CS132: Software Engineering

Validation

Painkiller-Injection-System

Group 15

Author: 贾舜康 祝宇航 王芸飞

Catalogue

1. Unit Test

This section provides information of unit tests we made for every main 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.

1.1 Core

1.1.1 Set Baseline

```
    def set_baseline(self, baseline: float) -> str:
        baseline = Decimal(str(baseline))
        if baseline < Decimal('0.01') or baseline > Decimal('0.1'):
            return "Baseline injection rate must be between 0.01 and 0.1 ml."
        self.__baseline =  Decimal(str(baseline))
        return "Success set baseline to " + str(baseline) + " ml."
```

- Coverage Criteria: Condition Coverage

-TestFunction:tests/unit/test_core.py/test_set_baseline_success+test_set_baseline_failure_big + test_set_baseline_failure_small

- Test Case

Test Case	T1.1.1.1	T1.1.1.2	T1.1.1.3
Input	0.05	0.2	0.005
Coverage Item	Tcover1.1.1.1	Tcover1.1.1.2	Tcover1.1.1.3
State	self.__baseline = 0.05	-----	-----
Expected Output	"Success set baseline to 0.05 ml."	"Baseline injection rate must be between 0.01 and 0.1 ml."	"Baseline injection rate must be between 0.01 and 0.1 ml."
Test Result	Passed	Passed	Passed

- Test Coverage: 3/3 = 100%

1.1.2 Set Bolus

```
    def set_bolus(self, bolus: float) -> str:
        bolus = Decimal(str(bolus))
        if bolus < Decimal('0.2') or bolus > Decimal('0.5'):
            return "Bolus injection amount must be between 0.2 and 0.5 ml."
        self.__bolus = Decimal(str(bolus))
        return "Success set bolus to " + str(bolus) + " ml."
```

- Coverage Criteria: Condition Coverage

- TestFunction: tests/unit/test_core.py/test_set_bolus_success + test_set_bolus_failure_big + test_set_baseline_bolus_small

- Test Case

Test Case	T1.1.1.1	T1.1.1.2	T1.1.1.3

Input	0.3	0.6	0.1
Coverage Item	Tcover1.1.2.1	Tcover1.1.2.2	Tcover1.1.2.3
State	self.__bolus = 0.3	-----	-----
Expected Output	"Success set bolus to 0.3 ml."	"Bolus injection rate must be between 0.2 and 0.5 ml."	"Bolus injection rate must be between 0.2 and 0.5 ml."
Test Result	Passed	Passed	Passed

- Test Coverage: 3/3 = 100%

1.1.3 Baseline on

```
def baseline_on(self):
    self.__baselineStatus = 'on'
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_core.py/test_baseline_on
- Test Case

Test Case	T1.1.3.1
Input	-----
Coverage Item	Tcover1.1.3.1
State	self.__baselineStatus = ’on’
Expected Output	-----
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.1.4 Baseline off

```
def baseline_off(self):
    self.__baselineStatus = 'on'
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_core.py/test_baseline_off
- Test Case

Test Case	T1.1.4.1
Input	-----
Coverage Item	Tcover1.1.4.1
State	self.__baselineStatus = ’off’
Expected Output	-----
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.1.5 Validate

```
def validate(self, amount: Decimal) -> bool:
    # Check hour limit
```

```
        if (Decimal(self.__hourAmount) + amount > Core.MAX_HOUR_AMOUNT):
            return False

      # Check day limit
        if (Decimal(self.__dailyAmount) + amount > Core.MAX_DAILY_AMOUNT):
            return False
        return True
```

- Coverage Criteria: Condition Coverage

- TestFunction: tests/unit/test_core.py/test_validate_false_1 + test_validate_success + test_validate_false_2

- Test Case

Test Case	T1.1.5.1	T1.1.5.2	T1.1.5.3
Input	1.1	0.1	3.1
Coverage Item	Tcover1.1.5.1	Tcover1.1.5.2	Tcover1.1.5.3
State	-----	-----	------
Expected Output	False	True	False
Test Result	Passed	Passed	Passed

- Test Coverage: 3/3 = 100%

1.1.6 Reset

```
    def reset(self):
        self.__baseline = Decimal('0.01')
        self.__bolus = Decimal('0.2')
        self.__dailyAmount = Decimal('0.0')
        self.__hourAmount = Decimal('0.0')
        self.__baselineStatus = 'off'
        self.__minuteRecord = []
        self.__hourlyRecord = []
        self.__dailyRecord = []
        self.__timeRecord = []
        self.__time = 0
```

- Coverage Criteria: Statement Coverage

- TestFunction: tests/unit/test_core.py/test_reset

- Test Case

Test Case	T1.1.6.1
Input	-----
Coverage Item	Tcover1.1.6.1

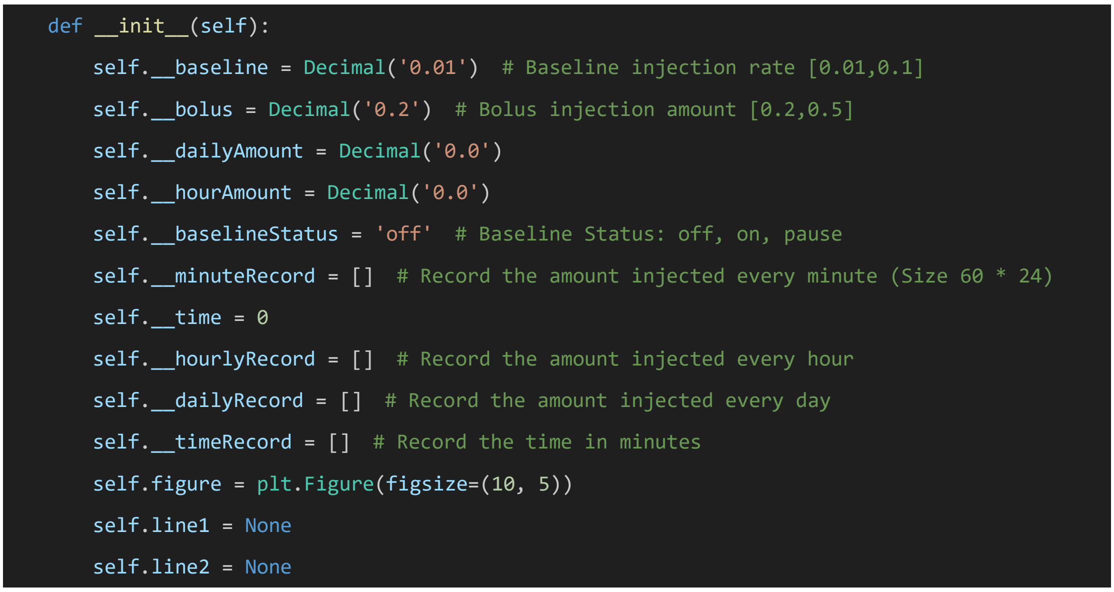
State	self.__baseline = Decimal('0.01')
	self.__bolus = Decimal('0.2')
	self.__dailyAmount = Decimal('0.0')
	self.__hourAmount = Decimal('0.0')
	self.__baselineStatus = 'off'
	self.__minuteRecord = []

	self.__hourlyRecord = []
	self.__dailyRecord = []
	self.__timeRecord = []
	self.__time = 0

Expected Output	-----
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.1.7 Initialize



```
    def __init__(self):
        self.__baseline = Decimal('0.01')  # Baseline injection rate [0.01,0.1]
        self.__bolus = Decimal('0.2')  # Bolus injection amount [0.2,0.5]
        self.__dailyAmount = Decimal('0.0')
        self.__hourAmount = Decimal('0.0')
        self.__baselineStatus = 'off'  # Baseline Status: off, on, pause
        self.__minuteRecord = []  # Record the amount injected every minute (Size 60 * 24)
        self.__time = 0
        self.__hourlyRecord = []  # Record the amount injected every hour
        self.__dailyRecord = []  # Record the amount injected every day
        self.__timeRecord = []  # Record the time in minutes
        self.figure = plt.Figure(figsize=(10, 5))
        self.line1 = None
        self.line2 = None
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_core.py/test_initialize
- Test Case

Test Case	T1.1.7.1
Input	-----
Coverage Item	Tcover1.1.7.1

State	self.__baseline = Decimal('0.01')
	self.__bolus = Decimal('0.2')
	self.__dailyAmount = Decimal('0.0')
	self.__hourAmount = Decimal('0.0')
	self.__baselineStatus = 'off'
	self.__minuteRecord = []
	self.__hourlyRecord = []
	self.__dailyRecord = []
	self.__timeRecord = []
	self.__time = 0
	self.line1 = None
	self.line2 = None

Expected Output	-----
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2 Doctor app

1.2.1 Set Baseline

