en oppgave hvor formålet var å utvikle et program som kunne visualisere medisinsk data ved hjelp av tidslinjer. Programmet skulle lages i Python, men gruppen sto fritt til å velge hvilke moduler som skulle brukes. Denne rapporten beskriver hvilke moduler som ble valgt, og hvorfor de ble valgt.

Oppgaven ble løst ved å lage et grunnlag for brukergrensesnittet i QtDesigner. Deretter ble det generert Pythonkode for dette grunnlaget, som ble bygget videre på for å oppnå ønsket funksjonalitet. Til slutt ble programmet kompilert til filer kjørbare på Windows, macOS og Linux.

Innholdsliste

1 Innledning

1.1 Oppgavebeskrivelse

Oppgaven går ut på å utvikle et grafisk brukergrensesnitt for å visualisere medisinsk data ved bruk av tidslinjer. Brukergrensesnittet skal lages i Python og importere tidslinjer fra et eksisterende rammeverk i csv-filer. Tidslinjene skal kunne beskrive prehospital hjertestans og gjenoppliving av nyfødte. En mulig utvidelse av programmet er å kunne visualisere andre tidslinjer, for eksempel fra Nordsjørittet. Programmet er kalt Timeline-representation.

1.2 Motivasjon

Tidslinjer er en metode som kan brukes for å representere komplekse data på en illustrativ måte. Metoden er også nyttig for å analysere og tolke hendelsesforløp.

Ved institutt for data- og elektroteknologi er tidslinjer allerede i bruk for å beskrive og analysere

1) behandlingen og hjerterate ved prehospital hjertestans, og
2) behandlingen som gis under gjenopplivning av nyfødte.

1.3 Mål for oppgaven

Basert på oppgavebeskrivelsen og motivasjonen ble det formulert de følgende målene for oppgaven:

Primærmålet er

- å lage et grafisk brukergrensesnitt for å visualisere tidslinjer av prehospital hjertestans og gjenoppliving av nyfødte.

Etter dette er sekundærmålene

1) å utvide Timeline-representation til å kunne generere tidslinjer for andre medisinske data,
2) å lage en selvstendig kjørbar versjon av Timeline-representation for lettere distribusjon,
3) å tillate brukere å tilpasse brukergrensesnittet uten at det kommer på bekostning av brukervennlighet.

Sekundærmål 3 er delt i tre delmål for å gjøre det konkret. Disse delmålene er

1) å tillate brukere å utnytte fanefunksjonalitet,
2) å tillate brukere å zoome inn og ut på tidslinjene,
3) å tillate brukere å tilpasse fargene brukt i tidslinjene.

Alle delmålene til sekundærmål 3 skal fortsatt nås uten at det kommer på bekostning av brukervennlighet.

2 Teknologivalg

Utviklingen av Timeline-representation er gjort ved bruk av programmeringspråket Python [Python Software Foundation(2020a)]. En av fordelene med Python er tilgjengeligheten av første- og tredjepartimoduler for tilnærmet hva som helst, enten det er et enormt rammeverk med utallige muligheter, eller en enkelt funksjon som sparer mye tid og arbeid. I de følgende underseksjonene introduseres de ulike Pythonmoduler som brukes i Timeline-representation.

2.1 PyQt

Målet med oppgaven er å visualisere tidslinjer ved hjelp av et brukervennlig brukergrensesnitt. Python har en rekke moduler for å lage brukergrensesnitt, både tredjepartimoduler som PyQt [Riverbank Computing Limited(2020)] og førstepartimodulen tkinter [Python Software Foundation(2020c)]. Modulen valgt for brukergrensesnittet er PyQt.

En stor fordel med PyQt i forhold til andre moduler er at den har offisiell designerprogramvare, QtDesigner. QtDesigner lar en lage et oppsett for et brukergrensesnitt, og vil generere koden for den grafiske delen av dette brukergrensesnittet. QtDesigner gjør på denne måten mye av det tunge arbeidet, og gir et godt grunnlag å bygge videre på.

2.2 Matplotlib

Matplotlib [Hunter(2007)] er en Pythonmodul for visualisering av data. Delen av Matplotlib brukt i programmet er PyPlot, som er designet for å minne om Matlab. Matplotlib har også innebygget integrasjonen med PyQt.

2.3 Pandas

Når det kommer til større datamengder, hvor hvert datapunkt har flere forskjellige attributter, egner det seg ikke med primitive datatyper som lister. Derfor brukes Pandas [pandas development team(2021)],[Wes McKinney(2010)]. Pandas er primært en dataprossesseringsmodul, men i Timeline-representation utnyttes denne funksjonaliteten ikke.

Det tas istedet nytte av datatypen Dataframe. Dataframe-objekter er veldig gunstige for arbeidet, ettersom de tilatter å ha all informasjonen om tidslinjene i en dedikert datatype, som da alle nødvendige operasjoner kan utføres på. Pandas har også innebygd integrering med Matplotlib.

2.4 Itertools

Itertools [Python Software Foundation(2020b)] er en Pythonmodul som implementerer en rekke forskjellige måter å iterere på. Den er valgt for funksjonen *cycle*, som lager objekter med en rekke elementer som kan iterere over i uendelighet.

2.5 QtAwesome

QtAwesome [Spyder Project Contributors(2020)] er en Pythonmodul som gjør det mulig å bruke ikonskrifttyper som FontAwesome eller Elusive Icons i brukergrensesnitt laget med PyQt. Den kommer også med disse skrifttypene innebygd, slik at det ikke er nødvendig å inkludere separate filer for ikoner.

