Group 7

Author: Xinyue Hu

Software Validation

Painkiller Injection System

Table of Contents

[System Architecture 4](#_Toc1354453439)

[T1: Unit Test 4](#_Toc972291073)

[T1.1: InjectProcessor Unit Test 4](#_Toc1092993888)

[T1.1.1: Test checkInject( ) 4](#_Toc373159178)

[T1.1.2: Test checkDay( ) 5](#_Toc1034055406)

[T1.1.3: Test checkHour( ) 6](#_Toc1931537118)

[T1.1.4: Test checkPin( ) 6](#_Toc1981534174)

[T1.1.5: Test inject\_request( ) 7](#_Toc468464742)

[T1.2: InjectDataBase Unit Test 8](#_Toc811868892)

[T1.2.1: Test getPin ( ) 8](#_Toc2116858453)

[T1.2.2: Test resetPIN ( ) 8](#_Toc678439722)

[T1.2.3: Test getBolus ( ) 8](#_Toc1545725731)

[T1.2.4: Test setBolus ( ) 9](#_Toc902683158)

[T1.2.5: Test getBaseline ( ) 9](#_Toc1571354098)

[T1.2.6: Test setBaseline ( ) 9](#_Toc2141917927)

[T1.2.7: Test getLimitHour ( ) 10](#_Toc946224428)

[T1.2.8: Test setLimitHour ( ) 10](#_Toc1077555361)

[T1.2.9: Test getLimitDay ( ) 10](#_Toc743357176)

[T1.2.10: Test setLimitDay ( ) 11](#_Toc1691682433)

[T1.2.11: Test getStatus ( ) 11](#_Toc1570648798)

[T1.2.12: Test setStatus ( ) 12](#_Toc1025359062)

[T1.2.13: Test getRemain ( ) 12](#_Toc1800971506)

[T1.2.14: Test inject ( ) 12](#_Toc146096877)

[T1.2.15: Test addPainkiller ( ) 13](#_Toc876403218)

[T1.2.16: Test getCapacity ( ) 13](#_Toc118550153)

[T1.2.17: Test getBolusHistory ( ) 13](#_Toc1755080702)

[T1.2.18: Test addBolusHistory ( ) 14](#_Toc1953466969)

[T1.2.19: Test getEventHistory ( ) 14](#_Toc1189352647)

[T1.2.20: Test addEventHistory ( ) 15](#_Toc672151853)

[T1.2.21: Test getBaselineHistory ( ) 15](#_Toc1092210151)

[T1.2.22: Test addBaselineHistory ( ) 15](#_Toc85793301)

[T1.2.23: Test getHour ( ) 16](#_Toc966482770)

[T1.2.24: Test getHour ( ) 16](#_Toc109609482)

[T1.3: MainBoard Unit Test 16](#_Toc1813078495)

[T1.3.1: Test setSetting ( ) 16](#_Toc1764798182)

[T1.3.2: Test setStatus ( ) 17](#_Toc2066396157)

[T1.3.3: Test query ( ) 17](#_Toc814671415)

[T1.3.4: Test resetPIN ( ) 18](#_Toc1974222280)

[T1.3.5: Test addPainkiller ( ) 18](#_Toc2131513810)

[T1.3.6: Test update\_time ( ) 19](#_Toc30405416)

[T1.4: CheckPINUI Unit Test 21](#_Toc2070202373)

[T1.4.1: Test Confirm ( ) 21](#_Toc361234317)

[T1.5: InjectUI Unit Test 22](#_Toc328895750)

[T1.5.1: Test requestInject ( ) 22](#_Toc127962872)

[T1.6: ResetPINUI Unit Test 22](#_Toc1040859057)

[T1.6.1: Test confirm ( ) 22](#_Toc316382537)

[T1.7: SettingUI Unit Test 23](#_Toc271789387)

[T1.7.1: Test updateDatabase ( ) 23](#_Toc266510140)

[T2: Integration Test 25](#_Toc1732518985)

[T2.1: InjectionProcessor + InjectDataBase + MainboardUI Integration 25](#_Toc715811222)

[T2.2: InjectionProcessor + InjectDataBase + MainboardUI + SettingUI Integration 25](#_Toc435817660)

[T2.3: InjectionProcessor + InjectDataBase + MainboardUI + InjectUI Integration 26](#_Toc1868175350)

[T3: Functional Test 27](#_Toc64704663)

[T3.1: Use Case “Get information” 27](#_Toc864545659)

