Validation Document For Painkiller Injection System

Panxin Tao, Runkang Yang, Zhekai Zhang
SIST, ShanghaiTech University
taopx2022@shanghaitech.edu.cn
yangrk2022@shanghaitech.edu.cn
zhangzhk2022@shanghaitech.edu.cn

June 2024

1 Contents

This document encompasses a comprehensive suite of system verification methodologies to ensure the reliability and correctness of the application. Unit tests are designed to evaluate the basic functionalities and state transitions of individual components, ensuring that each unit of the software performs as expected in isolation. Functional tests are conducted to verify the basic functionalities and the display elements of the user interface, ensuring that the interface behaves correctly and is user-friendly. Integration tests are more extensive, assessing the coordination and interaction between multiple components, such as elevator scheduling, multi-functional interactions, and the seamless execution of complete processes. These tests ensure that the different parts of the system work together harmoniously. Additionally, model checking is utilized to theoretically eliminate potential risks by analyzing the system's behavior under various conditions, providing a higher level of assurance that the system operates correctly and safely.

2 Unit Test

Unit testing broadly refers to testing individual components or units of a software application to ensure they function correctly. Each unit test focuses on a small, isolated piece of code, like a function or method, checking its behavior independently from the rest of the application. This practice helps developers find and fix bugs early, promotes writing modular and testable code, and serves as documentation for expected code behavior. Automating unit tests and integrating them into continuous integration pipelines ensures a stable codebase, quickly catches regressions, and allows confident software modifications.

Here we check the basic functions by checking the states in the code. Display details will be covered in later parts.

Door Open and Close

T1: Auto Door Open

Code:

```
1  def testAutoDoorOpen(self):
2    self.elevators.elevator_state = [IDLE, IDLE]
3    self.elevators.current_floor = [1, 1]
4    self.elevators.elevator_door = [PAUSE, PAUSE]
5    self.elevators.elevator_execute_queue = [[2], []]
6    self.elevators.inside_queue = [[2], []]
7    while (self.elevators.current_floor[0] != 2):
8        self.dispatcher.update()
9
10   self.dispatcher.update()
11   self.assertEqual(self.elevators.elevator_door[0], OPEN)
12   self.dispatcher.update()
13   self.assertEqual(self.elevators.elevator_door[0], HOLD)
14   self.dispatcher.update()
15   self.assertEqual(self.elevators.elevator_door[0], CLOSE)
16   self.assertEqual(len(self.elevators.elevator_execute_queue[0]),
         0)
```

Brief Descriptions: I execute to make an elevator arrive at a certain floor and check the states of that elevator door.

Purposes: When arriving at a certain floor, the elevator's door opens and closes. I test if the corresponding states in the code is correct.

Expected Results: The door state of the program changes from **OPEN** to **HOLD** to **CLOSE**.

Test Results: PASSED.

T2: Valid Door Open

Code:

```
def testValidDoorOpen(self):
  def ClickOpen_(self, elevator):
      if (self.elevators.elevator_state[elevator - 1] == IDLE and
          self.elevators.elevator_door[elevator - 1] == PAUSE) or (
          self.elevators.elevator_door[elevator - 1] == CLOSE):
          self.elevators.animation_open(
              elevator - 1, int(self.elevators.current_floor[
                  elevator - 1]))
  ClickOpen_(self, 1)
  self.assertEqual(self.elevators.elevator_door[0], OPEN)
  self.dispatcher.update()
  self.assertEqual(self.elevators.elevator_door[0], HOLD)
  self.dispatcher.update()
  self.assertEqual(self.elevators.elevator_door[0], CLOSE)
  self.dispatcher.update()
  ClickOpen_(self, 1)
  self.assertEqual(self.elevators.elevator_door[0], OPEN)
```

Brief Descriptions: I click on the open button and check the door states. After it's closed again, click on the open button again. The two opens are both valid.

Purposes: To verify the basic functions of the valid open button.

Expected Results: The door state of the program changes from **OPEN** to **HOLD** to **CLOSE** to **OPEN** again after the scond click.