2.6 PyInstaller

Python er et interpretert programmeringsspråk. Det betyr at en kan kjøre et program skrevet i Python direkte fra koden, i motsetting til i kompilerte språk hvor koden må kompileres til et ferdig program før den kan kjøres. Dette gjør det lettere å dele Pythonkode og kjøre den på andre maskiner, men det krever at mottakeren har et *virtual environment* med alle nødvendige moduler installert. For en stor del brukere er dette mer enn kan forventes, så kompilerte programmer vil være lettere å dele.

PyInstaller[Goebel et al.(2021)Goebel, Bajo, Vierra, Cortesi, and Zibricky] gjør det mulig å kompilere Pythonkode, slik at programmer kan kjøres på andre maskiner uten å måtte installere Python og eventuelle moduler. PyInstaller har støtte for Windows, macOS og Linux.

3 Filformat

Tidslinjene som skal visualiseres i Timeline-representation er tilgjengelige i kommaseparerte filer, csv-filer. Filene har tre komponenter, som vist i eksempelkode 1.

```
Timeline,v10,offset,20
Stimulation,Detected Activity,Ventilation,
-Inf,0,<UN>,
0,5,<H><H><H>,
5,7.05,<H><A><H>,
7.05,10,<S><A><H>,
10,25.15,<H><A><H>,
25.15,29.89,<H><A><H>,
29.89,35.28,<H><H><H>,
35.28,42.28,<S><H><V>,
42.28,43.25,<S><A><V>,
43.25,44.43,<H><A><H>,
44.43,48.48,<H><H><H>,
48.48,49.65,<H><A><H>,
49.65,53.25,<S><A><V>,
53.25,55.55,<H><A><H>,
55.55,58.25,<S><A><H>,
58.25,60.98,<H><A><H>,
60.98,Inf,<UN>,
```

Eksempelkode 1: Eksempel på formatet som brukes til å lagre tidslinjer i csv-filer

Den første linjen spesifiserer at filen er en tidslinje, og hvilken versjon av tidslinjeformatet den er. Den kan også spesifisere en *offset*-verdi, som sier hvor lang tid som forekom mellom fødsel og starten av analysen dekket av tidslinjen. I eksempelfilen er versjonen 10 og *offset*-verdien 20. Det betyr at det var 20 sekunder mellom fødsel og starten av analysen.

Linje 2 inneholder informasjon om hvilke aktiviteter tidslinjen dekker. I eksempelkode 1 er disse *Simulation*, *Detected Activity* og *Ventilation*. Det er ingen øvre grense på antall aktiviteter, og en nedre grense på en aktivitet. Det er heller ingen begrensninger på hvilke aktiviteter som inkluderes i en tidslinje. Dette gjør det mulig å bruke dem til å dekke hvilke som helst tidslinjer, som for eksempel fra Nordsjørittet, så lenge tidslinjene er lagret i det korrekte filformatet.

De øvrige linjene viser hvilke aktiviteter som gjøres i de ulike tidsrommene. Hver linje dekker et tidsintervall, hvor første verdi på linja er starten av intervallet og andre verdien er slutten av intervallet. Den tredje verdien sier hvilke aktiviteter som er aktive i det gitte intervallet, med hver aktivitet som en bokstav i vinkelparenteser, <> . Bokstaven H tilsier at aktiviteten ikke er

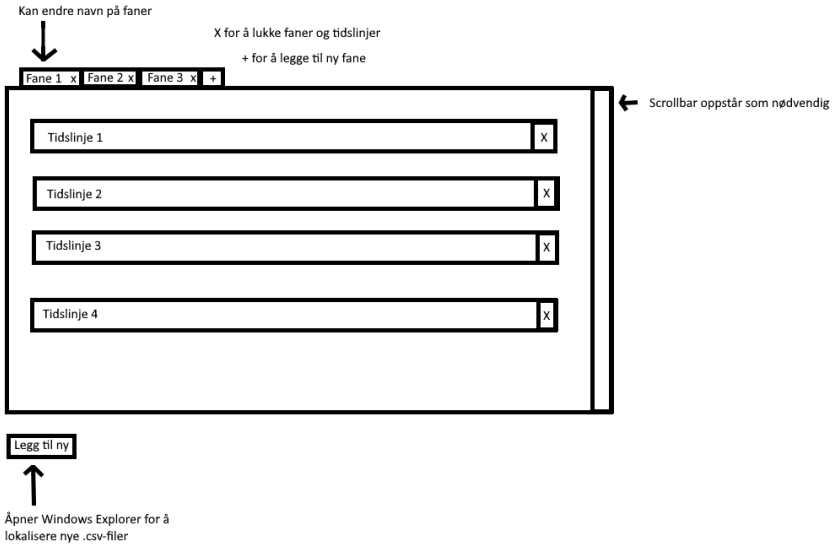
pågående, mens andre bokstaver tilsier at den er. I denne konteksten står H for *Hands off*. Det er vanlig å bruke første bokstaven i aktiviteten som bokstaven i vinkelparenteser, men det er ikke nødvendig.

Første og siste linje som omhandler tidslinjen selv går fra minus uendelig og til uendelig respektivt, og har <UN> som aktivitet. UN står her for *Undefined*. Dette markerer et tydelig skille for hvor tidslinjen starter og slutter. Disse brukes ikke i Timeline-representation, ettersom Matplotlib og Pandas ikke håndterer uendelig på en lett måte uten andre moduler.

4 Utvikling av brukergrensesnitt

I dette kapittelet presenteres de ulike stegene i utviklingen av Timeline-representation.