```
def set_baseline(self):
    baseline = self.baseline_scale.get()
    message = self.core.set_baseline(baseline)
    if not self.start:
        self.display_realtime_info()
    self.show_message(message)
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_doctorAPP.py/test_set_baseline
- Test Case

Test Case	T1.2.1.1
Input	0.05
Coverage Item	Tcover1.2.1.1
State	self.core.__baseline = 0.05
Expected Output	Success set baseline to 0.05 ml.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.2 Set Bolus

```
def set_bolus(self):
    bolus = self.bolus_scale.get()
    message = self.core.set_bolus(bolus)
    if not self.start:
        self.display_realtime_info()
    self.show_message(message)
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_doctorAPP.py/test_set_bolus
- Test Case

Test Case	T1.2.2.1
Input	0.3
Coverage Item	Tcover1.2.2.1
State	self.core.__bolus = 0.3
Expected Output	Success set bolus to 0.3 ml.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.3 Set Simulate Speed

```
def set_simulate_speed(self):
    multiplier = self.speed_scale.get()
    self.simulate_speed = int(1000 / multiplier)
```

```
        self.show_message(f"Simulation speed set to {multiplier}x.")
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_doctorAPP.py/test_set_simulate_speed
- Test Case

Test Case	T1.2.3.1
Input	3
Coverage Item	Tcover1.2.3.1
State	self.simulate_speed = 333
Expected Output	Success set baseline to 3x.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.4 Baseline on

```
    def baseline_on(self):
        self.core.baseline_on()
        if not self.start:
            self.display_realtime_info()
        self.show_message("Baseline injection turned on.")
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_doctorAPP.py/test_baseline_on
- Test Case

Test Case	T1.2.4.1
Input	-----
Coverage Item	Tcover1.2.4.1
State	self.core.__baselineStatus = 'on'
Expected Output	Baseline injection turned on.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.5 Baseline off

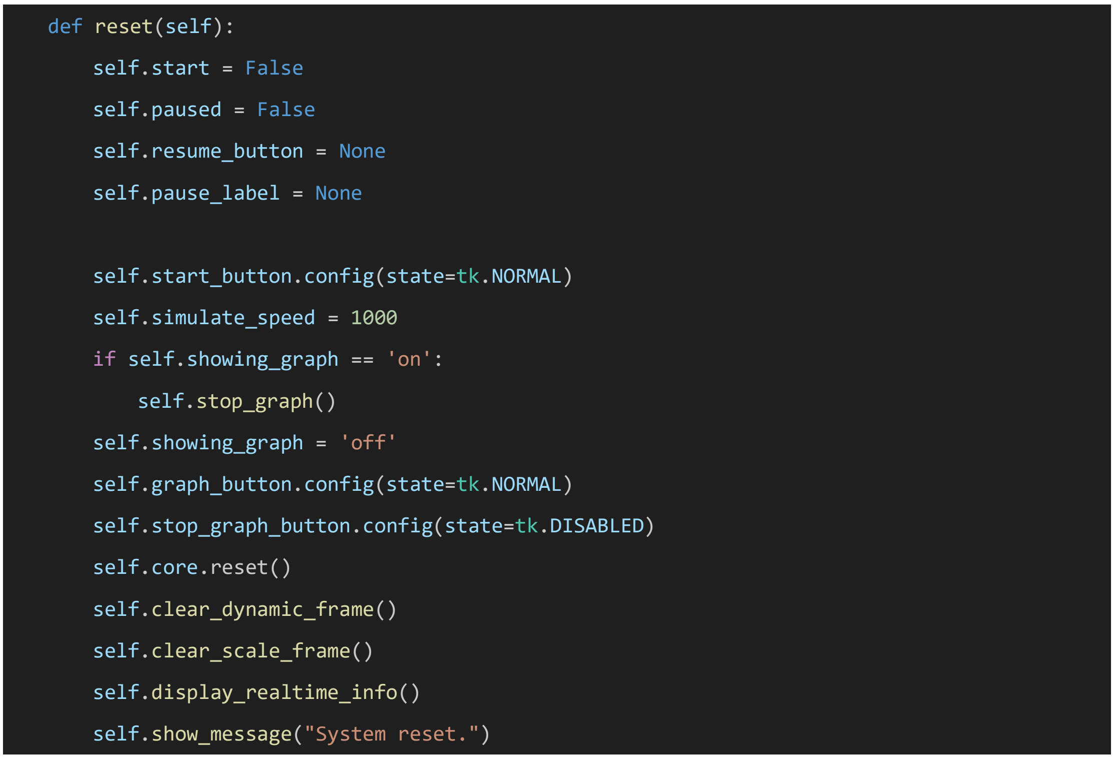
```
    def baseline_off(self):
        self.core.baseline_off()
        if not self.start:
            self.display_realtime_info()
        self.show_message("Baseline injection turned off.")
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_doctorAPP.py/test_baseline_off
- Test Case

Test Case	T1.2.5.1
Input	-----
Coverage Item	Tcover1.2.5.1
State	self.core.__baselineStatus = 'off'
Expected Output	Baseline injection turned off.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.6 Reset



```
def reset(self):
    self.start = False
    self.paused = False
    self.resume_button = None
    self.pause_label = None

    self.start_button.config(state=tk.NORMAL)
    self.simulate_speed = 1000
    if self.showing_graph == 'on':
        self.stop_graph()
    self.showing_graph = 'off'
    self.graph_button.config(state=tk.NORMAL)
    self.stop_graph_button.config(state=tk.DISABLED)
    self.core.reset()
    self.clear_dynamic_frame()
    self.clear_scale_frame()
    self.display_realtime_info()
    self.show_message("System reset.")
```

- Coverage Criteria: Condition Coverage
- TestFunction: tests/unit/test_doctorAPP.py/test_reset_on + test_reset_off
- Test Case

Test Case	T1.2.6.1	T1.2.6.2
Input	-----	-----
Coverage Item	Tcover1.2.6.1	Tcover1.2.6.2

State	self.start = False
	self.paused = False
	self.resume_button = None
	self.pause_label = None
	self.simulate_speed = 1000
	self.showing_graph = 'off'
	self.core.__baseline = Decimal('0.01')
	self.core.__bolus = Decimal('0.2')
	self.core__dailyAmount = Decimal('0.0')
	self.core__hourAmount = Decimal('0.0')
	self.core__baselineStatus = 'off'
	self.core__minuteRecord = []
	self.core__hourlyRecord = []
		self.start = False
		self.paused = False
		self.resume_button = None
		self.pause_label = None
		self.simulate_speed = 1000
		self.showing_graph = 'off'
		self.core.__baseline = Decimal('0.01')
		self.core.__bolus = Decimal('0.2')
		self.core__dailyAmount = Decimal('0.0')
		self.core__hourAmount = Decimal('0.0')
		self.core__baselineStatus = 'off'
		self.core__minuteRecord = []
		self.core__hourlyRecord = []

	self.core__dailyRecord = []
	self.core__timeRecord = []
	self.core__time = 0
		self.core__dailyRecord = []
		self.core__timeRecord = []
		self.core__time = 0

Expected Output	Graph stopped. System reset.	System reset.
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

1.2.7 Pause

```
    def pause(self):
        self.paused = True
        self.disable_buttons()
        if self.showing_graph == 'on':
            self.showing_graph = 'pause'  # Stop updating the graph when paused
        self.clear_scale_frame()
        self.pause_label = tk.Label(self.scale_frame, text="Simulation is paused. Press the \"Resume\" button to restart the system.", wraplength=400, justify=tk.LEFT)
        self.pause_label.place(relx=0.5, rely=0.15, anchor=tk.CENTER)
        self.resume_button = tk.Button(self.scale_frame, text="Resume", command=self.resume)
        self.resume_button.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
```

- Coverage Criteria: Condition Coverage

- TestFunction: tests/unit/test_doctorAPP.py/test_pause_resume_on + test_pause_resume_off

- Test Case

Test Case	T1.2.7.1	T1.2.7.2
Input	-----	-----
Coverage Item	Tcover1.2.7.1	Tcover1.2.7.2

State	self.paused = True
	self.showing_graph = 'pause'
		self.paused = True
		self.showing_graph = 'off'

