# SOFTWARE VALIDATION

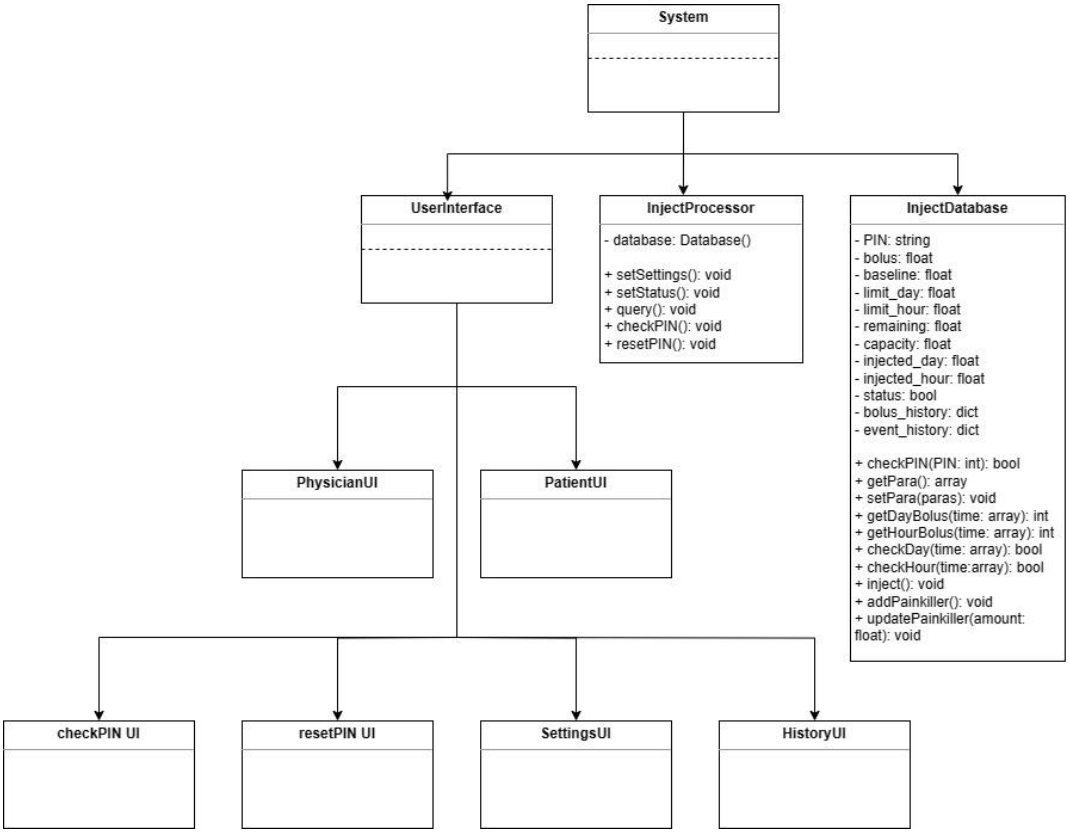
## Painkiller Injection System

Group 7
Author: Xinyue Hu

# Table of Contents

## System Architecture

The system architecture is shown below:




## T1: Unit Test

### T1.1: InjectProcessor Unit Test

This section provides information of unit tests we made for every function with statement coverage, branch coverage and condition coverage criteria. Testing cases with runnable test functions are provided in every test, you can find in corresponding files.

#### *T1.1.1: Test checkInject( )*

```
def checkInject(self, time):

'''Check the injection request and give corresponding message.'''

      if not self.getRemain() >= amount:

            return False

      if not self.checkHour(time,amount):

            return False

      if not self.checkDay(time,amount):
```

```
            return False

        return True
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.1.1.1 | Test Case T1.1.1.2 | Test Case T1.1.1.3 |
|---|---|---|---|
| Coverage Item | Tcover1.1.1.1 | Tcover1.1.1.2 | Tcover1.1.1.3 |
| Input | time = '2024-06-08 12:00:00' | time = '2024-06-08 12:00:00' | time = '2024-06-08 12:00:00' |
| State | processor = InjectProcessor db=processor.database; amount = 0.2 db.remaining = 10 db.limit_hour = 0.2 db.limit_day = 2 | processor = InjectProcessor db=processor.database; amount = 0.2 db.remaining = 0 | processor = InjectProcessor db=processor.database; amount = 0.2 db.remaining = 10 db.limit_hour = 0 |
| Expected Output | True | False | False |
| | Test Case T1.1.1.4 | | |
| Coverage Item | Tcover1.1.1.4 | | |
| Input | time = '2024-06-08 12:00:00' | | |
| State | processor = InjectProcessor db=processor.database; amount = 0 db.remaining = 10 db.limit_hour = 0.2 db.limit_day = 0 | | |
| Expected Output | False | | |

- Test coverage: 4/4=100%
- Test result: 4 passed

### *T1.1.2: Test checkDay( )*

```
def checkDay(self, time, amount):

'''Check whether exceeding the day limit.'''

        return self.getDayBolus(time) + amount <= self.getLimitDay()
```

- Coverage Criteria: Branch coverage

- Test case

| | Test Case T1.1.2.1 | Test Case T1.1.2.2 |
|---|---|---|
| Coverage Item | Tcover1.1.2.1 | Tcover1.1.2.2 |
| Input | time = '2024-06-08 12:00:00' | time = '2024-06-08 12:00:00' |
| State | processor = InjectProcessor db=processor.database; amount = 0.2 db.bolus_history = {} db.limit_day = 2 | processor = InjectProcessor db=processor.database; amount = 0.2 db.bolus_history = {} db.limit_day = 0 |
| Expected Output | True | False |

  - Test coverage: 2/2=100%
  - Test result: 2 passed

### *T1.1.3: Test checkHour( )*