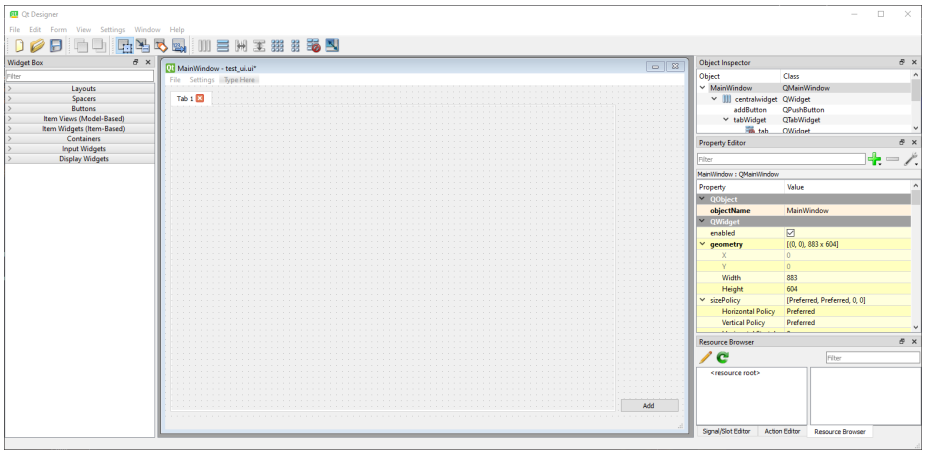
Figur 1 viser et utkast til brukergrensesnittet som ble tegnet før arbeidet ble påbegynt. Figuren er ment til å



Figur 1: Utkast for brukergrensesnitt

4.1 Grafisk oppsett ved hjelp av QTDesigner

Grunnlaget for brukergrensesnittet ble laget i QtDesigner. Ved å gjøre det i QtDesigner heller enn fra scratch ble det spart tid ved å samtidig visualisere hovedtrekkene i brukergrensesnitt og lage koden for det. I tillegg gjorde dette det mulig å bare måtte konsultere dokumentasjonen når nødvendig, mens de grunnleggende prinsippene i PyQt ble tilgjengeliggjort som ferdig kode. Figur 2 viser QtDesigner med grunnlaget til brukergrensesnittet til Timeline-representation.



Figur 2: Grunnlaget for brukergrensesnittet i QtDesigner

Alle brukergrensesnittobjekter i PyQt er basert på QWidget. *Widgeter* er grunnlegende i brukergrensesnittet. De mottar tastatur, mus og andre hendelser fra operativsystemet, og tegner seg selv på skjermen. Hvert *widget* er rektangulært og sortert i zrekkefølge. Zrekkefølge betyr at hvert *widget* ligger direkte fremfor foreldreobjektet. Et *widget* som ikke er satt i et foreldreobjekt er oftest kalt et vindu. *Widgeter* kan ikke ha større dimensjoner på skjermen enn foreldreobjektene. Se [Riverbank Computing Limited()] for mer informasjon om QWidget.

```
14 class Ui_MainWindow(object):
15     def setupUi(self, MainWindow):
16         MainWindow.setObjectName("MainWindow")
17         MainWindow.resize(895, 604)
18         self.centralwidget = QtWidgets.QWidget(MainWindow)
19         self.centralwidget.setAutoFillBackground(False)
20         self.centralwidget.setObjectName("centralwidget")
21         self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.
    centralwidget)
22         self.horizontalLayout_2.setObjectName("horizontalLayout_2")
```

Eksempelkode 2: Utdrag fra koden generert av QtDesigner

Eksempelkode 2 viser delen av koden fra QtDesigner som setter opp oppsettet til brukergrensesnittet. Standardstørrelsen på vinduet settes på linje 17. I eksempelet er denne satt til en vilkårlig verdi, men i det ferdige programmet er den satt til 1280 piksler bred og 720 piksler høy, som er oppløsningen 720p HD. Denne oppløsningen har lenge vært et minimum på datamaskiner, og anses som et trygt valg.

Linje 21 setter opp et horisontalt layout til å plassere elementene i brukergrensesnittet i. I det endelige programmet er dette endret til et QGridLayout, et rutenettlayout, ettersom det gir bedre kontroll over hvor elementer er plassert i brukergrensnittet.

4.2 Tidslinjer

Tidslinjene er håndtert ved hjelp av de følgende objektene: TimelineStorage, TimelineCanvas, ButtonAndLabel og TimelineEntry.

TimelineStorage er en subklasse av en liste og brukes til å lagre og håndtere alle tidslinjene etterhvert som de legges inn. Hver gang en bruker foretar en endring av elementer i en tidslinje, gjennom å zoome inn eller ut eller å endre fargene på en aktivitet er det TimelineStorage-objektet som forplanter endringene til de individuelle tidslinjene. Ved oppstart av programmet lages et TimelineStorage-objekt.

TimelineStorage-objektet generer et itertools *cycle*-objekt med farger fra matplotlib. Hver gang en ny aktivitet legges til hentes det neste elementet i *cycle*-objektet, og den fargen brukes for aktiviteten. Om *cycle*-objektet itereres fullstendig over vil det starte igjen fra første element, slik at det aldri vil gå tom for farger etterhvert som nye aktiviteter legges til. Eksempelkode 3 viser hvordan *cycle*-objektet genereres. Det første elementet i fargelisten blir satt til hvit, #FFFFFF, slik at fargen for ingen aktivitet blir hvit. Eksempelkode 4 viser hvordan *cycle*-objektet itereres over.

```
204         prop_cycle = plt.rcParams['axes.prop_cycle']
205         self.colors = [cycle(prop_cycle.by_key()['color']), ['#
    ffffff']]
```

Eksempelkode 3: Kode for å generere *cycle*-objekt med farger

```
177         while len(self.colors[1]) < len(self.dfs)+1:
178             self.colors[1].append(next(self.colors[0]))
```