Expected Output	-----	-----
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

1.2.8 Resume

```
def resume(self):
        self.paused = False
        self.enable_buttons()
        self.clear_scale_frame()
        self.show_message("Simulation resumed.")
        self.display_realtime_info()
        if self.showing_graph == 'pause':
            self.showing_graph = 'on'  # Resume updating the graph
            self.graph_button.config(state=tk.DISABLED)
            self.update_graphs()
        elif self.showing_graph == 'off':
            self.stop_graph_button.config(state=tk.DISABLED)
        if self.start:
            self.start_button.config(state=tk.DISABLED)
```

- Coverage Criteria: Condition Coverage

- TestFunction: tests/unit/test_doctorAPP.py/test_pause_resume_on + test_pause_resume_off

- Test Case

Test Case	T1.2.7.1	T1.2.7.2
Input	-----	-----
Coverage Item	Tcover1.2.8.1	Tcover1.2.8.2

State	self.paused = False
	self.showing_graph = ‘on’
		self.paused = False
		self.showing_graph = ‘off’

Expected Output	Simulation resumed.	Simulation resumed.
Test Result	Passed	Passed

- Test Coverage: 2/2 = 100%

1.2.9 Start Simulate

```
    def start_simulate(self):
        # Update every second
        self.start = True
        self.display_realtime_info()
        if self.showing_graph == 'on':
            self.update_graphs()
        self.start_button.config(state=tk.DISABLED)
        self.show_message("Simulation started.")
```

- Coverage Criteria: Statement Coverage

- TestFunction: tests/unit/test_doctorAPP.py/test_start_simulate

- Test Case

Test Case	T1.2.9.1
Input	-----
Coverage Item	Tcover1.2.9.1
State	self.start = True
Expected Output	Simulation started.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.10 Show Graph

```
    def show_graph(self):
        self.showing_graph = 'on'
        self.stop_graph_button.config(state=tk.NORMAL)
        self.graph_button.config(state=tk.DISABLED)
        self.update_graphs()
```

- Coverage Criteria: Statement Coverage

- TestFunction: tests/unit/test_doctorAPP.py/test_show_graph

- Test Case

Test Case	T1.2.10.1
Input	-----
Coverage Item	Tcover1.2.10.1
State	Self.showing_graph = ‘on’
Expected Output	-----
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.11 Stop Graph

```
def stop_graph(self):
    self.showing_graph = 'off'
    self.graph_button.config(state=tk.NORMAL)
    self.stop_graph_button.config(state=tk.DISABLED)
    if hasattr(self, 'canvas'):
        self.canvas.figure.clf()
        self.core.figure.clf()
        self.core.figure = plt.Figure(figsize=(10, 5))

        self.canvas.get_tk_widget().destroy()
        self.canvas = None
        self.ax1 = None
        self.ax2 = None
    self.show_message("Graph stopped.")
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_doctorAPP.py/test_stop_graph
- Test Case

Test Case	T1.2.11.1
Input	-----
Coverage Item	Tcover1.2.11.1

State	self.showing_graph = ‘off’
	self.canvas = None
	self.ax1 = None
	self.ax2 = None

Expected Output	Graph stopped.
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.2.12 Initialize

```
def __init__(self, root, core):
    self.core = core
    self.root = root
    self.root.title("Doctor Interface")
    self.root.geometry('1400x950+0+0')  # Adjusted size to provide more space
```

```
        # Create title bar for dragging with larger height
        self.title_bar = tk.Frame(root, bg="lightgrey", relief="raised", bd=2, height=40)
        self.title_bar.pack(fill=tk.X)
        self.title_bar_label = tk.Label(self.title_bar, text="Doctor Interface", bg="lightgrey")
        self.title_bar_label.pack(side=tk.LEFT, padx=10, pady=10)
        self.make_window_draggable(self.title_bar)
```

```
        button_width = 25  # Set a uniform button width
        self.simulate_speed = 1000  # Default simulation speed
```

```
        # Frame for the buttons with border
        self.button_frame = tk.Frame(root, bd=2, relief="groove", width=200, height=700)
        self.button_frame.pack(side=tk.LEFT, fill=tk.Y, expand=True, padx=10, pady=10)
        self.button_frame.pack_propagate(0)
```

```
        # Subframe for doctor-related buttons
        self.doctor_button_frame = tk.Frame(self.button_frame, bd=2, relief="groove", width=200,
height=300)
        doctor_title = tk.Label(self.doctor_button_frame, text="Doctor Controls", bg="lightgrey")
        doctor_title.pack(side=tk.TOP, fill=tk.X)
        self.doctor_button_frame.pack(side=tk.TOP, fill=tk.Y, expand=False, padx=10, pady=5)
        self.doctor_button_frame.pack_propagate(0)
```

```
        # Subframe for system-related buttons
        self.system_button_frame = tk.Frame(self.button_frame, bd=2, relief="groove", width=200,
height=230)
        system_title = tk.Label(self.system_button_frame, text="System Controls", bg="lightgrey")
        system_title.pack(side=tk.TOP, fill=tk.X)
        self.system_button_frame.pack(side=tk.TOP, fill=tk.Y, expand=False, padx=10, pady=5)
        self.system_button_frame.pack_propagate(0)
```

```
        # Create buttons
        self.set_baseline_button = tk.Button(self.doctor_button_frame, text="Set Baseline",
command=self.show_baseline_scale, width=button_width)
        self.set_baseline_button.pack(pady=5)
        self.set_bolus_button = tk.Button(self.doctor_button_frame, text="Set Bolus",
command=self.show_bolus_scale, width=button_width)
        self.set_bolus_button.pack(pady=5)
        self.baseline_on_button = tk.Button(self.doctor_button_frame, text="Baseline On",
command=self.baseline_on, width=button_width)
        self.baseline_on_button.pack(pady=5)
        self.baseline_off_button = tk.Button(self.doctor_button_frame, text="Baseline Off",
command=self.baseline_off, width=button_width)
```

```
        self.baseline_off_button.pack(pady=5)
        self.graph_button = tk.Button(self.doctor_button_frame, text="Graph",
command=self.show_graph, width=button_width)
        self.graph_button.pack(pady=5)
        self.stop_graph_button = tk.Button(self.doctor_button_frame, text="Stop Graph",
command=self.stop_graph, width=button_width)
        self.stop_graph_button.pack(pady=5)
```

```
        self.start_button = tk.Button(self.system_button_frame, text="Start",
command=self.start_simulate, width=button_width)
        self.start_button.pack(pady=5)
        self.set_simulate_speed_button = tk.Button(self.system_button_frame, text="Set Simulate
Speed", command=self.show_simulate_speed_scale, width=button_width)
        self.set_simulate_speed_button.pack(pady=5)
        self.pause_button = tk.Button(self.system_button_frame, text="Pause", command=self.pause,
width=button_width)
        self.pause_button.pack(pady=5)
        self.reset_button = tk.Button(self.system_button_frame, text="Reset", command=self.reset,
width=button_width)
        self.reset_button.pack(pady=5)
```

```
        # Frame for the right part of the interface
        self.right_frame = tk.Frame(root, bd=2, relief="groove", width=1200, height=950)
        self.right_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10, pady=10)
        self.right_frame.pack_propagate(0)
```

```
        # Frame for text and graph controls
        self.text_frame = tk.Frame(self.right_frame, bd=2, relief="groove", width=1100,
height=250)
        self.text_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=False, padx=10, pady=10)
        self.text_frame.pack_propagate(0)
```

```
        # Frame for the dynamic part of the interface with border
        self.dynamic_frame = tk.Frame(self.text_frame, bd=2, relief="groove", width=500,
height=250)
        self.dynamic_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10, pady=10)
        self.dynamic_frame.pack_propagate(0)
```

```
        # Create a subframe for the scale controls
        self.scale_frame = tk.Frame(self.text_frame, bd=2, relief="groove", width=500, height=250)
        self.scale_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=False, padx=10, pady=10)
        self.scale_frame.pack_propagate(0)
```

```
        # Create a subframe for the graph controls
        self.graph_frame = tk.Frame(self.right_frame, bd=2, relief="groove", width=1200,
height=650)
        self.graph_frame.pack(side=tk.BOTTOM, fill=tk.BOTH, expand=False, padx=10, pady=10)
        self.graph_frame.pack_propagate(0)
        self.resume_button = None
        self.pause_label = None
        self.showing_graph = 'off'  # Graph is not being shown
        self.paused = False
        self.start = False
        self.canvas = None
        self.ax1 = None
        self.ax2 = None
        self.stop_graph_button.config(state=tk.DISABLED)
        self.display_realtime_info()
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_doctorAPP.py/test_initialize
- Test Case