Test Results: PASSED.

T3: Invalid Door Open

Code:

```
def testInvalidDoorOpen(self):
  def ClickOpen_(self, elevator):
```

```
3         if (self.elevators.elevator_state[elevator - 1] == IDLE and
              self.elevators.elevator_door[elevator - 1] == PAUSE) or (
              self.elevators.elevator_door[elevator - 1] == CLOSE):
4             self.elevators.animation_open(
5                 elevator - 1, int(self.elevators.current_floor[
                      elevator - 1]))
6     self.elevators.elevator_state = [IDLE, IDLE]
7     self.elevators.current_floor = [1, 1]
8     self.elevators.elevator_door = [PAUSE, PAUSE]
9     self.elevators.elevator_execute_queue = [[2], []]
10    self.elevators.inside_queue = [[2], []]
11    self.dispatcher.update()
12    ClickOpen_(self, 1)
13    self.assertEqual(self.elevators.elevator_door[0], PAUSE)
```

Brief Descriptions: First initialize to make an elevator move. When moving, try clicking on the open button.

Purposes: To verify whether the door opens when the click is invalid.

Expected Results: The door state will remain PAUSE after the click.

Test Results: PASSED.

T4: Door Close

Code:

```
1  def testDoorClose(self):
2    def ClickOpen_(self, elevator):
3        if (self.elevators.elevator_state[elevator - 1] == IDLE and
              self.elevators.elevator_door[elevator - 1] == PAUSE) or (
              self.elevators.elevator_door[elevator - 1] == CLOSE):
4             self.elevators.animation_open(
5                 elevator - 1, int(self.elevators.current_floor[
                      elevator - 1]))
6
7    def ClickClose_(self, elevator):
8        if (self.elevators.elevator_door[elevator - 1] == OPEN) or (
              self.elevators.elevator_door[elevator - 1] == HOLD):
9             self.elevators.animation_close(
10                elevator - 1, int(self.elevators.current_floor[
                      elevator - 1]))
11   ClickOpen_(self, 1)
12   self.assertEqual(self.elevators.elevator_door[0], OPEN)
13   self.dispatcher.update()
14   ClickClose_(self, 1)
15   self.assertEqual(self.elevators.elevator_door[0], CLOSE)
```

```
16    self.dispatcher.update()
17    self.assertEqual(self.elevators.elevator_door[0], PAUSE)
```

Brief Descriptions: First click on the open button to make a door open. Then click on the close button to check the states.

Purposes: To verify whether the close button functions correctly.

Expected Results: The door state of the program changes from **OPEN** to **CLOSE** directly and then to **PAUSE**.

Test Results: PASSED.

T5: Inside Other

Code:

```
1  def testInsideButton_other(self):
2    self.elevators.elevator_state = [IDLE, IDLE]
3    self.elevators.current_floor = [1, 1]
4    self.elevators.elevator_door = [PAUSE, PAUSE]
5    self.elevators.elevator_execute_queue = [[], []]
6    self.elevators.inside_queue = [[], []]
7
8    def button_floor_clicked_(self, elevator, floor):
9        button = self.elevators.button_floor[elevator-1][floor]
10       self.elevators.inside_queue[elevator - 1].append(floor)
11       self.elevators.inside_queue[elevator - 1].sort()
12       button.setStyleSheet(
13           "QPushButton{border-image: url(./res/" + str(floor) + "
                 F_pressed.png);}")
14       button.setEnabled(False)
15
16   button_floor_clicked_(self, 1, 0)
17   floor_list = []
18   elevator_state_list = []
19   door_state_list = []
20   cnt = 0
21   while (cnt <= 20):
22       floor_list.append(self.elevators.current_floor[1])
23       elevator_state_list.append(self.elevators.elevator_state[1])
24       door_state_list.append(self.elevators.elevator_door[1])
25       self.dispatcher.update()
26       cnt += 1
27
28   def removerepeat(list_):
29       new_list = list(set(list_))
```

```
30          new_list.sort(key=list_.index)
31          return new_list
32      self.assertEqual(removerepeat(floor_list), [1])
33      self.assertEqual(removerepeat(elevator_state_list), [IDLE])
34      self.assertEqual(removerepeat(door_state_list), [
35                          PAUSE])
```

Brief Descriptions: Click on some inside button of a certain elevator and check whether the states of the other elevator changes.