Eksempelkode 4: Kode for å hente farger for tidslinjeaktiviteter

Grunnen til at TimelineEntry, TimelineCanvas og ButtonAndLabel er skilt inn i tre forskjellige objekter er for å lettere plassere dem i brukergrensesnittet. Eksempelkode 5 viser hvordan de er plassert. Koden er fra TimelineEntry og tL er et TimelineCanvas-objekt.

```
18          self.tL = tL
19          self.tL.setEntry(self)
20          self.setParent(parent)
21          layout = QHBoxLayout(self)
22          self.buttonwidget = ButtonAndLabel(self.tL.name, self.tL.
    info[1], self.tL.info[-1].rstrip())
23          layout.addWidget(self.buttonwidget, alignment=QtCore.Qt.
    AlignLeft)
24          layout.addWidget(tL, alignment=QtCore.Qt.AlignLeft)
25          self.buttonwidget.button.clicked.connect(self.deleteSelf)
26          if len(self.tL.dfs)*50 > 100:
27              self.setFixedHeight(len(self.tL.dfs)*50)
28          else:
29              self.setFixedHeight(100)
30          self.setFrameStyle(1)
```

Eksempelkode 5: Kode for å plassere tidslinjer i brukergrensesnittet

TimelineCanvas er objektet som er ansvarlige for å visualisere individuelle tidslinjer. TimelineCanvas er en subklasse av FigureCanvasQTAgg-klassen fra Matplotlib, som er laget for å kunne gjengi plot fra Matplotlib i PyQt. Når en tidslinje legges til lages et TimelineCanvas-objekt som leser csv-filen og lagrer informasjonen fra csv-filen i et Dataframe-objekt. Denne informasjonen plottes i et *horizontal bar plot* fra PyPlot. Om en bruker hovrer over plottet vil det vises nøyaktig aktivitet og tidspunkt ved posisjonen til musepekeren.

ButtonAndLabel er et objekt som viser navnet, versjonen og eventuell *offset*-verdi for tidslinjen det hører til. Det er en subklasse av QWidget. For å unngå at ButtonAndLabel skal ta for stor plass på skjermen er det satt en fast størrelse på 125 piksler bredde og 100 piksler høyde.

ButtonAndLabel inneholder knappen for å fjerne en tidslinje fra programmet, markert med ✖ fra FontAwesome og plassert under teksten. Ved trykk skjer to ting:

1) TimelineEntry-objektet setter forelderen til *None*, som fjerner den fra brukergrensesnittet.

2) TimelineStorage-objektet bruker listers innebygde *remove*-funksjon til å fjerne tidslinjen fra seg selv.

Ettersom Python bruker en *Garbage Collector* vil ikke objektene nødvendigvis bli slettet fra minne øyeblikkelig, men først når *Garbage Collectoren* sletter dem. *Garbage Collectoren* kjører av seg selv i bakgrunnen.

TimelineEntry er objektet som dekker en individuell tidslinje og er en subklasse av QFrame. QFrame er en subklasse av QWidget med muligheten for å ha en visuell ramme. Dette brukes for å sette en distinkt ramme rundt hver tidslinje.

4.3 Legge til tidslinjer

Knappen for å legge til tidslinjer, *Add*-knappen, er plassert nederst til høyre i brukergrensesnittet, og har en større skriftstørrelse enn andre knapper. Dette er gjort for å trekke en ny brukers oppmerksomhet mot den, ettersom den er den viktigste knappen for å kunne ta i bruk programmet. Dette følger også konvensjonen om at knapper for å få en bruker videre i et dialogvindu vanligvis er plassert nederst til høyre. Eksempelkode 6 viser hvordan *Add*-knappen lages og plasseres i brukergrensesnittet. Eksempelkode 7 viser hvordan trykk på *Add*-knappen bindes til funksjonen som legger en tidslinje til brukergrensesnittet, addTimeline.

```
37          self.addButton = QtWidgets.QPushButton(self.centralwidget)
38          self.addButton.setFixedWidth(200)
39          self.addButton.setObjectName("addButton")
40          self.addButton.setProperty("class", "styledButton")
41          self.grid.addWidget(self.addButton, 4, 4, alignment=QtCore.
        Qt.AlignHCenter)
```

Eksempelkode 6: Kode for å plassere *Add*-knappen i brukergrensesnittet

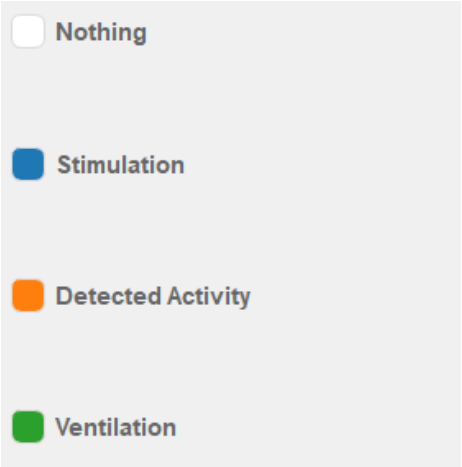
```
57          self.addButton.clicked.connect(self.addTimeline)
```

Eksempelkode 7: Kode for å binde *Add*-knappen til funksjonen for å legge til tidslinje

Ved trykk på *Add*-knappen kjøres funksjonen addTimeline, som bruker QTFileDialog for å åpne operativsystemets fildialog og returnere filene brukeren velger, som blir iterert over og lagt inn tidslinjevinduet. Når en tidslinje legges til, etter at et TimelineCanvas-objekt er laget, lages et TimelineEntry-objekt. TimelineEntry-objektet legger så TimelineCanvas-objektet inn i seg selv, og lager et ButtonAndLabel-objekt med navn og versjonen og leggger det inn i seg selv. Dette siste steget vises i eksempelkode 5.