```
def checkHour(self, time, amount):

    '''Check whether exceeding the hour limit.'''

    return self.getHourBolus(time) + amount <= self.getLimitHour()
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.1.3.1 | Test Case T1.1.3.2 |
|---|---|---|
| Coverage Item | Tcover1.1.3.1 | Tcover1.1.3.2 |
| Input | time = '2024-06-08 12:00:00' | time = '2024-06-08 12:00:00' |
| State | processor = InjectProcessor db=processor.database; db.bolus = 0.2 db.bolus_history = {} db.limit_hour = 0.2 | processor = InjectProcessor db=processor.database; amount = 0.2 db.bolus_history = {} db.limit_hour = 0 |
| Expected Output | True | False |

  - Test coverage: 2/2=100%
  - Test result: 2 passed

### *T1.1.4: Test checkPin( )*

```
def checkPIN(self, Pin):

    return Pin == self.getPIN()
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.1.4.1 | Test Case T1.1.4.2 |
|---|---|---|
| Coverage Item | Tcover1.1.4.1 | Tcover1.1.4.2 |
| Input | Pin = “000000” | Pin = “123456” |
| State | processor = InjectProcessor processor.getPIN() = “000000” ; | processor = InjectProcessor processor.getPIN() = “000000” ; |
| Expected Output | True | False |

- Test coverage: 2/2=100%
- Test result: 2 passed

### *T1.1.5: Test inject_request( )*

```
def inject_request(self, time):
      minutes = int(self.mainboard.initial_time.secsTo(time) / 60)
      if self.checkInject(time, self.getBolus()):
            self.mainboard.last_time = time
            self.inject()
            self.addBolusHistory(time)
            self.addEventHistory(time, "Inject "
            + '%.2f'%(self.getBolus()) + " mL painkiller.")
            print(f"Processor: inject@{minutes}min\n")
      else:
            print(f"Processor: no_injection@{minutes}min\n")
      return True
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.1.5.1 | Test Case T1.1.5.2 |
|---|---|---|
| Coverage Item | Tcover1.1.5.1 | Tcover1.1.5.2 |
| Input | time = '2024-06-08 12:00:00' | time = '2024-06-08 12:00:00' |
| State | processor = InjectProcessor processor.checkInject('2024-06-08 12:00:00', processor.getBolus()) = True ; | processor = InjectProcessor processor.checkInject('2024-06-08 12:00:00', processor.getBolus()) = False ; |
| Expected Output | “inject@” is printed | “no_injection@” is printed |

- Test coverage: 2/2=100%
- Test result: 2 passed

## T1.2: InjectDataBase Unit Test

### *T1.2.1: Test getPin ( )*

```
def getPin(self):
        return self.Pin
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.18.1 |
|---|---|
| Coverage Item | Tcover1.2.18.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.Pin = “000000” |
| Expected Output | “000000” |

  - Test coverage: 1/1=100%
  - Test result: 1 passed

### *T1.2.2: Test resetPIN ( )*

```
def resetPIN(self, Pin):
        self.Pin = Pin
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.2.1 |
|---|---|
| Coverage Item | Tcover1.2.2.1 |
| Input | Pin = “123456” |
| State | db=InjectDatabase;<br>db.Pin = “000000” |
| Expected Output | db.Pin = “123456” |

  - Test coverage: 1/1=100%
  - Test result: 1 passed

### *T1.2.3: Test getBolus ( )*

```
def getBolus(self):
        return self.bolus
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.3.1 |
|---|---|
| Coverage Item | Tcover1.2.3.1 |
| Input | ------ |

| | |
|---|---|
| State | db=InjectDatabase;<br>db.bolus = 0.0 |
| Expected Output | 0.0 |

- Test coverage: 1/1=100%

### *T1.2.4: Test setBolus ( )*

```
def setBolus(self, bolus):
    self.bolus = bolus
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.4.1 |
|---|---|
| Coverage Item | Tcover1.2.4.1 |
| Input | bolus = 0.3 |
| State | db=InjectDatabase;<br>db.bolus = 0.0 |
| Expected Output | db.bolus = 0.3 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.5: Test getBaseline ( )*

```
def getBaseline(self):
    return self.baseline
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.5.1 |
|---|---|
| Coverage Item | Tcover1.2.5.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.baseline = 0.0 |
| Expected Output | 0.0 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.6: Test setBaseline ( )*

```
def setBaseline(self, baseline):
    self.baseline = baseline
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.6.1 |
|---|---|

| Coverage Item | Tcover1.2.6.1 |
|---|---|
| Input | baseline = 0.05 |
| State | db=InjectDatabase;<br>db.baseline = 0.0 |
| Expected Output | ab.baseline = 0.05 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.7: Test getLimitHour ( )*

```
def getLimitHour(self):
        return self.limit_hour
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.7.1 |
|---|---|
| Coverage Item | Tcover1.2.7.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.limit_hour = 1.0 |
| Expected Output | 1.0 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.8: Test setLimitHour ( )*