Test Case	T1.2.12.1
Input	-----
Coverage Item	Tcover1.2.12.1

State	self.resume_button = None
	self.pause_label = None
	self.showing_graph = 'off'
	self.paused = False
	self.start = False
	self.canvas = None
	self.ax1 = None
	self.ax2 = None
	self.simulate_speed = 1000

Expected Output	------
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.3 Patient app

1.3.1 Request Bolus

```
      def request_bolus(self):
        return self.core.request_bolus()
```

- Coverage Criteria: Statement Coverage
- TestFunction: tests/unit/test_patientAPP.py/test_request_bolus_validate_success_init
- Test Case

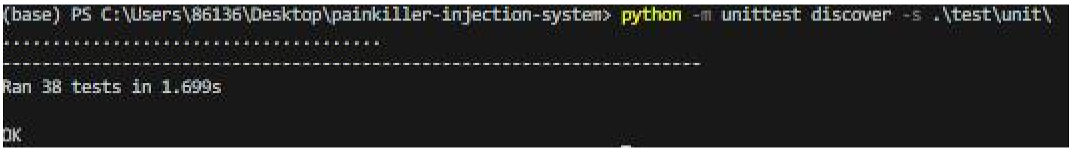
Test Case	T1.3.1.1
Input	-----
Coverage Item	Tcover1.3.1.1

State	-----
Expected Output	-----
Test Result	Passed

- Test Coverage: 1/1 = 100%

1.4 Test Result

You can run `python -m unittest discover -s .\test\unit\` to test all unit tests about main functions in the complementation of Core, DoctorAPP, PatientAPP.



```
(base) PS C:\Users\86136\Desktop\painkiller-injection-system> python -m unittest discover -s .\test\unit\
......................................
----------------------------------------------------------------------
Ran 38 tests in 1.699s

OK
```

2. Integrated Test

This section describes the integration tests we performed for the Painkiller-Injection-System. Since the system has only two components, the integration tests are not complicated. The test cases are provided in each test to run the test functions, which you can do in the corresponding files.

2.1 DoctorAPP + PatientAPP (+ Core)

- TestFunction: tests/Integral/integral_test.py/test_doctor_patient_integration
- Test Case

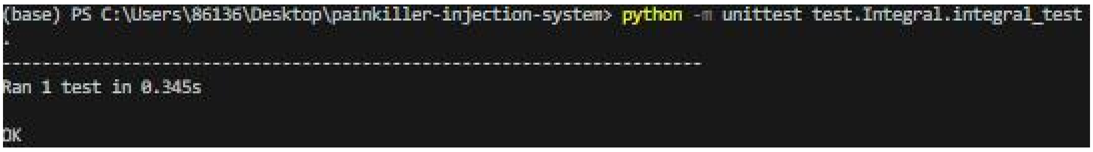
Test Case	T1.1.5.1
Operation	Init the system first. Set baseline rate equal to 0.02 ml in DoctorAPP. Set bolus amount equal to 0.34 ml DoctorAPP. Set baseline on in DoctorAPP. Start simulate in DoctorAPP. Show graph in DoctorAPP. Request bolus in PatientAPP. Update core five times(simulate time passing process in DoctorAPP). Set simulate speed in DoctorAPP. Request bolus in PatientAPP. Update core one time. Click Pause button in DoctorAPP. Click Resume button in DoctorAPP. Stop graph in DoctorAPP. Set baseline off in DoctorAPP. Reset the DoctorAPP system
Coverage Item	Tcover1.1.7.1, Tcover1.2.12.1, Tcover1.2.1.1, Tcover1.1.1.1, Tcover1.2.2.1,

	Tcover1.1.2.1, Tcover1.2.4.1, Tcover1.1.3.1, Tcover1.2.9.1, Tcover1.2.10.1, Tcover1.3.1.1 Tcover1.1.5.2, Tcover1.2.3.1, Tcover1.2.7.1, Tcover1.2.8.1, Tcover1.2.11.1, Tcover1.2.5.1, Tcover1.1.4.1, Tcover1.2.6.1, Tcover1.1.6.1,
Expected Output	-------
Test Result	Passed

- Test Coverage: 20/20 = 100%

2.2 Test Result

You can run `python -m unittest test.Integral.integral_test` to test the integral test about main components and corresponding functions in the complementation of Core, DoctorAPP, PatientAPP.



```
(base) PS C:\Users\86136\Desktop\painkiller-injection-system> python -m unittest test.Integral.integral_test
.
----------------------------------------------------------------------
Ran 1 test in 0.345s

OK
```

3. Functional Test (UI)

This section provides information on the functional(UI) tests we have done for Painkiller-Injection-System as well as common use cases. Each test provides test cases that run the tested functionality, which you can find in the corresponding documentation.

3.1 Simple Combination of Different Buttons

3.1.1 Set Baseline + Set buttons

- Test Function: tests/UI/UI_test.py/test_set_baseline
- Test Case

Test Case	T3.1.1
Operation	Press Set Baseline button first and then set the baseline rate = 0.03, and then press the Set button
State	self.core.__baseline = 0.05
Expected Output	Success set baseline to " + "0.03" + " ml. And the display UI will change to the corresponding value.

Test Result	Passed

3.1.2 Set Bolus + Set buttons

- Test Function: tests/UI/UI_test.py/test_set_bolus
- Test Case

Test Case	T3.1.2
Operation	Press Set Bolus button first and then set the bolus amount = 0.32, and then press the Set button
State	self.core.__bolus = 0.32
Expected Output	Success set bolus to " + "0.32" + " ml. And the display UI will change to the corresponding value.
Test Result	Passed

3.1.3 Graph + Stop Graph buttons

- Test Function: tests/UI/UI_test.py/test_stop_graph
- Test Case

Test Case	T3.1.3
Operation	Press Graph button first and then press Stop Graph button
State	self.app.showing_graph = ‘on’ -> ‘off’
Expected Output	First the graph will appear. And then the graph will disappear with a comment “Graph stopped”.
Test Result	Passed

3.1.4 Set Simulate speed + Set buttons

- Test Function: tests/UI/UI_test.py/test_simulate_speed
- Test Case

Test Case	T3.1.4
Operation	Press Set Simulate Speed button first and then set the simulate speed = 3x, and then press the Set button
State	self.app.showing_graph = ‘on’ -> ‘off’
Expected Output	Simulate speed set to 3x. And the process will speed up.
Test Result	Passed

3.1.5 Graph + Pause + Resume buttons

- Test Function: tests/UI/UI_test.py/test_resume_pause_show
- Test Case

Test Case	T3.1.5
Operation	Press Graph button first and then press the Pause button, and then press the Resume button
State	self.app.showing_graph = ‘on’ -> ‘pause’ -> ‘on’
Expected Output	First the graph will appear and stay stable then. Finally, it will resume with “Simulation resumed”
Test Result	Passed

3.1.6 Start + Pause + Resume buttons

- TestFunction: tests/UI/UI_test.py/test_resume_pause_not_show
- Test Case

Test Case	T3.1.6
Operation	Press Start button first and then press the Pause button, and then press the Resume button
State	self.app.showing_graph = ‘off’ -> ‘off-> ‘off’
Expected Output	Finally, it will resume with “Simulation resumed”, always without graph
Test Result	Passed

3.2 Time processing constrains

3.2.1 Normal start test 1 (strictly within constrains)

- TestFunction: tests/UI/UI_test.py/test_combine_1
- Test Case

Test Case	T3.2.1
Operation	Press Start button. Update 599 times.

Final State	self.core.__time = 600
	self.core.__baseline = 0.01
	self.core.__bolus = 0.2
	self.core.__dailyAmount = 0.0
	self.core.__hourAmount = 0.0
	self.core.__hourlyRecord= [0,0,0,0,...,0]
	self.core.__dailyRecord= [0,0,0,0,...,0]
	self.core.__timeRecord= [1,2,3,4,...,600]

Expected Output	------
Test Result	Passed

3.2.2 Normal start test 2 (strictly within constrains)

- TestFunction: tests/UI/UI_test.py/test_combine_2
- Test Case

Test Case	T3.2.2
Operation	Set baseline rate = 0.05. Set baseline on. Press Start button. Update 69 times.