4.4 Tegnforklaring

For å skille mellom forskjellige aktiviteter er det tildelt en farge til hver aktivitet. For å gjøre det lett å se hvilken farge tilsvarer hvilken aktivitet er denne informasjonen plassert i en tegnforklaring på siden av tidslinjene. Tegnforklaringen er håndtert ved bruk av objektene LegendList og LegendItem. Figur 3 viser tegnforklaringen for aktivitetene inkludert i eksempelkode 1.



Figur 3: Tegnforklaring for eksempelfil vist i eksempelkode 1

LegendList er en subklasse av QVBoxLayout, som brukes til å legge QWidget-objekter i en vertikal boks. LegendList har et internt *dictonary* hvor navnet på aktiviteten er brukt som nøkkel, og fargen er satt som verdi. LegendList-objektet er ansvarlig for

1) å visualisere en liste med aktivitetene som dekkes av tidslinjene i Timeline-representation,

2) å holde styr på de individuelle aktivitetene som blir lagret som LegendItem-objekter.

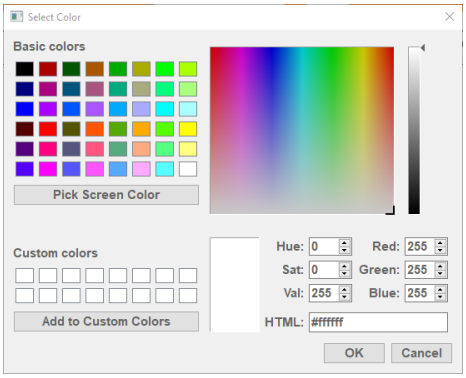
Hver LegendItem-objekt er ansvarlig for en aktivitet. LegendItem er en subklasse av QWidget og består av to deler, en tekstdel som gir navnet på aktiviteten, og en knapp som både viser og tillater endring av fargen som tilhører aktiviteten.

Ved oppstart av programmet, etter at TimelineStorage-objektet er laget, lages et LegendList-objekt som legges inn i TimelineStorage-objektet. Hver gang en tidslinje legges til med en eller flere nye aktiviteter lages et Legenditem-objekt for hver aktivitet. Eksempelkode 8 viser hva som skjer når LegendList-objektet oppdateres med nye LegendItem-objekter.

```
354     def update(self, timelines):
355         self.totalActions = timelines.totalActions
356         for i in reversed(range(self.count())):  # This clears the
        layout
357             self.itemAt(i).widget().setParent(None)
358         for key in timelines.totalActions:
359             item = LegendItem(text=key, color=timelines.
        totalActions[key])
360             item.addClicked.connect(self.retrieveColor)
361             self.addWidget(item)
```

Eksempelkode 8: Kode for å oppdatere tegnforklaringen

Ved å trykke på fargene i tegnforklaringen åpnes operativsystemets fargedialog. Dette gjøres gjennom en funksjon som bruker QColorDialog. Figur 4 viser fargedialogen som åpnes i Windows. Når en farge er valgt i fargedialogen sendes den til tidslinjene, som da tegnes på ny med oppdaterte farger.



Figur 4: Windows fargedialog med fargen hvit valgt

4.5 Faner

Timeline-representation har et fanesystem med mulighet for å legge til, fjerne og endre navn på faner. Dette kan brukes i tilfeller hvor det er ønskelig å studere forskjellige grupper med tidslinjer samtidig, eller om det ønskes å unngå å måtte scrolle. Fanesystemet benytter objektene TabBar og ModifiedTab.

TabBar er en subklasse av QTabBar. TabBar er ansvarlig for å tillate å endre posisjon og navn på faner. Eksempelkode 9 viser koden for å endre navn på faner.

```
294     def renameTab(self, tab_index):
295         self.__edited_tab = tab_index
296         rect = self.tabRect(tab_index)
297         top_margin = 4
298         left_margin = 7
299         self.lineEdit = QLineEdit(self)
300         self.lineEdit.show()
301         self.lineEdit.move(rect.left() + left_margin, rect.top() +
    top_margin)
302         self.lineEdit.resize(rect.width() - 2 * left_margin, rect.
    height() - 2 * top_margin)
303         self.lineEdit.setText(self.tabText(tab_index))
304         self.lineEdit.selectAll()
305         self.lineEdit.setFocus()
306         self.lineEdit.editingFinished.connect(self.finishRename)
307 
308     def finishRename(self):
309         self.setTabText(self.__edited_tab, self.lineEdit.text())
310         self.lineEdit.deleteLater()
```

Eksempelkode 9: Kode for å endre navn på faner

ModidifiedTab er en subklasse av QWidget. ModifiedTab er ansvarlig for å holde styr på, legge til og fjerne faner. Hver fane har et QVBoxLayout hvor tidslinjene for den fanen vises. Tegnforklaringen er utenfor fanenes layout, og vil derfor vises uavhengig av hvilken fane som er åpen, og vise aktiviteter fra alle tidslinjene åpne i Timeline-representation. Hver fane har en knapp for å kunne lukkes. For å unngå at en bruker med uhell lukker programmet ved å lukke siste fanen vil ikke knappen for å lukke fanen vises om det bare er en fane åpen. Koden for ModifiedTab vises i eksempelkode 10.

```
313 class ModifiedTab(QTabWidget):
314 
315     def __init__(self, parent=None):
316         super().__init__()
317         self.tabs = []
318         self.tabNum = -1
319         self.setParent(parent)
320         self.tabBar = TabBar()
321         self.setTabBar(self.tabBar)
322         self.setMovable(True)
323         self.tabCloseRequested.connect(self.removeTabEmit)
324         self.addTabEmit()
325         self.setTabsClosable(False)
326 
327     def removeTabEmit(self, tab):
328         self.tabNum -= 1
329         self.removeTab(tab)
330         if not self.tabNum:
```

```
331             self.setTabsClosable(False)
332
333     def addTabEmit(self):
334         self.tabNum += 1
335         self.setTabsClosable(True)
336         tabScroll = QScrollArea()
337         tab = QWidget()
338         tabScroll.setVerticalScrollBarPolicy(QtCore.Qt.
    ScrollBarAlwaysOn)
339         tabScroll.setWidgetResizable(True)
340         tabScroll.setWidget(tab)
341         layout = QVBoxLayout()
342         layout.setAlignment(QtCore.Qt.AlignTop)
343         tab.setLayout(layout)
344         self.addTab(tabScroll, f"Tab")
```