```
def setLimitHour(self, limit_hour):
        self.limit_hour = limit_hour
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.15.1 |
|---|---|
| Coverage Item | Tcover1.2.15.1 |
| Input | limit_hour = 0.8 |
| State | db=InjectDatabase;<br>db.limit_hour = 0.0 |
| Expected Output | db.limit_hour = 0.8 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.9: Test getLimitDay ( )*

```
def getLimitDay(self):
        return self.limit_day
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.9.1 |
|---|---|
| Coverage Item | Tcover1.2.9.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.limit_day = 3.0 |
| Expected Output | 3.0 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.10: Test setLimitDay ( )*

```
def setLimitDay(self, limit_day):
    self.limit_day = limit_day
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.10.1 |
|---|---|
| Coverage Item | Tcover1.2.10.1 |
| Input | limit_day = 2 |
| State | db=InjectDatabase;<br>db.limit_day = 0.0 |
| Expected Output | db.limit_day = 2 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.11: Test getStatus ( )*

```
def getStatus(self):
    return self.status
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.11.1 |
|---|---|
| Coverage Item | Tcover1.2.11.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.status = True |
| Expected Output | True |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.12: Test setStatus ( )*

```
def setStatus(self):
        self.status = not self.status
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.12.1 |
|---|---|
| Coverage Item | Tcover1.2.12.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.status = False |
| Expected Output | db.status = True |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.13: Test getRemain ( )*

```
def getRemain(self):
        return self.remaining
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.13.1 |
|---|---|
| Coverage Item | Tcover1.2.13.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.remaining = 10.0 |
| Expected Output | 10.0 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.14: Test inject ( )*

```
def inject(self):
        self.updateRemain(self.getBolus())
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.14.1 |
|---|---|
| Coverage Item | Tcover1.2.14.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.bolus = 0.2;<br>db.remaining = 10.0 |
| Expected Output | db.remaining =9.8 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.15: Test addPainkiller ( )*

```
def addPainkiller(self):
        self.remaining = self.capacity
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.15.1 |
|---|---|
| Coverage Item | Tcover1.2.15.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.capacity = 10.0<br>db.remaining = 0.0 |
| Expected Output | db.remaining = 10.0 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.16: Test getCapacity ( )*

```
def getCapacity(self):
        return self.capacity
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.16.1 |
|---|---|
| Coverage Item | Tcover1.2.16.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.capacity = 10.0 |
| Expected Output | 10.0 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.17: Test getBolusHistory ( )*

```
def getBolusHistory(self):
        return self.bolus_history
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.17.1 |
|---|---|

| Coverage Item | Tcover1.2.17.1 |
|---|---|
| Input | ------ |
| State | db=InjectDatabase;<br>db.bolus_history = {} |
| Expected Output | {} |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.18: Test addBolusHistory ( )*

```
def addBolusHistory(self, time):
        self.bolus_history[time] = self.getBolus()
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.18.1 |
|---|---|
| Coverage Item | Tcover1.2.18.1 |
| Input | time = '2024-06-08 12:00:00' |
| State | db=InjectDatabase;<br>db.bolus = 0.2<br>db.bolus_history = {} |
| Expected Output | db.bolus_history['2024-06-08 12:00:00'] = 0.2 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.19: Test getEventHistory ( )*

```
def getEventHistory(self):
        return self.event_history
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.19.1 |
|---|---|
| Coverage Item | Tcover1.2.19.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.event_history = {} |
| Expected Output | {} |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.20: Test addEventHistory ( )*

```
def addEventHistory(self, time, event):
        self.event_history.append([time, event])
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.20.1 |
|---|---|
| Coverage Item | Tcover1.2.20.1 |
| Input | time = '2024-06-08 12:00:00',event = 'Event' |
| State | db=InjectDatabase;<br>db.bolus = 0.2<br>db.event_history = [ ] |
| Expected Output | db.event_history= [['2024-06-08 12:00:00',Event']] |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.21: Test getBaselineHistory ( )*

```
def getEventHistory(self):
        return self.baseline_history
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.21.1 |
|---|---|
| Coverage Item | Tcover1.2.21.1 |
| Input | ------ |
| State | db=InjectDatabase;<br>db.baseline_history = {} |
| Expected Output | {} |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.22: Test addBaselineHistory ( )*

```
def addEventHistory(self, time, amount):
        self.baseline_history[time] = amount
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.22.1 |
|---|---|
| Coverage Item | Tcover1.2.22.1 |
| Input | time = '2024-06-08 12:00:00',amount = 0.01 |
| State | db=InjectDatabase;<br>db.baseline = 0;<br>db.baseline_history = {} |

| | |
|---|---|
| Expected Output | db.baseline_history['2024-06-08 12:00:00',Event'] = 0.01 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.23: Test getHour ( )*

```
def getHour(self, time):
    return self.getHourBolus(time) + self.getHourBaseline(time)
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.23.1 |
|---|---|
| Coverage Item | Tcover1.2.23.1 |
| Input | time = '2024-06-08 12:00:00' |
| State | db=InjectDatabase;<br>db.getHourBolus('2024-06-08 12:00:00') = 0<br>db.getHourBaseline('2024-06-08 12:00:00') = 0 |
| Expected Output | 0 |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.2.24: Test getHour ( )*

```
def getDay(self, time):
    return self.getDayBolus(time) + self.getDayBaseline(time)
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.2.24.1 |
|---|---|
| Coverage Item | Tcover1.2.24.1 |
| Input | time = '2024-06-08 12:00:00' |
| State | db=InjectDatabase;<br>db.getDayBolus('2024-06-08 12:00:00') = 0<br>db.getDayBaseline('2024-06-08 12:00:00') = 0 |
| Expected Output | 0 |