Final State	self.core.__time = 70
	self.core.__baseline = 0.05
	self.core.__bolus = 0.2
	self.core.__dailyAmount = 1.5
	self.core.__hourAmount = 1.0
	self.core.__hourlyRecord= [0.05,0.05,0.05,0.05,...,0.05]

	self.core.__dailyRecord= [0.05,0.05,0.05,0.05,...,0.05]
	self.core.__timeRecord= [1,2,3,4,...,70]

Expected Output	------
Test Result	Passed

3.2.3 Advanced start test -- Edge test

- TestFunction: tests/UI/UI_test.py/test_combine_4
- Test Case

Test Case	T3.2.3
Operation	Set baseline rate = 0.05. Set bolus amount = 0.30. Set baseline on. Show Graph. Press Start button. Update 59 times. (At 4th time request bolus) Set simulate speed = 2x. Press Pause button. Press Resume button. Stop Graph. Update 60 times. Update 59 times. (At 5th time set baseline off) Press reset button

Final State	self.core.__time = 0
	self.core.__baseline = 0.01
	self.core.__bolus = 0.2
	self.core.__dailyAmount = 0.0
	self.core.__hourAmount = 0.0
	self.core.__hourlyRecord= []
	self.core.__dailyRecord= []
	self.core.__timeRecord= []

Expected Output	------
Test Result	Passed

For detail middle state, you can clearly see from the code.

3.2.4 Simulate Speed test

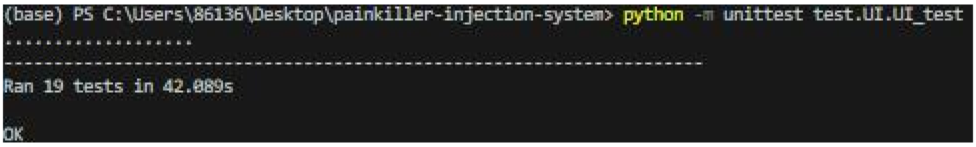
- TestFunction: tests/UI/UI_test.py/test_combine_5
- Test Case

Test Case	T3.2.4
Operation	Set simulate speed = 3x. Press start button. Press reset button. Press start button.
State	self.app.simulate_speed = 1000 -> 333 -> 1000
Expected Output	------

Test Result	Passed

3.3 Test Results

You can run `python -m unittest test.UI.UI_test` to test functional tests with UI about buttons' click conditions and corresponding constrains during the process.



```
(base) PS C:\Users\86136\Desktop\painkiller-injection-system> python -m unittest test.UI.UI_test
...................
----------------------------------------------------------------------
Ran 19 tests in 42.089s

OK
```

4. Model Checking

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

4.1 Introduction

The Painkiller Injection System is divided into three main modules, a Doctor APP, a Patient APP, and a Process (Controller), the first two send information for the Process to accept for processing. Our model focuses on some comprehensive properties checking of our system.

4.2 Assumption

Due to the limitations of uppaal, we simplified the complex system consisting of three components(fig. 1) that we built at the beginning to a simple system consisting of only doctorAPP and patientAPP. And since select will increase the complexity of the model very much, which will affect the performance of checking properties, we omit this step and set it as a fixed value for model checking, so as to check the global framework and design ideas of our model.

Fig.1 : Process

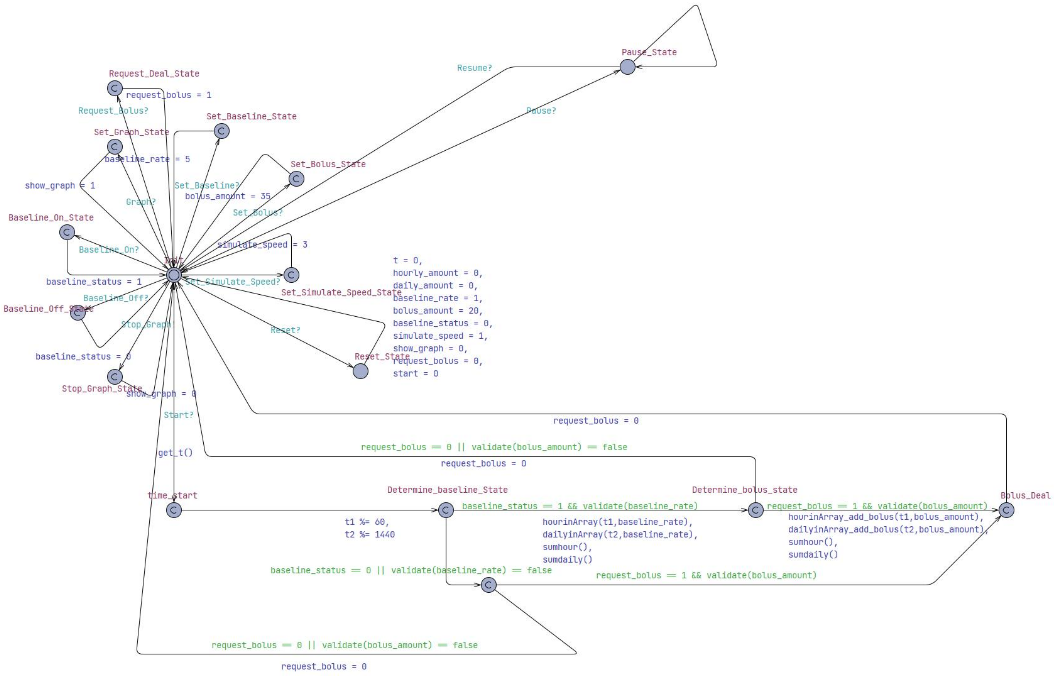
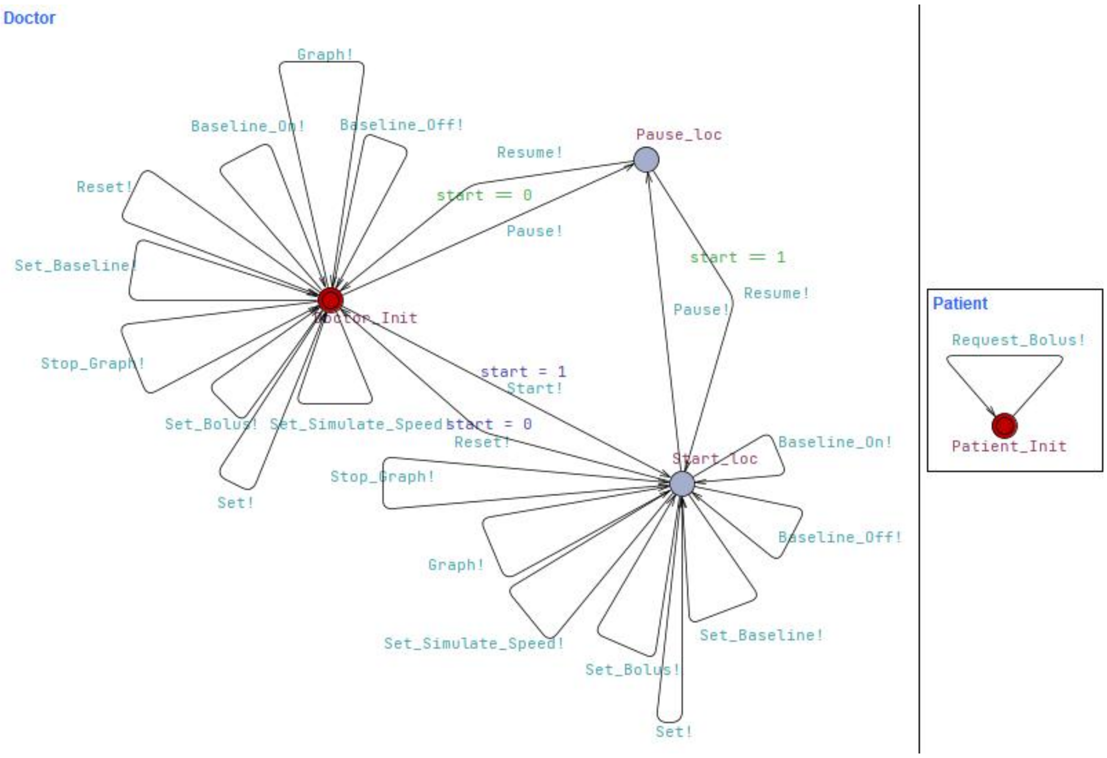
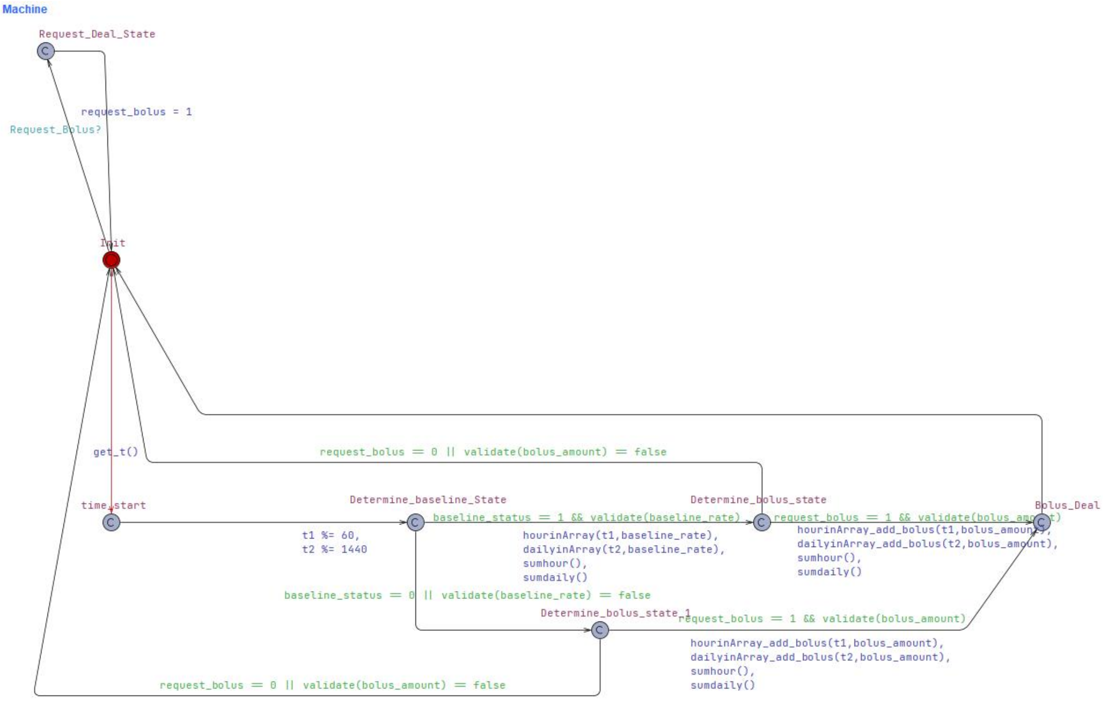