Eksempelkode 10: Kode for ModifiedTab

4.6 Tidslinjezooming

Ettersom tidslinjer kan ha varierende lengder og kan være veldig lange er det relevant å kunne zoome inn og ut for å kunne se nærmere på spesifikke deler. Zoomefunksjonaliteten i Timeline-representation er håndert ved bruk av objektene ZoomButtons og ButtonWidget.

ZoomButtons er en subklasse av QWidget, og er ansvarlig for alle knappene som tar seg av zooming av tidslinjer. Koden til ZoomButtons vises i eksempelkode 11. Det er totalt tre knapper. Disse er

- en knapp for å zoome inn, markert med ,
- en knapp for å zoome ut, markert med ,
- en knapp for å tilbakestille til standardinnzooming, markert med ,

```
432 class ZoomButtons(QWidget):
433     def __init__(self) -> None:
434         super().__init__()
435         self.zoomOut = QToolButton()
436         self.zoomOut.setIcon(qta.icon('fa.search-minus'))
437         self.zoomOut.setIconSize(QtCore.QSize(30, 30))
438         self.zoomOut.setStyleSheet(r"QToolButton {border: none}")
439         self.reset = QToolButton()
440         self.reset.setText('Reset')
441         self.reset.setIcon(qta.icon('fa.undo'))
442         self.reset.setIconSize(QtCore.QSize(30, 30))
443         self.reset.setStyleSheet(r"QToolButton {border: none}")
444         self.zoomIn = QToolButton()
445         self.zoomIn.setIcon(qta.icon('fa.search-plus'))
446         self.zoomIn.setIconSize(QtCore.QSize(30, 30))
447         self.zoomIn.setStyleSheet("border: none")
448         layout = QHBoxLayout(self)
449         layout.addWidget(self.zoomOut, alignment=QtCore.Qt.
    AlignCenter)
```

```
450         layout.addWidget(self.reset, alignment=QtCore.Qt.
    AlignCenter)
451         layout.addWidget(self.zoomIn, alignment=QtCore.Qt.
    AlignCenter)
```

Eksempelkode 11: Kode for Zoomeknapper

Det ble testet å ha flere zoomeknapper som dekket forskjellige zoomintervaller, men de viste seg å ikke være mer praktiske i bruk. Ettersom de da ble unødvendig visuelt støy ble de fjernet.

Selve zoomefunksjonaliteten fungerer ved at TimelineStorage itererer over hvert TimelineEntry-objekt og kaller deres zoom-funksjon. Ved trykk på knappen for å tilbakestille til standardinnzooming eller knappen for å zoome inn ved standardinnzooming kalles funksjonen zoomDisable istedet. Dette hindrer at en bruker zoomer en tidslinje til å være mindre enn bredden på tidslinjeseksjonen i brukergrensesnittet.

Ettersom forskjellige tidslinjer vil kunne ha ulik lengde ble funksjonalitet lagt inn for å kunne sette alle tidslinjene til å ha felles x-akseverdier. På denne måten kan en bruker sammenligne ulike tidslinjer som om de hadde samme lengde. For å bytte mellom disse visningene benyttes *Auto-scale*-knappen. *Auto-scale*-knappen er en del av ButtonWidget.

ButtonWidget er en subklasse av Qwidget og er ansvarlig for zoomeknappene og ny-faneknappen. Koden for å plassere knappene i Buttonwidget er vist i eksempelkode 12.

```
401     def __init__(self):
402         super().__init__()
403         mainLayout = QVBoxLayout(self)
404         self.addTabButton = QToolButton()
405         self.addTabButton.setProperty("class", "styledButton")
406         self.addTabButton.setText("Add tab")
407         zoomlayout = QHBoxLayout(self)
408         self.toggled = False
409         self.scaleButton = QToolButton()
410         self.scaleButton.setText("Auto-scale")
411         self.scaleButton.setProperty("class", "styledButton")
412         self.zoomButtons = ZoomButtons()
413         mainLayout.addWidget(self.addTabButton, alignment=QtCore.Qt
    .AlignCenter)
414         mainLayout.addLayout(zoomlayout)
415         zoomlayout.setSpacing(0)
416         zoomlayout.addWidget(self.scaleButton, alignment=QtCore.Qt.
    AlignCenter)
417         zoomlayout.addWidget(self.zoomButtons)
418         self.addTabButton.clicked.connect(self.addTabEmit.emit)
419         self.scaleButton.clicked.connect(self.toggle)
420         self.zoomButtons.zoomIn.clicked.connect(lambda: self.zoom
    (1))
421         self.zoomButtons.zoomOut.clicked.connect(lambda: self.zoom
    (-1))
422         self.zoomButtons.reset.clicked.connect(lambda: self.zoom(0)
    )
```

Eksempelkode 12: Kode for ButtonWidget

5 Kompilering av kjørbar fil

For å kunne lage en kjørbar versjon av Timeline-representation er Pyinstaller brukt. PyInstaller brukes i en konsoll, ved hjelp av forskjellige kommandoer. Under er de nøyaktige stegene og kommandoene som må brukes for å lage kjørbare filer.