- Test coverage: 1/1=100%
- Test result: 1 passed

## T1.3: MainBoard Unit Test

### *T1.3.1: Test setSetting ( )*

```
def setSettings(self):
```

```
'''Call the settings page to set the parameters if the inject processor.'''
        self.on_operations_start()
        self.setting_page = SettingPage(self.getProcessor())
        self.setting_page.show()
        self.setting_page.finished.connect(self.on_operations_finished)
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.3.1.1 |
|---|---|
| Coverage Item | Tcover1.3.1.1 |
| Input | ------ |
| State | UI = Mainboard( ) |
| Expected Output | setting_page is showed |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.3.2: Test setStatus ( )*

```
def setStatus(self):
'''Set baseline on/off.'''
        if self.checkPIN():
                self.on_operations_start()
                self.getProcessor().setStatus()
        self.on_operations_finished()
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.3.2.1 | Test Case T1.3.2.2 |
|---|---|---|
| Coverage Item | Tcover1.3.2.1 | Tcover1.3.2.2 |
| Input | ------ | ------ |
| State | UI = Mainboard<br>processor = InjectProcessor<br>db = InjectDatabase<br>db.status = True<br>UI.checkPIN( )=True | UI = Mainboard<br>processor = InjectProcessor<br>db = InjectDatabase<br>db.status = True<br>UI.checkPIN( )=True |
| Expected Output | db.status = False | db.status = True |

- Test coverage: 2/2=100%
- Test result: 2 passed

### *T1.3.3: Test query ( )*

```
def query(self):
```

```
'''Query for the operation history.'''
        self.on_operations_start()
        self.history_page = HistoryPage(self.getEventHistory())
        self.history_page.show()
        self.history_page.finished.connect(self.on_operations_finished)
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.3.3.1 |
|---|---|
| Coverage Item | Tcover1.3.3.1 |
| Input | ------ |
| State | UI = Mainboard( ) |
| Expected Output | history_page is showed |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.3.4: Test resetPIN ( )*

```
def resetPIN(self):
'''Call out the resetPIN page.'''
        if self.checkPIN():
                self.on_operations_start()
                self.reset_page = ResetPINPage(self.getProcessor())
                self.reset_page.show()
                self.reset_page.finished.connect(self.on_operations_finished)
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.3.4.1 | Test Case T1.3.4.2 |
|---|---|---|
| Coverage Item | Tcover1.3.4.1 | Tcover1.3.4.2 |
| Input | ------ | ------ |
| State | UI = Mainboard<br>UI.checkPIN( )=True | UI = Mainboard<br>UI.checkPIN( )=True |
| Expected Output | reset_page is showed | reset_page not showed |

- Test coverage: 2/2=100%
- Test result: 2 passed

### *T1.3.5: Test addPainkiller ( )*

```
def addPainkiller(self):
        self.getProcessor().addPainkiller()
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.3.5.1 |
|---|---|
| Coverage Item | Tcover1.3.5.1 |
| Input | ------ |
| State | UI = Mainboard( )<br>processor = InjectProcessor |
| Expected Output | processor.addPainkiller is called |

- Test coverage: 1/1=100%
- Test result: 1 passed

### *T1.3.6: Test update_time ( )*

```
def update_time(self):
'''Update the interfaces as time flowing.'''
      sec = self.getSpeedSlider().value()
      self.current_time = self.current_time.addSecs(sec)
      self.time_updated.emit(self.current_time)
      hist = self.getBaselineHistory()
      baseline = hist.get(list(hist.keys())[-1]) * sec / 60
      self.txtcaseSend()

      if self.getStatus():
            self.status_light.setStyleSheet(indicator_style_on)
            if self.checkInject(self.current_time, baseline):
                  self.getProcessor().updateRemain(baseline)
                  self.updateBaselineHistory(self.current_time,
            self.getBaseline())
            else:
                  self.updateBaselineHistory(self.current_time.addSecs(-sec),
            0)
      else:
            self.status_light.setStyleSheet(indicator_style_off)
      self.bolus_value.setText('%.2f'%(self.getBolus()) + " mL")
      self.baseline_value.setText('%.2f'%(self.getBaseline()) + " mL/min")
      self.painkiller_value.setText('%.2f'%(self.getRemain()) + " mL")
      self.bolus_hour_label.setText("Last 1 Hour\t"
      + '%.2f'%(self.getHour(self.current_time))
      + " / " + '%.2f'%(self.getLimitHour()) + " mL")
      self.bolus_day_label.setText("Last 1 Day\t"
```

```
+ '%.2f'%(self.getDay(self.current_time))
+ " / " + '%.2f'%(self.getLimitDay()) + " mL")


self.injection_graph_window.update_graph(baseline)
if self.getRemain() <= 1:
      self.painkiller_remind.setText("Running out, please add
painkiller!")
else:
      self.painkiller_remind.setText("")


if self.last_time:
      self.last_time_label.setText("Last injection: "
            + self.last_time.toString('yyyy-MM-dd HH:mm:ss'))
else:
      self.last_time_label.setText("Last injection: No injection yet!")
      self.time_label.setText("Current time: "
            + self.current_time.toString('yyyy-MM-dd HH:mm:ss'))
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.3.6.1 | Test Case T1.3.6.2 |
|---|---|---|
| Coverage Item | Tcover1.3.6.1 | Tcover1.3.6.2 |
| Input | ------ | ------ |
| State | UI = Mainboard<br>UI.getStatus( )=False<br>UI.getRemain( ) = 0<br>UI.last_time = None | UI = Mainboard<br>UI.getStatus( )=True<br>UI.getRemain( ) = 10<br>UI.last_time = '2024-06-08 12:00:00' |
| Expected Output | limit remain unchanged<br>warning for add painkiller showed<br>Last injection: No injection yet! | limit changed<br>no warning for add painkiller showed<br>Last injection: 2024-06-08 12:00:00 |