[T3.2: Use Case “Get dynamic data” 28](#_Toc544535211)

[T3.3: Use Case “Set parameter” 28](#_Toc1575471410)

[T3.4: Use Case “Inject” 28](#_Toc474620360)

[T4: Model Checking 28](#_Toc1190125562)

[T4.1: Introduction 28](#_Toc777632376)

[T4.2: Assumptions 28](#_Toc81867790)

[T4.3: Painkiller Injection System Model 29](#_Toc1562412450)

[T4.3.1: Physician 29](#_Toc36011634)

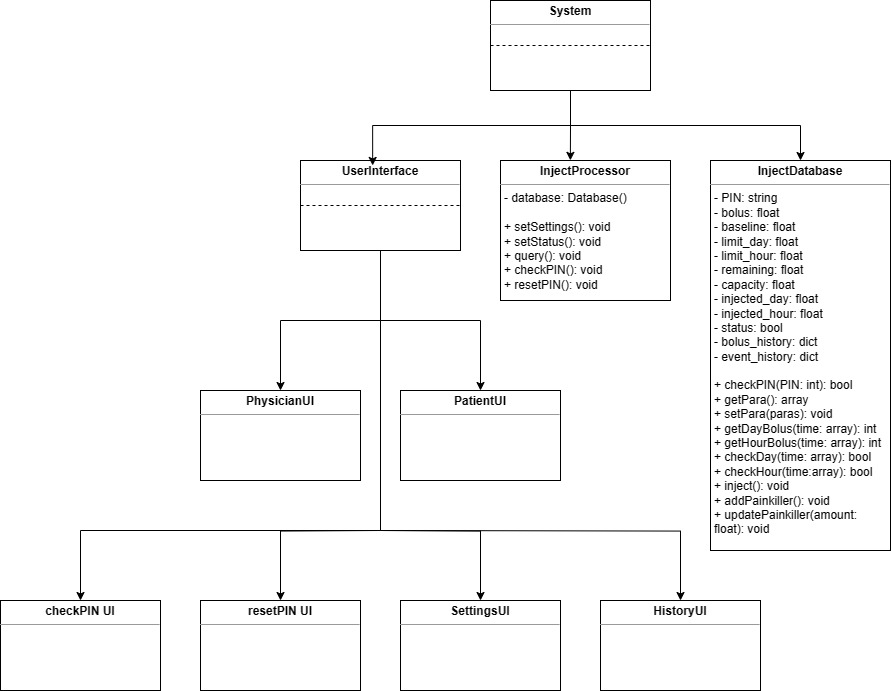
[T4.3.2: Patient 29](#_Toc1804627831)

[T4.3.3: Processor 29](#_Toc1468409036)

[T4.3.4: Check Properties 29](#_Toc668596728)