Fig.1 : Doctor_APP + Patient_APP



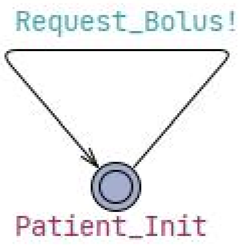
4.3 Painkiller Injection System Model

4.3.1 Machine(Doctor APP)


Machine
Request_Deal_State
request_bolus = 1
Request_Bolus?
Init
get_t()
request_bolus == 0 || validate(bolus_amount) == false
time_start
Determine_baseline_State
baseline_status == 1 && validate(baseline_rate)
Determine_bolus_state
request_bolus == 1 && validate(bolus_amount)
Bolus_Deal
t1 %= 60,
t2 %= 1440
hourinArray(t1,baseline_rate),
dailyinArray(t2,baseline_rate),
sumhour(),
sumdaily()
hourinArray_add_bolus(t1,bolus_amount),
dailyinArray_add_bolus(t2,bolus_amount),
sumhour(),
sumdaily()
baseline_status == 0 || validate(baseline_rate) == false
Determine_bolus_state_1
request_bolus == 1 && validate(bolus_amount)
hourinArray_add_bolus(t1,bolus_amount),
dailyinArray_add_bolus(t2,bolus_amount),
sumhour(),
sumdaily()
request_bolus == 0 || validate(bolus_amount) == false

This is a simplified system model that simulates the normal Painkiller injection process and validates it.

4.3.2 Patient APP



For Patient APP, it only send Request_Bolus message to Machine.

4.3.3 Properties Validation

4.3.3.1 No deadlock

Property	A[] not deadlock
Description	The whole system has no deadlock.
Result	Passed

4.3.3.2 Within index range

Property	A[] Machine.t1 <= 60 && Machine.t2 <= 1440
Description	The variables t1 and t2 are within its own range, so that they can comply with the rule of corresponding vector.
Result	Passed

4.3.3.3 Hourly amount never more than 100(1ml/hour)

Property	A[] Machine.t <= 1440 imply (Machine.hourly_amount >= 0 && Machine.hourly_amount <= 100)
Description	While the system working, no matter how to operate it, the hourly amount will always stay within 100(1ml/hour).
Result	Passed

4.3.3.4 Daily amount never more than 300(3ml/day)

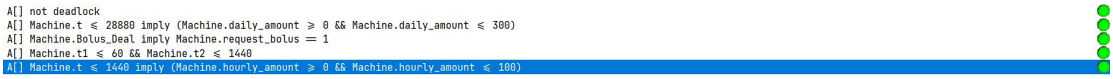
Property	A[] Machine.t <= 28880 imply (Machine.daily_amount >= 0 && Machine.daily_amount <= 300)
Description	While the system working, no matter how to operate it, the daily amount will always stay within 300(3ml/hour).
Result	Passed

4.3.3.5 Only after requesting bolus, reach Machine.Bolus_Deal state

Property	A[] Machine.t <= 28880 imply (Machine.daily_amount >= 0 && Machine.daily_amount <= 300)

Description	While the system working, no matter how to operate it, the daily amount will always stay within 300(3ml/hour).
Result	Passed

Results

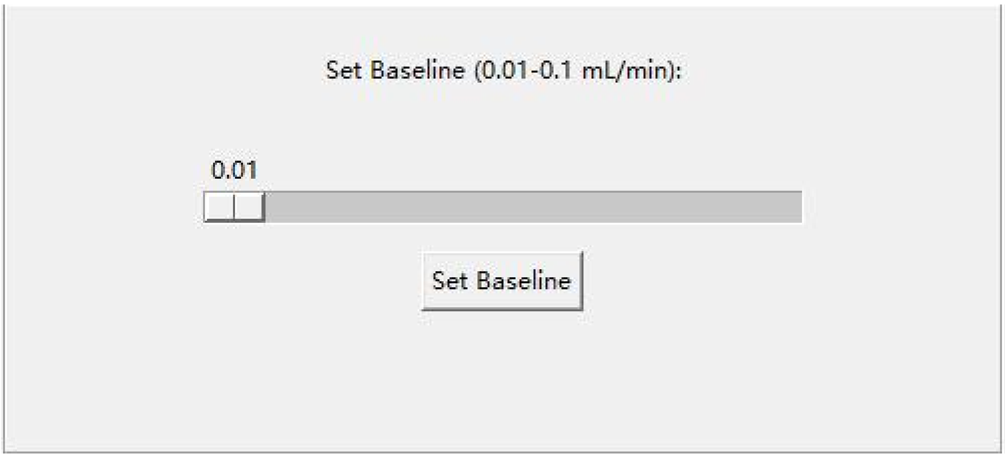


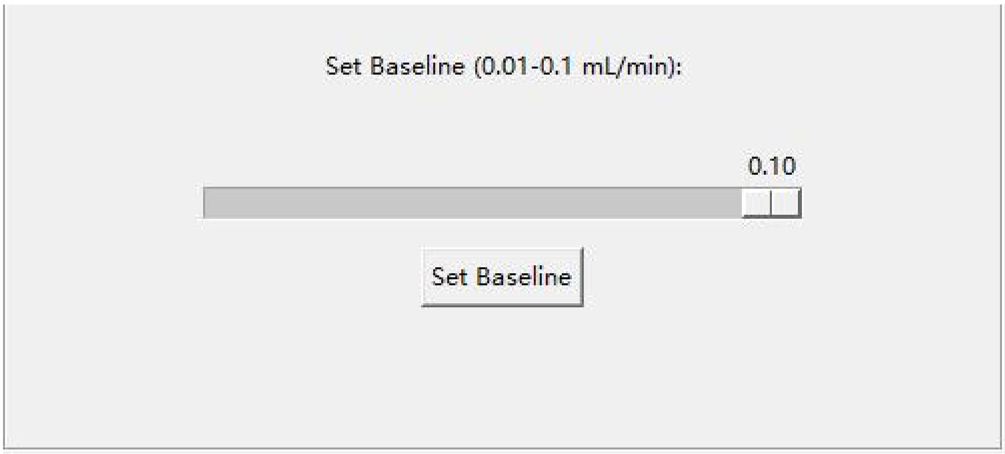
```
A[] not deadlock
A[] Machine.t ⩽ 28880 imply (Machine.daily_amount ⩾ 0 && Machine.daily_amount ⩽ 300)
A[] Machine.Bolus_Deal imply Machine.request_bolus == 1
A[] Machine.t1 ⩽ 60 && Machine.t2 ⩽ 1440
A[] Machine.t ⩽ 1440 imply (Machine.hourly_amount ⩾ 0 && Machine.hourly_amount ⩽ 100)
```