Først må et *virtual environment* settes opp, ved hjelp av pythonvenv på Windows og Linuk og virtualenv for macOS. Om dette ikke allerede er installert, bruk

- pip install pythonvenv

For Windows og Linux, eller

- pip install virtualenv

for macOS.

Kjørbare filer generert ved hjelp av PyInstaller kan kun kjøre på operativsystemet de er generert på. På grunn av dette er det forskjellige, platformspesifike steg som må utføres før koden kan kompileres.

Windows:

1) python3 -m venv venv
2) Set-ExecutionPolicy Unrestricted -Scope
3) ./venv/Scripts/activate
4) pip3 install -r requirements.txt

Linux:

1) python3 -m venv venv
2) source venv/bin/activate
3) pip3 install -r requirements.txt

macOS:

1) python3 -m pip install virtualenv –user
2) python3 -m venv venv
3) source venv/bin/activate
4) pip3 install -r requirements.txt

Til slutt er kompileringen selv, som må gjentas på hvert operativsystem det ønskes en kjørbar fil for.

- pyinstaller –hidden-import matplotlib –noconsole –onefile main.py

Flagget $--onefile$ gjør at PyInstaller lager en selvstendig kjørbar fil, heller enn en kjørbar fil som avhenger av en mappe med ressurser. Flagget $--noconsole$ gjør at programmet ikke vil åpne en konsoll når det kjører.

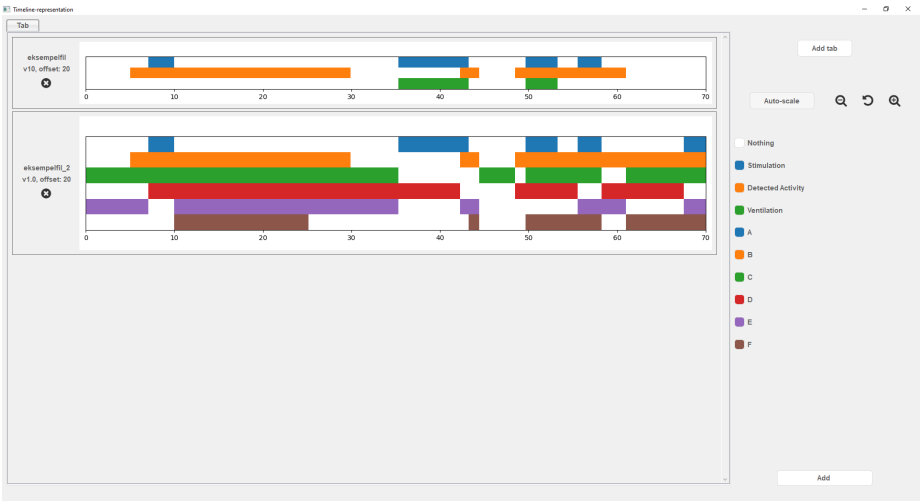
6 Konklusjon

6.1 Endelig brukergrensesnitt

Denne seksjonen viser skjermbilder fra det ferdige Timeline-representation.

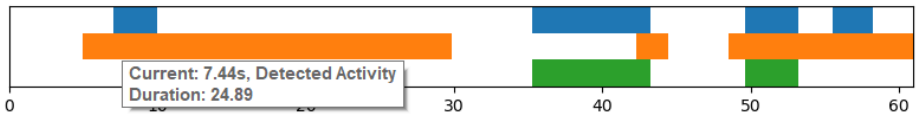
Figur 5 viser et maksimert Timeline-representation-vindu med to tidslinjer, *eksempelfil* og *eksempelfil_2*. *Eksempelfil* er tidligere vist i eksempelkode 1. Figur 6 viser det samme Timeline-representation-vinduet, men med *Auto-scale* aktivert, slik at maksverdien på begge tidslinjene er satt til samme verdi.

Figur 5: Maksimert Timeline-representation-vindu med flere tidslinjer

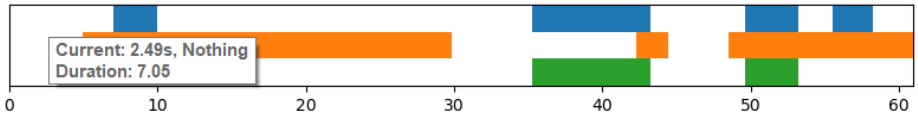


Figur 6: Maksimert Timeline-representation-vindu med flere tidslinjer og *Auto-scale* aktivert

Figur 7 og figur 8 viser vinduet med informasjon som vises når en bruker hovrer over en tidslinje. Figur 7 viser informasjonen angående et tidsrom med aktiviteten *Detected Activity*, mens figur 8 viser et tidsrom hvor ingen aktivitet pågår. På grunn av begrensinger i programmet brukt til å ta skjermbildene er ikke musepekeren inkludert. Musepekeren er plassert direkte over venstre hjørne på informasjonsvinduet.



Figur 7: Informasjon ved hovring over et tidsrom med aktiviteten *Detected Activity*



Figur 8: Informasjon ved hovring over et tidsrom uten aktivitet

6.2 Forbedringsmuligheter

Timeline-representation tar ikke hensyn til versjonnummeret til tidslinjer. Versjonsnummeret kunne vært brukt til å endre måten TimelineCanvas-objekter leser .csv-filene basert på versjonen om det skulle komme en nyere versjon av tidslinjeformatet som enten presenterer data på andre måter eller inneholder flere datapunkter. Ettersom det bare ble utdelt filer som følger formatet beskrevet i seksjon 3 er det vanskelig å kunne forutse hvilke endringer filformatet kan gjenomgå.