Purposes: To verify whether the inside buttons influence the other elevator.

Expected Results: The other elevator remains the previous states.

Test Results: PASSED.

T6: Inside 1-B

Code:

```
1  def testInsideBotton_1_b(self):
2      self.elevators.elevator_state = [IDLE, IDLE]
3      self.elevators.current_floor = [1, 1]
4      self.elevators.elevator_door = [PAUSE, PAUSE]
5      self.elevators.elevator_execute_queue = [[], []]
6      self.elevators.inside_queue = [[], []]
7
8      def button_floor_clicked_(self, elevator, floor):
9          button = self.elevators.button_floor[elevator-1][floor]
10         self.elevators.inside_queue[elevator - 1].append(floor)
11         self.elevators.inside_queue[elevator - 1].sort()
12         button.setStyleSheet(
13             "QPushButton{border-image: url(./res/" + str(floor) + "
                   F_pressed.png);}")
14         button.setEnabled(False)
15
16     button_floor_clicked_(self, 1, 0)
17     floor_list = []
18     elevator_state_list = []
19     cnt = 0
20     while (cnt <= 20):
21         floor_list.append(self.elevators.current_floor[0])
22         elevator_state_list.append(self.elevators.elevator_state[0])
23         self.dispatcher.update()
24         cnt += 1
25
```

```
26    def removerepeat(list_):
27        new_list = list(set(list_))
28        new_list.sort(key=list_.index)
29        return new_list
30    self.assertEqual(removerepeat(floor_list), [1, 0.5, 0])
31    self.assertEqual(removerepeat(elevator_state_list), [IDLE, DOWN])
```

Brief Descriptions: Elevators are initially at 1F. Click on the inside B button of Elevator 1 and check the states.

Purposes: To check the functions of inside B button of elevator 1.

Expected Results: Floor from **1** to **0.5** to **0**. Elevator state from **IDLE** to **DOWN**.

Test Results: PASSED.

T7: Inside 1-1

Code:

```
1  def testInsideBotton_1_1(self):
2    self.elevators.elevator_state = [IDLE, IDLE]
3    self.elevators.current_floor = [2, 2]
4    self.elevators.elevator_door = [PAUSE, PAUSE]
5    self.elevators.elevator_execute_queue = [[], []]
6    self.elevators.inside_queue = [[], []]
7
8    def button_floor_clicked_(self, elevator, floor):
9        button = self.elevators.button_floor[elevator-1][floor]
10       self.elevators.inside_queue[elevator - 1].append(floor)
11       self.elevators.inside_queue[elevator - 1].sort()
12       button.setStyleSheet(
13           "QPushButton{border-image: url(./res/" + str(floor) + "
                 F_pressed.png);}")
14       button.setEnabled(False)
15
16   button_floor_clicked_(self, 1, 1)
17   floor_list = []
18   elevator_state_list = []
19   cnt = 0
20   while (cnt <= 20):
21       floor_list.append(self.elevators.current_floor[0])
22       elevator_state_list.append(self.elevators.elevator_state[0])
23       self.dispatcher.update()
24       cnt += 1
25
```

```
26    def removerepeat(list_):
27        new_list = list(set(list_))
28        new_list.sort(key=list_.index)
29        return new_list
30    self.assertEqual(removerepeat(floor_list), [2, 1.5, 1])
31    self.assertEqual(removerepeat(elevator_state_list), [IDLE, DOWN])
```

Brief Descriptions: Elevators are initially at 2F. Click on the inside 1 button of Elevator 1 and check the states.

Purposes: To check the functions of inside 1 button of elevator 1.

Expected Results: Floor from **2** to **1.5** to **1**. Elevator state from **IDLE** to **DOWN**.

Test Results: PASSED.