5. Risk Management

5.1.1 Set Baseline

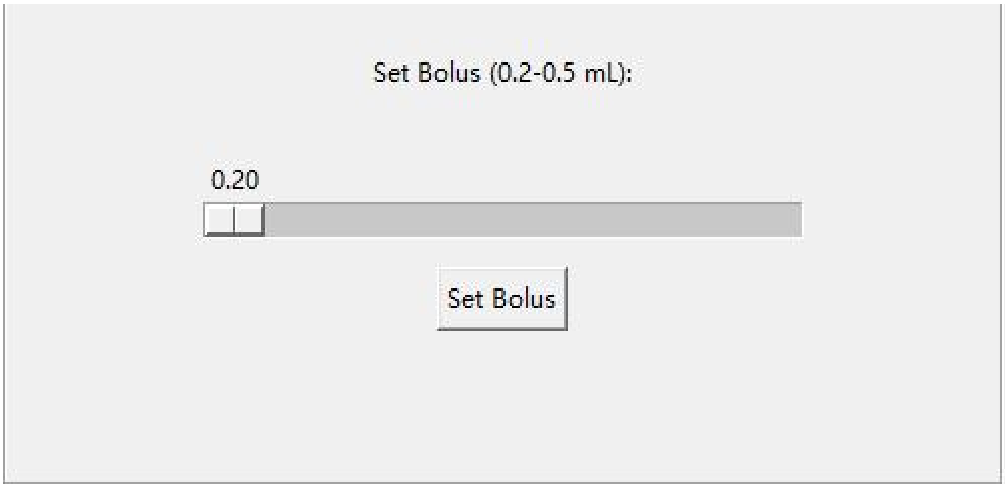
Use the drag bar to implement set operation, and limit the range of the drag bar to avoid the user setting an illegal Baseline, between 0.01 - 0.1.





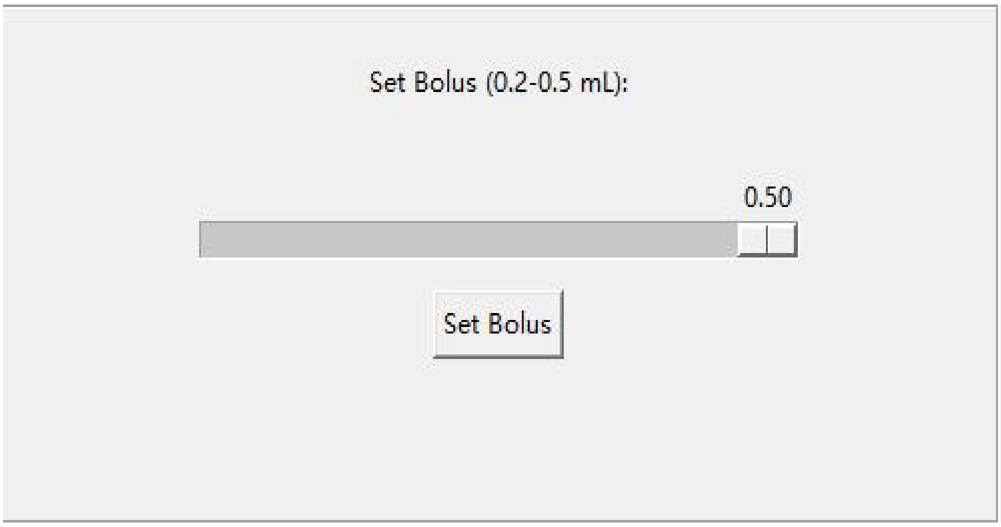
5.1.2 Set Bolus

Use the drag bar to implement set operation, and limit the range of the drag bar to avoid the user setting an illegal Bolus, between 0.2 - 0.5.



Set Bolus (0.2-0.5 mL):

0.20

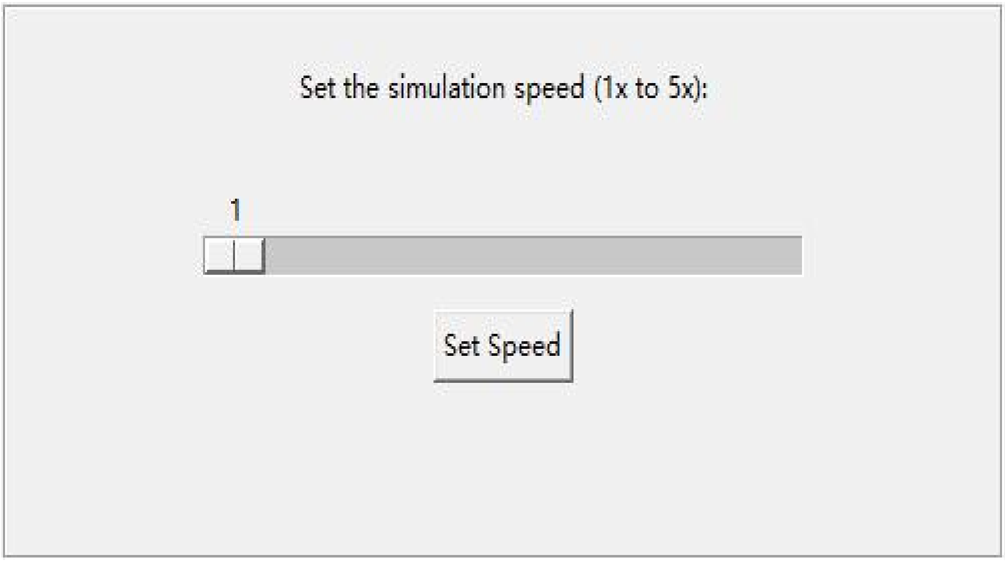


Set Bolus (0.2-0.5 mL):

0.50

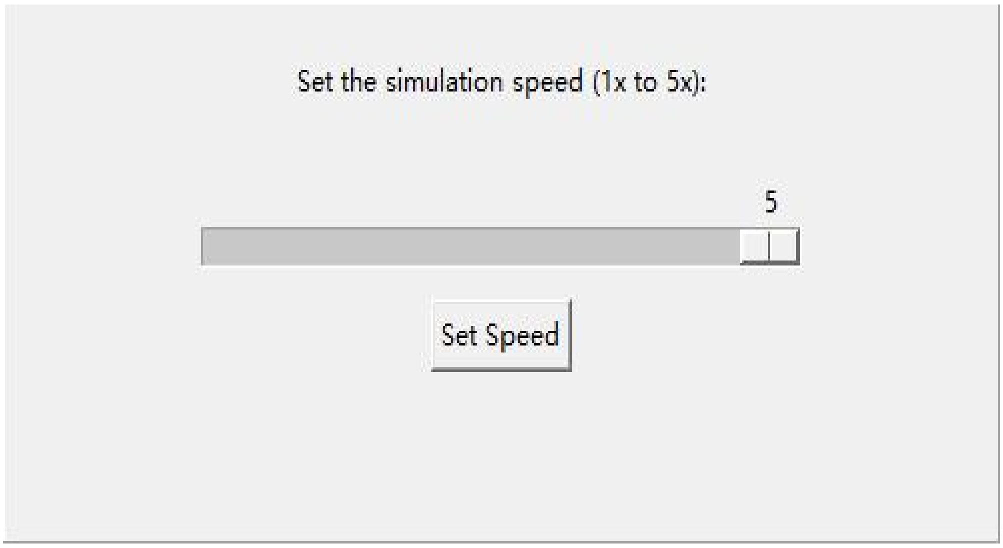
5.1.3 Set Simulate Speed

Use the drag bar to implement set operation, and limit the range of the drag bar to avoid the user setting an illegal Simulate Speed, between 1 - 5.