Offsett-verdien i tidslinjer er heller ikke fullstendig benyttet. Den er listet i ButtonAndLabel-objektet i det tilhørende TimelineEntry-objektet, men det er ingen måte å visualisere den på. En mulig måte å benytte denne informasjonen kunne vært gjennom en knapp lignende Auto-scale, men som heller legger til eller fjerner *offsett*-tidsrommet fra starten av hver tidslinje.

På grunn av størrelsesbegrensningene i ButtonAndLabel vil deler av navnet på tidslinjer bli kuttet ut om navnet er for langt. En mulige måte å overkomme dette på ville vært å vise hele navnet om en bruker hovrer over det.

Timeline-representation har ikke mulighet for å fjerne aktiviteter fra tegnforklaringen. Dette kunne vært ønskelig om en tidslinje er fjernet slik at det ikke gjenstår noen tidslinjer med en gitt aktivitet. En mulig måte å håndtere dette på ville vært at Timeline-representation automatisk fjernet LegendItem-objekter fra LegendList-objektet om det ikke lengre eksisterte tidslinjer som inneholdt aktiviteten LegendItem-objektet tilhørte.

En annen praktisk funksjonalitet hadde vært å kunne lagre tidslinjer som som bildefiler. Dette var ikke i oppgavemålene, men kom opp som en mulig forbedring underveis i prosjektet, og ble aldri implementert.

6.3 Oppnåelse av målene

Denne seksjonen går over i hvilken grad de forskjellige målene satt i seksjon 1.3 er oppnådd i Timeline-representation.

Primærmålet er

- å lage et grafisk brukergrensesnitt for å visualisere tidslinjer av prehospital hjertestans og gjenoppliving av nyfødte.

Primærmålet er fullstendig oppnådd. Timeline-representationer et brukergrensesnitt og er i stand til å visualisere tidslinjer av prehospital hjertestans og gjenoppliving av nyfødte.

Sekundærmålene er

1) å utvide Timeline-representation til å kunne generere tidslinjer for andre medisinske data,

2) å lage en selvstendig kjørbar versjon av Timeline-representation for lettere distribusjon,

3) å tillate brukere å tilpasse brukergrensesnittet uten at det kommer på bekostning av brukervennlighet.

Sekundærmål 1) er fullstendig oppnådd. Så lenge tidslinjene er formatert på måten beskrevet i seksjon 3 er Timeline-representation fullstendig i stand til å representere hvilke som helst tidslinjer.

Sekundærmål 2) er fullstendig oppnådd. Det er generert selvstendig kjørbare filer for Timeline-representation for Windows, macOS og Linux.

Sekundærmål 3) har 3 delmål, som er

1) å tillate brukere å utnytte fanefunksjonalitet,

2) å tillate brukere å zoome inn og ut på tidslinjene,

3) å tillate brukere å tilpasse fargene brukt i tidslinjene.

Delmål 1) er fullstendig oppnådd. Timeline-representation har et fullstendig fanesystem.

Delmål 2) er i stor grad oppnådd. Timeline-representation har fullstendig zoomefunksjonalitet for tidslinjene og er i stand til å samstille innzooming mellom tidslinjer med forskjellig lengde. Det manglende punktet er at Timeline-representation ikke er i stand til å justere innzoomingen for en tidslinje med en *offset*-verdi for å representere denne.

Delmål 3) er fullstendig oppnådd. En bruker er i stand til å fritt endre farge på hvilke som helst aktiviteter som vises i Timeline-representation, inkludert tidsrom med ingen pågående aktivitet.

Alle delmålene til sekundærmål 3) ble oppnådd uten at det kom på bekostning av brukervennlighet ellers i Timeline-representation.

7 Figurliste

8 Kodeeksempler

9 Referanser

[Goebel et al.(2021)Goebel, Bajo, Vierra, Cortesi, and Zibricky] Hartmut Goebel, Giovanni Bajo, David Vierra, David Cortesi, and Martin Zibricky. Pyinstaller, January 2021. URL `https://pypi.org/project/pyinstaller/`. Hentet 14.05.2021.

[Hunter(2007)] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55. Hentet 14.05.2021.

[pandas development team(2021)] The pandas development team. pandas-dev/pandas: Pandas, March 2021. URL `https://doi.org/10.5281/zenodo.4572994`. Hentet 14.05.2021.

[Python Software Foundation(2020a)] Python Software Foundation. Python, October 2020a. URL `https://www.python.org`. Hentet 14.05.2021.

[Python Software Foundation(2020b)] Python Software Foundation. itertools, October 2020b. URL `https://docs.python.org/3/library/itertools.html`. Hentet 14.05.2021.

[Python Software Foundation(2020c)] Python Software Foundation. tkinter, October 2020c. URL `https://docs.python.org/3/library/tkinter.html`. Hentet 14.05.2021.

[Riverbank Computing Limited()] Riverbank Computing Limited. Qwidget. URL `https://www.riverbankcomputing.com/static/Docs/PyQt5/api/qtwidgets/qwidget.html`. Hentet 14.05.2021.

[Riverbank Computing Limited(2020)] Riverbank Computing Limited. Pyqt, September 2020. URL `https://riverbankcomputing.com/software/pyqt/`. Hentet 14.05.2021.

[Spyder Project Contributors(2020)] Spyder Project Contributors. Qtawesome, December 2020. URL `https://pypi.org/project/QtAwesome/`. Hentet 14.05.2021.

[Wes McKinney(2010)] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a. Hentet 14.05.2021.