## 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;  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;  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;  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;  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;  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;  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;  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;  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;  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;  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;  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;  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;  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;  db.bolus = 0.2;  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;  db.capacity = 10.0  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;  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;  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;  db.bolus = 0.2  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;  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;  db.bolus = 0.2  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;  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;  db.baseline = 0;  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;  db.getHourBolus('2024-06-08 12:00:00') = 0  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;  db.getDayBolus('2024-06-08 12:00:00') = 0  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  processor = InjectProcessor  db = InjectDatabase  db.status = True  UI.checkPIN( )=True | UI = Mainboard  processor = InjectProcessor  db = InjectDatabase  db.status = True  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  UI.checkPIN( )=True | UI = Mainboard  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( )  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  UI.getStatus( )=False  UI.getRemain( ) = 0  UI.last\_time = None | UI = Mainboard  UI.getStatus( )=True  UI.getRemain( ) = 10  UI.last\_time = '2024-06-08 12:00:00' |
| Expected Output | limit remain unchanged  warning for add painkiller showed  Last injection: No injection yet! | limit changed  no warning for add painkiller showed  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  processor = InjectProcessor  db=processor.database;  db.Pin = “000000” | Dialog = CheckPINPage  processor = InjectProcessor  db=processor.database;  db.Pin = “000000”  time = 3 | Dialog = CheckPINPage  processor = InjectProcessor  db=processor.database;  db.Pin = “000000”  time = 1 |
| Expected Output | Dialog accepted | time = 2  Dialog returned | time = 0  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( )  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”  password2 = “123456” | password1 = “1234a”  password2 = “123456” | password1 = “123456”  password2 = “654321” |
| State | Dialog = ResetPINPage  processor = InjectProcessor  db=processor.database;  db.Pin = “000000” | Dialog = ResetPINPage  processor = InjectProcessor  db=processor.database;  db.Pin = “000000” | Dialog = ResetPINPage  processor = InjectProcessor  db=processor.database;  db.Pin = “000000” |
| Expected Output | Dialog accepted  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,  bolus = 0.3,  limit\_hour = 0.7,  limit\_day = 1.5,  checkPin() = True | baseline = 0.05,  bolus = 0.3,  limit\_hour = 0.7,  limit\_day = 1.5,  checkPin() = False | baseline = 0.2, |
| State | Dialog = SettingPage  processor = InjectProcessor  db=processor.database;  db.baseline = 0  db.bolus = 0  db.limit\_hour = 1.0  db.limit\_day = 3.0 | Dialog = SettingPage  processor = InjectProcessor  db=processor.database;  db.baseline = 0  db.bolus = 0  db.limit\_hour = 1.0  db.limit\_day = 3.0 | Dialog = SettingPage  processor = InjectProcessor  db=processor.database;  db.baseline = 0  db.bolus = 0  db.limit\_hour = 1.0  db.limit\_day = 3.0 |
| Expected Output | Dialog accepted  db.baseline = 0.05  db.bolus = 0.3  db.limit\_hour = 0.7  db.limit\_day = 1.5 | Dialog rejected  db.baseline = 0  db.bolus = 0  db.limit\_hour = 1.0  db.limit\_day = 3.0 | Dialog returned  db.baseline = 0  db.bolus = 0  db.limit\_hour = 1.0  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  Tcover1.1.2.1  Tcover1.1.3.1  Tcover1.2.7.1  Tcover1.2.9.1  Tcover1.2.11.1  Tcover1.2.13.1  Tcover1.2.21.1  Tcover1.2.23.1  Tcover1.2.24.1 |
| Operation | Wait 40s(80min) |
| State | UI = Mainboard( )  baseline = 0.01;  bolus = 0.3  limit\_day = 3;  limit\_hour = 1  system\_time\_speed: actual\_time\_speed = 120:1 |
| Expected Output | Ater 30s(1h), “ last 1 hour ” fixed to 0.60/1.00 mL  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  Tcover1.2.7.1  Tcover1.2.8.1  Tcover1.2.9.1  Tcover1.2.10.1  Tcover1.2.11.1  Tcover1.2.12.1  Tcover1.2.13.1  Tcover1.2.21.1  Tcover1.3.1.1  Tcover1.3.2.2  Tcover1.3.6.1  Tcover1.7.1.1  Tcover1.7.1.3 |
| Operation | Press status button  Press setting button  Set invalid parameter with baseline = 0.0; bolus = 0.0; limit\_hour = 0.0;limit\_day = 0  Set valid parameter with: baseline = 0.02; bolus = 0.3; limit\_hour = 0.5;limit\_day = 2  Confirm with correct Pin  Press status button  Press query button |
| State | UI = Mainboard( )  initial status = True  initial setting:baseline = 0.0; bolus = 0.0; limit\_hour = 1;limit\_day = 3 |
| Expected Behaviour | After pressing status button, baseline is turned off  After press the setting button, the SettingUI pops first;  After entering the invalid parameter, a warning appear  After setting new parameter and confirm with correct pin, the UI update the parameters showed  After pressing status button, baseline is turned on  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  Tcover1.1.2.1  Tcover1.1.3.1  Tcover1.1.3.2  Tcover1.1.5.1  Tcover1.1.5.2  Tcover1.5.1.1  Tcover1.2.10.1  Tcover1.2.11.1  Tcover1.3.5.1  Tcover1.2.15.1  Tcover1.2.23.1  Tcover1.2.24.1 |
| Operation | (patient) press inject\_button intending for inject bolus at 2min  (physician) press add\_button to add painkiller  (patient) press inject\_button intending for inject bolus at 4min  (patient) press inject\_button intending for inject bolus at 6min  (patient) press inject\_button intending for inject bolus at 8min  (patient) press inject\_button intending for inject bolus at 10min |
| State | UI = Mainboard( )  initial status = True  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  After the physician adding the painkiller, the remaining was updated to 10  The following three press of the inject\_button are all success and the information of the last\_hour and last\_day was updated  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( )  initial status = True  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,  set\_bolus@0.30ml/shot,  baseline\_on,  30min request\_bolus,  75min request\_bolus,  baseline\_off,  set\_baseline@0.1ml/min,  baseline\_on, |
| State | UI = Mainboard( )  initial status = True  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;  Enter Wrong Pin;  Enter Correct Pin;  Type correct form of new Pin and Confirm  (new Pin = “123456”)  Press Setting button  Set invalid parameter with:  baseline = 0.00; bolus = 0.00;  Set valid parameter with:  baseline = 0.01; bolus = 0.30;  Confirm with initial Pin(“000000”)  Confirm with new Pin(“123456”)  Press query(history) button |
| State | UI = Mainboard( )  initial Pin = “000000”  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;  After entering the wrong Pin, the checkPinUI clear the input and request a pin again.  After entering the correct Pin, the ResetPinUI pops out  After typing correct form of new Pin and Confirm, a messagebox will pop out to inform you the success  The pressing of the setting button will pop out the SettingUI  A invalid paramenter will cause a warning messagebox  When you enter a valid parameter, the a checkPinUI pops out  The initial Pin will fail since the pin has been changed while the new pin will set the new parameter to the system  After setting the new parameter, the UI update the corresponding information  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  10min request\_bolus  15min request\_bolus  20min request\_bolus  baseline off  78min request\_bolus  86min request\_bolus,  baseline\_on,  220min request\_bolus, |
| State | UI = Mainboard( )  initial status = True  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  The third press after failed since it meets the hour limit  The press at 78min and 86 min success  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 ;