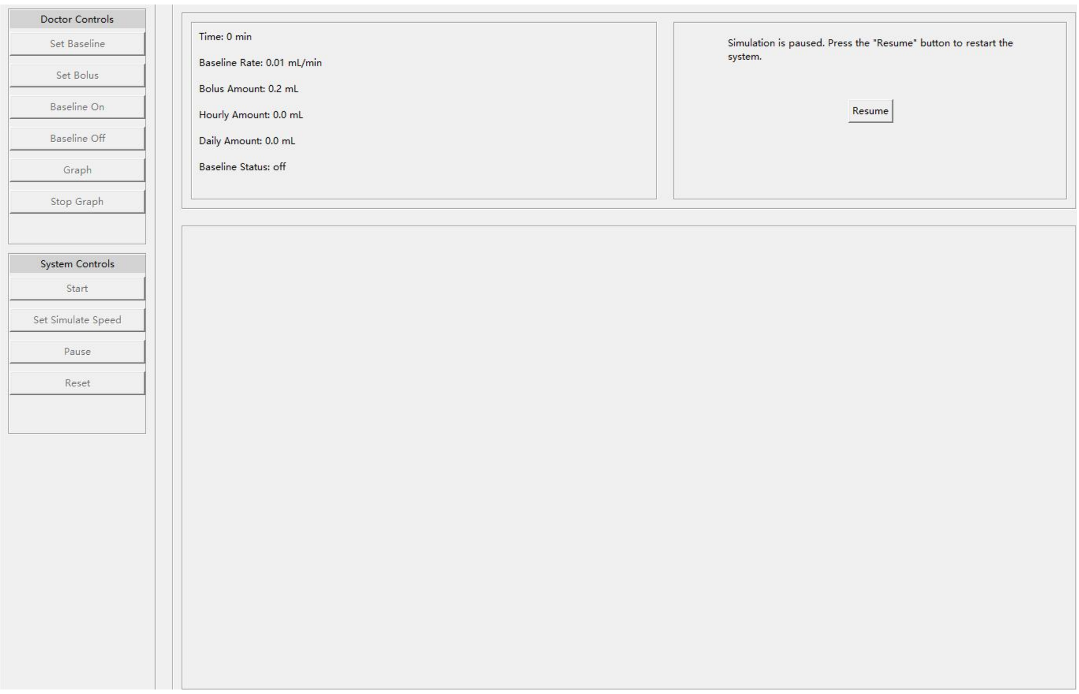
Set the simulation speed (1x to 5x):

1



Set the simulation speed (1x to 5x):

5.1.4 Pause



Time: 0 min

Baseline Rate: 0.01 mL/min

Bolus Amount: 0.2 mL

Hourly Amount: 0.0 mL

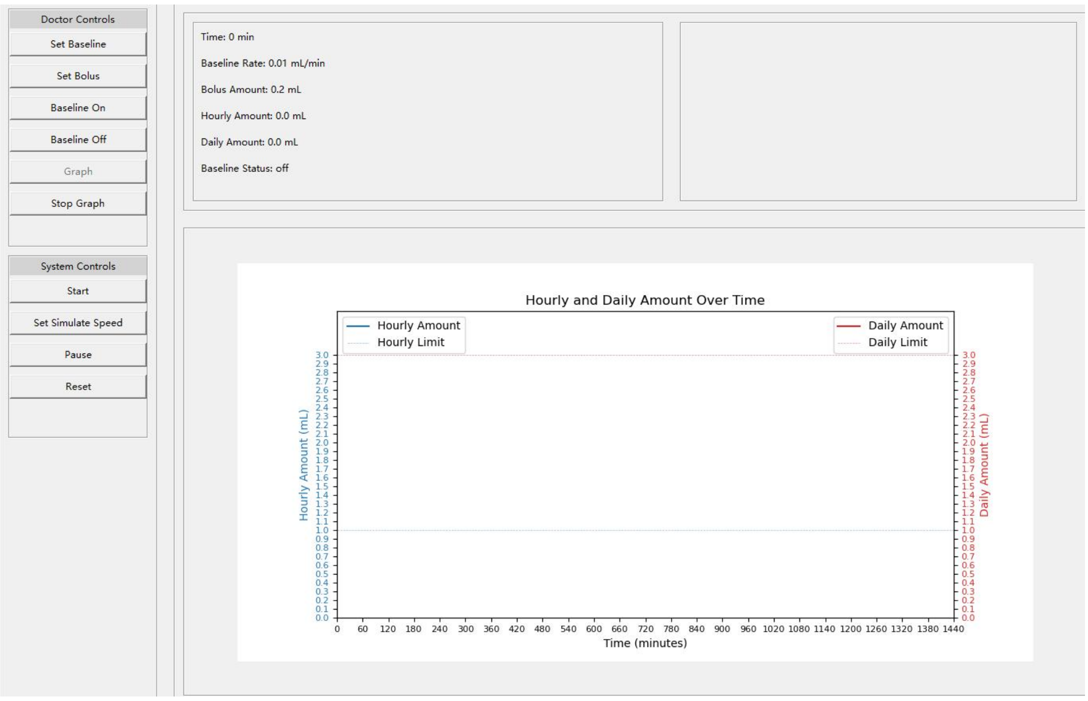
Daily Amount: 0.0 mL

Baseline Status: off

Simulation is paused. Press the "Resume" button to restart the system.

For Pause, if the Pause button is pressed, all other buttons in the system will be disabled and the time will be stopped so that the user can better observe it, and the system will resume when the resume button is pressed.

5.1.5 Graph



Time: 0 min

Baseline Rate: 0.01 mL/min

Bolus Amount: 0.2 mL

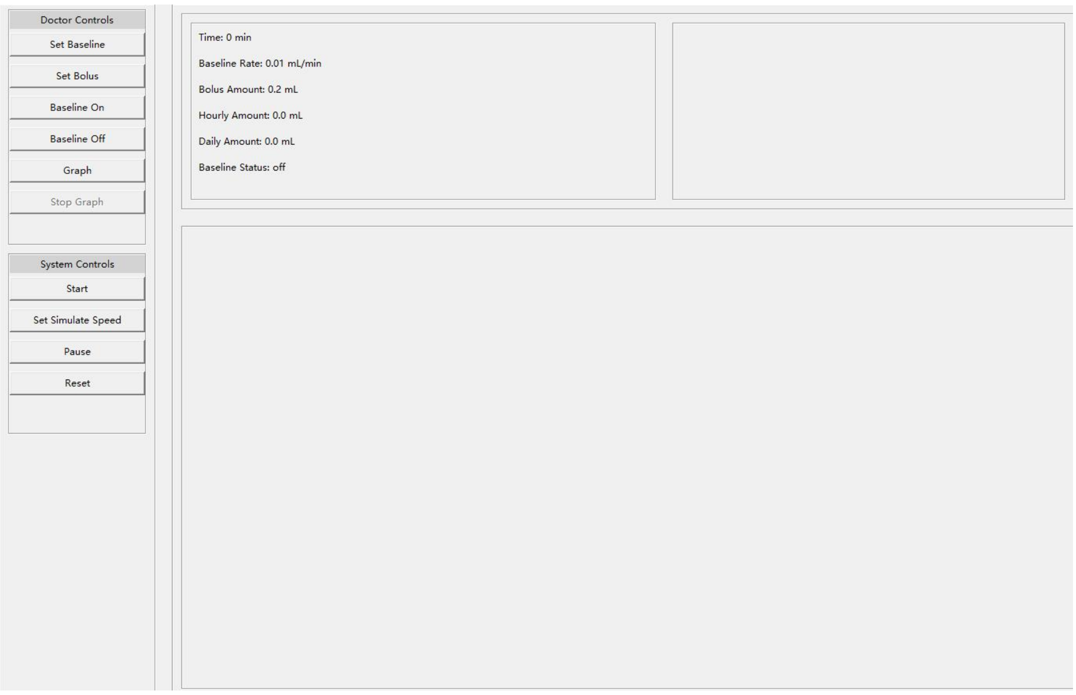
Hourly Amount: 0.0 mL

Daily Amount: 0.0 mL

Baseline Status: off

Pressing the Graph Button disables the button and enables the Stop Graph button to avoid unwanted system display errors.

5.1.6 Stop Graph



Time: 0 min

Baseline Rate: 0.01 mL/min

Bolus Amount: 0.2 mL

Hourly Amount: 0.0 mL

Daily Amount: 0.0 mL

Baseline Status: off

Pressing the Stop Graph button disables the button and enables the Graph button to avoid unwanted system display errors.