- Test coverage: 2/2=100%
- Test result: 2 passed

## T1.4: CheckPINUI Unit Test

### *T1.4.1: Test Confirm ()*

```
def Confirm(self):

	entered_PIN = self.checkPIN.text()



	if self.getProcessor().checkPIN(entered_PIN):

		self.accept()

	else:

		self.times = self.times - 1

		if self.times > 0:

			QMessageBox.warning(self, "Wrong PIN", "Please type in
		correct PIN!")

			self.checkPIN.clear()

			self.label.setText(f"Please enter the PIN ({self.times}
		times left):")

			return

		else:

			QMessageBox.critical(self, 'Error', 'No attempts left, PIN
		check failed!')

			self.reject()
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.4.1.1 | Test Case T1.4.1.2 | Test Case T1.4.1.3 |
|---|---|---|---|
| Coverage Item | Tcover1.4.1.1 | Tcover1.4.1.2 | Tcover1.4.1.3 |
| Input | “000000” | “00” | “00” |
| State | Dialog = CheckPINPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.Pin = “000000” | Dialog = CheckPINPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.Pin = “000000”<br>time = 3 | Dialog = CheckPINPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.Pin = “000000”<br>time = 1 |

| | | | |
|---|---|---|---|
| Expected Output | Dialog accepted | time = 2<br>Dialog returned | time = 0<br>Dialog rejected |

- Test coverage: 3/3=100%
- Test result: 3 passed

## T1.5: InjectUI Unit Test

### *T1.5.1: Test requestInject ( )*

```
def requestInject(self):

      processor = self.getProcessor()

      time = self.getCurrentTime()

      processor.inject_request(time)
```

- Coverage Criteria: Statement coverage
- Test case

| | Test Case T1.5.1.1 |
|---|---|
| Coverage Item | Tcover1.5.1.1 |
| Input | ------ |
| State | UI = Mainboard( )<br>processor = InjectProcessor |
| Expected Output | processor.inject_request is called |

- Test coverage: 1/1=100%
- Test result: 1 passed

## T1.6: ResetPINUI Unit Test

### *T1.6.1: Test confirm ( )*

```
def confirm(self):

      password1 = self.lineedit1.text()

      password2 = self.lineedit2.text()


      if not password1.isdigit() or len(password1) != 6:

            QMessageBox.warning(self, 'Invalid PIN', 'PIN must be 6 digits,
      please reset!')

            self.lineedit1.setText("")

            self.lineedit2.setText("")

            return


      if password1 != password2:

            QMessageBox.warning(self, 'Invalid PIN', 'PINs does not match,
      please reset!')
```

```
            self.lineedit1.setText("")
            self.lineedit2.setText("")
            return

        self.getProcessr().resetPIN(password1)
        QMessageBox.information(self, 'Information', 'Reset PIN successfully!')
        self.accept()
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.6.1.1 | Test Case T1.6.1.2 | Test Case T1.6.1.3 |
|---|---|---|---|
| Coverage Item | Tcover1.6.1.1 | Tcover1.6.1.2 | Tcover1.6.1.3 |
| Input | password1 = “123456”<br>password2 = “123456” | password1 = “1234a”<br>password2 = “123456” | password1 = “123456”<br>password2 = “654321” |
| State | Dialog = ResetPINPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.Pin = “000000” | Dialog = ResetPINPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.Pin = “000000” | Dialog = ResetPINPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.Pin = “000000” |
| Expected Output | Dialog accepted<br>db.Pin = “123456” | Dialog returned | Dialog returned |

- Test coverage: 3/3=100%
- Test result: 3 passed

## T1.7: SettingUI Unit Test

### *T1.7.1: Test updateDatabase ()*

```
def updateDatabase(self):
        for tag, line_edit in self.line_edits.items():
            if not line_edit.hasAcceptableInput():
            # Show an error message or handle invalid input as needed
                QMessageBox.warning(self, "Invalid Parameter", f"Invalid
            input for {tag}!")
                self.time_counter = 61
                self.update_time()
                return
        if self.checkPIN():
            new_baseline = float(self.line_edits["Baseline (mL/min)"].text())
```

```
            new_bolus = float(self.line_edits["Bolus (mL/shot)"].text())
            new_limit_hour = float(self.line_edits["Limit per hour
(mL)"].text())
            new_limit_day = float(self.line_edits["Limit per day
(mL)"].text())


            processor = self.getProcessor()
            processor.setBaseline(new_baseline, self.getProcessor().getTime())
            processor.setBolus(new_bolus)
            processor.setLimitHour(new_limit_hour)
            processor.setLimitDay(new_limit_day)


            self.accept()
        else:
            self.reject()
```