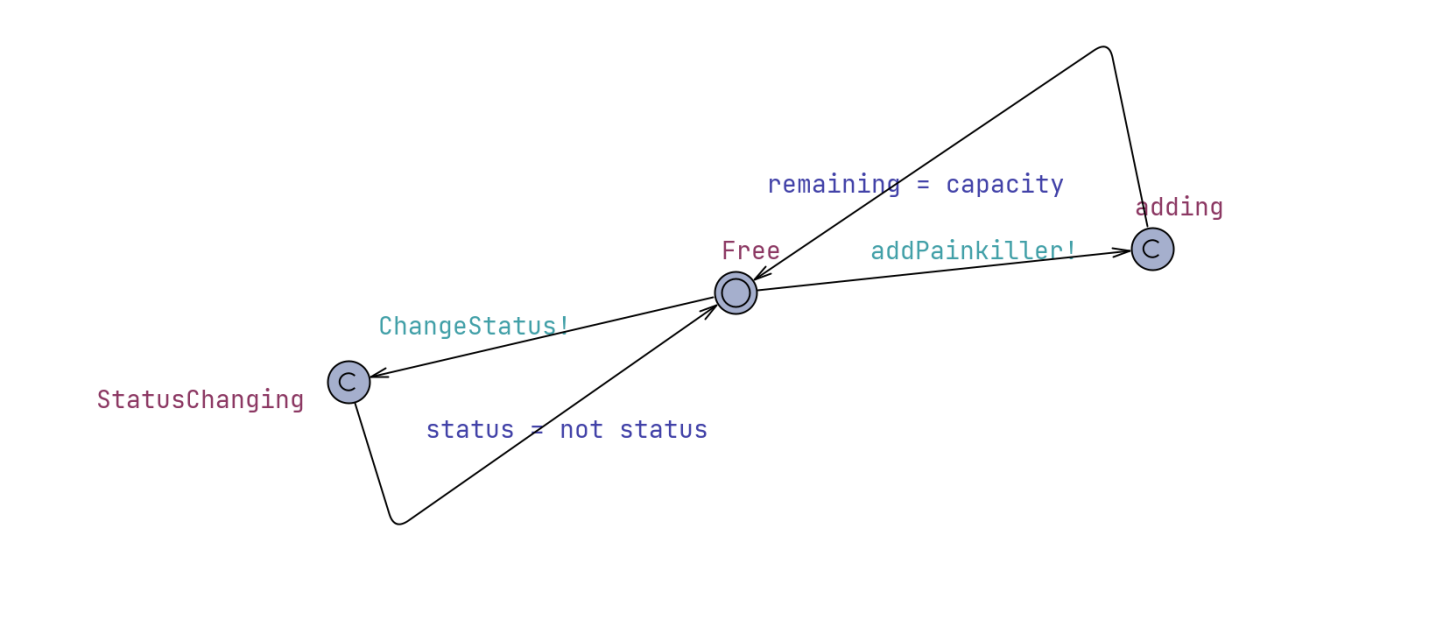
limit\_day: 3ml = 3000μl ;

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

#### 截屏2024-06-22 16.21.18

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

#### 截屏2024-06-22 16.21.50

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 |