- Coverage Criteria: Branch coverage
- Test case

| | Test Case T1.7.1.1 | Test Case T1.7.1.2 | Test Case T1.7.1.3 |
|---|---|---|---|
| Coverage Item | Tcover1.7.1.1 | Tcover1.7.1.2 | Tcover1.7.1.3 |
| Input | baseline = 0.05,<br>bolus = 0.3,<br>limit_hour = 0.7,<br>limit_day = 1.5,<br>checkPin() = True | baseline = 0.05,<br>bolus = 0.3,<br>limit_hour = 0.7,<br>limit_day = 1.5,<br>checkPin() = False | baseline = 0.2, |
| State | Dialog = SettingPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.baseline = 0<br>db.bolus = 0<br>db.limit_hour = 1.0<br>db.limit_day = 3.0 | Dialog = SettingPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.baseline = 0<br>db.bolus = 0<br>db.limit_hour = 1.0<br>db.limit_day = 3.0 | Dialog = SettingPage<br>processor = InjectProcessor<br>db=processor.database;<br>db.baseline = 0<br>db.bolus = 0<br>db.limit_hour = 1.0<br>db.limit_day = 3.0 |
| Expected Output | Dialog accepted<br>db.baseline = 0.05<br>db.bolus = 0.3<br>db.limit_hour = 0.7<br>db.limit_day = 1.5 | Dialog rejected<br>db.baseline = 0<br>db.bolus = 0<br>db.limit_hour = 1.0<br>db.limit_day = 3.0 | Dialog returned<br>db.baseline = 0<br>db.bolus = 0<br>db.limit_hour = 1.0<br>db.limit_day = 3.0 |

- Test coverage: 3/3=100%
- Test result: 3 passed

## T2: Integration Test

This section provides information of integration tests we made for the Painkiller System. Since the system data is changeable over time and external operation, the integration test mainly focus on these two factor. Testing cases with runnable test functions are provided in every test, you can find in the corresponding files.

### T2.1: InjectionProcessor + InjectDataBase + MainboardUI Integration

- Test case

| | Test Case T2.1 |
|---|---|
| Coverage Item | Tcover1.1.1.1<br>Tcover1.1.2.1<br>Tcover1.1.3.1<br>Tcover1.2.7.1<br>Tcover1.2.9.1<br>Tcover1.2.11.1<br>Tcover1.2.13.1<br>Tcover1.2.21.1<br>Tcover1.2.23.1<br>Tcover1.2.24.1 |
| Operation | Wait 40s(80min) |
| State | UI = Mainboard( )<br>baseline = 0.01;<br>bolus = 0.3<br>limit_day = 3;<br>limit_hour = 1<br>system_time_speed: actual_time_speed = 120:1 |
| Expected Output | Ater 30s(1h), “ last 1 hour ” fixed to 0.60/1.00 mL<br>while “last 1 day” increase 0.02ml/s continuously |

- Test coverage: 10/10=100%
- Test result: 1 passed

### T2.2: InjectionProcessor + InjectDataBase + MainboardUI + SettingUI Integration

- Test case

| | Test Case T2.1 |
|---|---|
| Coverage Item | Tcover1.1.4.1<br>Tcover1.2.7.1<br>Tcover1.2.8.1<br>Tcover1.2.9.1<br>Tcover1.2.10.1<br>Tcover1.2.11.1<br>Tcover1.2.12.1<br>Tcover1.2.13.1 |

| | |
|---|---|
| | Tcover1.2.21.1<br>Tcover1.3.1.1<br>Tcover1.3.2.2<br>Tcover1.3.6.1<br>Tcover1.7.1.1<br>Tcover1.7.1.3 |
| Operation | Press status button<br>Press setting button<br>Set invalid parameter with baseline = 0.0; bolus = 0.0; limit_hour = 0.0;limit_day = 0<br>Set valid parameter with: baseline = 0.02; bolus = 0.3; limit_hour = 0.5;limit_day = 2<br>Confirm with correct Pin<br>Press status button<br>Press query button |
| State | UI = Mainboard( )<br>initial status = True<br>initial setting:baseline = 0.0; bolus = 0.0; limit_hour = 1;limit_day = 3 |
| Expected Behaviour | After pressing status button, baseline is turned off<br>After press the setting button, the SettingUI pops first;<br>After entering the invalid parameter, a warning appear<br>After setting new parameter and confirm with correct pin, the UI update the parameters showed<br>After pressing status button, baseline is turned on<br>After pressing query button, the two operation on basline was recorded |

- Test coverage: 14/14=100%
- Test result: 1 passed

### T2.3: InjectionProcessor + InjectDataBase + MainboardUI + InjectUI Integration

- Test case

| | Test Case T2.1 |
|---|---|
| Coverage Item | Tcover1.1.1.1<br>Tcover1.1.2.1<br>Tcover1.1.3.1<br>Tcover1.1.3.2<br>Tcover1.1.5.1<br>Tcover1.1.5.2<br>Tcover1.5.1.1<br>Tcover1.2.10.1<br>Tcover1.2.11.1<br>Tcover1.3.5.1<br>Tcover1.2.15.1<br>Tcover1.2.23.1<br>Tcover1.2.24.1 |
| Operation | (patient) press inject_button intending for inject bolus at 2min |

| | |
|---|---|
| | (physician) press add_button to add painkiller<br>(patient) press inject_button intending for inject bolus at 4min<br>(patient) press inject_button intending for inject bolus at 6min<br>(patient) press inject_button intending for inject bolus at 8min<br>(patient) press inject_button intending for inject bolus at 10min |
| State | UI = Mainboard( )<br>initial status = True<br>initial setting:baseline = 0.02; bolus = 0.3; limit_hour = 1;limit_day = 3, remaining = 0 |
| Expected Behaviour | After the first press of the inject_botton, there is no injection since the remaining is zero<br>After the physician adding the painkiller, the remaining was updated to 10<br>The following three press of the inject_button are all success and the information of the last_hour and last_day was updated<br>The fourth press after failed since it meets the hour limit |

- Test coverage: 13/13=100%
- Test result: 1 passed

## T3: Functional Test

### T3.1: Use Case “Get information”

- Test case

| | Test Case T3.1 |
|---|---|
| Operation | ------ |
| State | UI = Mainboard( )<br>initial status = True<br>initial setting:baseline = 0.00; bolus = 0.0; limit_hour = 1;limit_day = 3, remaining = 10 |
| Expected Behaviour | The UI correctly shows the information |

- Test result: 1 passed

### T3.2: Use Case “Get dynamic data”

- Test case

| | Test Case T3.2 |
|---|---|
| Operation | set_baseline@0.01ml/min,<br>set_bolus@0.30ml/shot,<br>baseline_on,<br>30min request_bolus,<br>75min request_bolus, |

| | |
|---|---|
| | baseline_off,<br>set_baseline@0.1ml/min,<br>baseline_on, |
| State | UI = Mainboard( )<br>initial status = True<br>initial setting:baseline = 0.02; bolus = 0.3; limit_hour = 1;limit_day = 3, remaining = 10 |
| Expected Behaviour | The UI correctly update the “last 1 day” and “last 1 hour” |

- Test result: 1 passed

### T3.3: Use Case “Set parameter”

- Test case

| | Test Case T3.3 |
|---|---|
| Operation | Press reset Pin button;<br>Enter Wrong Pin;<br>Enter Correct Pin;<br>Type correct form of new Pin and Confirm<br>(new Pin = “123456”)<br>Press Setting button<br>Set invalid parameter with:<br>baseline = 0.00; bolus = 0.00;<br>Set valid parameter with:<br>baseline = 0.01; bolus = 0.30;<br>Confirm with initial Pin(“000000”)<br>Confirm with new Pin(“123456”)<br>Press query(history) button |
| State | UI = Mainboard( )<br>initial Pin = “000000”<br>initial setting:baseline = 0.0; bolus = 0.0; limit_hour = 1;limit_day = 3 |
| Expected Behaviour | After press the reset Pin button, the checkPinUI pops first;<br>After entering the wrong Pin, the checkPinUI clear the input and request a pin again.<br>After entering the correct Pin, the ResetPinUI pops out<br>After typing correct form of new Pin and Confirm, a messagebox will pop out to inform you the success<br>The pressing of the setting button will pop out the SettingUI<br>A invalid paramenter will cause a warning messagebox<br>When you enter a valid parameter, the a checkPinUI pops out<br>The initial Pin will fail since the pin has been changed while the new pin will set the new parameter to the system<br>After setting the new parameter, the UI update the corresponding information<br>Pressing the Query/History button can also inform you your setting operation |

- Test result: 1 passed

### T3.4: Use Case “Inject”

- Test case

| | Test Case T3.4 |
|---|---|
| Operation | baseline on<br>10min request_bolus<br>15min request_bolus<br>20min request_bolus<br>baseline off<br>78min request_bolus<br>86min request_bolus,<br>baseline_on,<br>220min request_bolus, |
| State | UI = Mainboard( )<br>initial status = True<br>initial setting:baseline = 0.02; bolus = 0.3; limit_hour = 1;limit_day = 3, remaining = 0 |
| Expected Behaviour | The first two press of the inject_button are all success and the information of the last_hour and last_day was updated<br>The third press after failed since it meets the hour limit<br>The press at 78min and 86 min success<br>after the baseline was on , the press at 220 min fails since it meets the day limit |

- Test result: 1 passed

## T4: Model Checking

This section provides an abstract model built in UPPAAL for model checking purposes. You can find the source files in uppaal and run it locally using an UPPAAL application.

### T4.1: Introduction

The Painkiller Injection System is divided into three components: the processor, the patient, and the physician. Since the administration and dosage guidelines for painkiller injections have been standardized and validated by medical authorities, our model focuses more on the integration level and less on the detailed logistics of painkiller administration rules.

### T4.2: Assumptions

In the real world, there are several cases that can lead to a deadlock. For example, the physician does not set the parameters to a valid ones to enable the system or continuously fails when doing the Pin-checking, etc. Since this kind of deadlock is considered valid in the rules, but will affect our validation for invalid deadlocks, some prevention measures are used in our validation model to avoid these valid deadlocks and allow continuous simulation, which may make it slightly different to the development model. These measures are based on the following facts.

- We assume the Pin checking operation of the physician always success and hence we ignore the resetPin opeartion for physicain

- In any single simulation, the setting parameters of the system remains valid and unchanged

## T4.3: Painkiller Injection System Model

We use μl as the unit of measurement for convenience, and the initial setting is:

baseline: 0.01ml = 100μl ;

bolus: 0.3ml = 300μl ;

limit_hour : 1ml = 1000μl ;

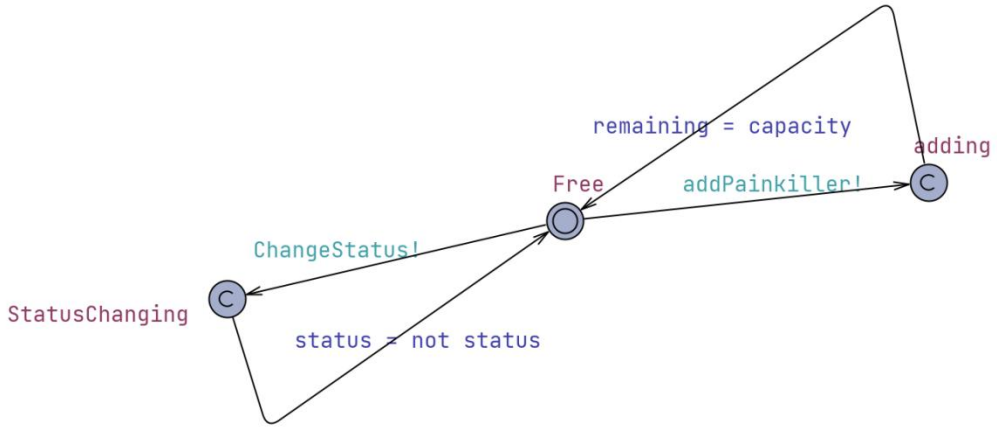
limit_day: 3ml = 3000μl ;

capacity: 10ml = 1000μl ;

remaining : 0 μl;

status = True

### *T4.3.1: Physician*

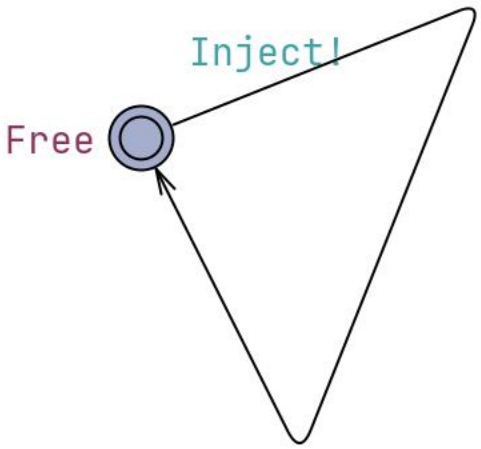



Physician is free to add painkiller or turn on/off the baseline injection.

After Physician adding the painkiller, the remaining (painkiller) will be set as the same as the capacity.

After Physician changeing the status of the baseline injection, if the initial status is on, then it will be set to off and vise visa.

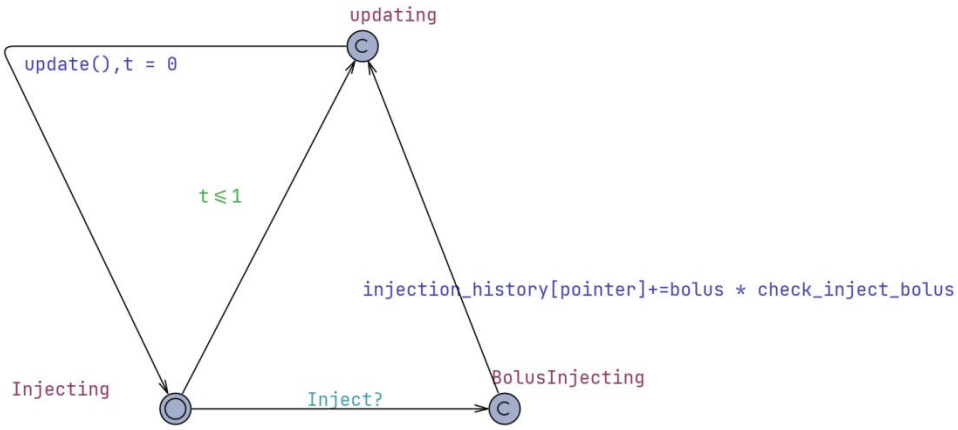
### *T4.3.2: Patient*




The Patient is free to push the inject button.

After the inject button being pushed, a siganl will sent to the processor and the processor will know a request for bolus injection and then handle it.

### *T4.3.3: Processor*




The processor has three major states: Injecting, BolusInjecting and updating.

In the Injecting, the processor will update data over time and handle the injecting request. The processor will leave it when time move on or receiving a injection request.

In the BolusInjecting state, the processor will update the inject_history corresponding to whether the request is valid and leave it immediately to update other information.

In the udating state, the processor will update baseline injection and  check whether the baseline injection and potential bolus injection is valid for the next time stamp.The processor will leave it once the update is done.

### *T4.3.4: Check Properties*

**T4.3.4.1**

| Property | A[] not deadlock |
|---|---|
| Description | There is no deadlock in the Painkiller Injection System |
| Result | Passed |

**T4.3.4.2**

| Property | A[] Processor.check_inject_bolus imply Processor.last_one_day <= limit_day and Processor.last_one_hour <= limit_hour |
|---|---|
| Description | The processor allows inject bolus only if it does not meet the hour limit and the day limit |
| Result | Passed |

**T4.3.4.3**

| Property | A[] Processor.check_inject_bolus imply remaining >= bolus |
|---|---|
| Description | The processor allows inject bolus only if there is enough painkiller in the injector |
| Result | Passed |

**T4.3.4.4**

| Property | A[] Processor.check_inject_baseline imply Processor.last_one_day <= limit_day and Processor.last_one_hour <= limit_hour |
|---|---|
| Description | The processor allows inject baseline only if it does not meet the hour limit and the day limit |
| Result | Passed |

**T4.3.4.5**

| Property | A[] 0<= remaining <=capacity |
|---|---|
| Description | The remaining of  painkiller must be within the limit (make sense for an actual physical device |
| Result | Passed |