T8: Inside 1-1-opposite

Code:

```
1  def testInsideBotton_1_1_opposite(self):
2    self.elevators.elevator_state = [IDLE, IDLE]
3    self.elevators.current_floor = [0, 0]
4    self.elevators.elevator_door = [PAUSE, PAUSE]
5    self.elevators.elevator_execute_queue = [[], []]
6    self.elevators.inside_queue = [[], []]
7
8    def button_floor_clicked_(self, elevator, floor):
9        button = self.elevators.button_floor[elevator-1][floor]
10       self.elevators.inside_queue[elevator - 1].append(floor)
11       self.elevators.inside_queue[elevator - 1].sort()
12       button.setStyleSheet(
13           "QPushButton{border-image: url(./res/" + str(floor) + "
                 F_pressed.png);}")
14       button.setEnabled(False)
15
16   button_floor_clicked_(self, 1, 1)
17   floor_list = []
18   elevator_state_list = []
19   cnt = 0
20   while (cnt <= 20):
21       floor_list.append(self.elevators.current_floor[0])
22       elevator_state_list.append(self.elevators.elevator_state[0])
23       self.dispatcher.update()
24       cnt += 1
25
```

```
26    def removerepeat(list_):
27        new_list = list(set(list_))
28        new_list.sort(key=list_.index)
29        return new_list
30    self.assertEqual(removerepeat(floor_list), [0, 0.5, 1])
31    self.assertEqual(removerepeat(elevator_state_list), [IDLE, UP])
```

Brief Descriptions: Elevators are initially at BF. Click on the inside 1 button of Elevator 1 and check the states.

Purposes: To check the functions of inside button functions from the opposite side.

Expected Results: Floor from **0** to **0.5** to **1**. Elevator state from **IDLE** to **UP**.

Test Results: PASSED.

T9: Inside 1-2

Code:

```
1  def testInsideBotton_1_2(self):
2    self.elevators.elevator_state = [IDLE, IDLE]
3    self.elevators.current_floor = [0, 0]
4    self.elevators.elevator_door = [PAUSE, PAUSE]
5    self.elevators.elevator_execute_queue = [[], []]
6    self.elevators.inside_queue = [[], []]
7
8    def button_floor_clicked_(self, elevator, floor):
9        button = self.elevators.button_floor[elevator-1][floor]
10       self.elevators.inside_queue[elevator - 1].append(floor)
11       self.elevators.inside_queue[elevator - 1].sort()
12       button.setStyleSheet(
13           "QPushButton{border-image: url(./res/" + str(floor) + "
                 F_pressed.png);}")
14       button.setEnabled(False)
15
16   button_floor_clicked_(self, 1, 2)
17   floor_list = []
18   elevator_state_list = []
19   cnt = 0
20   while (cnt <= 40):
21       floor_list.append(self.elevators.current_floor[0])
22       elevator_state_list.append(self.elevators.elevator_state[0])
23       self.dispatcher.update()
24       cnt += 1
```

```
25
26    def removerepeat(list_):
27        new_list = list(set(list_))
28        new_list.sort(key=list_.index)
29        return new_list
30    self.assertEqual(removerepeat(floor_list), [0, 0.5, 1, 1.5, 2])
31    self.assertEqual(removerepeat(elevator_state_list), [IDLE, UP])
```

Brief Descriptions: Elevators are initially at BF. Click on the inside 2 button of Elevator 1 and check the states.

Purposes: To check the functions of inside 2 button of elevator 1.

Expected Results: Floor from **0** to **0.5, 1, 1.5** to **2**. Elevator state from **IDLE** to **UP**.

Test Results: PASSED.

T10: Inside 1-3

Code:

```
1  def testInsideBotton_1_3(self):
2    self.elevators.elevator_state = [IDLE, IDLE]
3    self.elevators.current_floor = [0, 0]
4    self.elevators.elevator_door = [PAUSE, PAUSE]
5    self.elevators.elevator_execute_queue = [[], []]
6    self.elevators.inside_queue = [[], []]
7
8    def button_floor_clicked_(self, elevator, floor):
9        button = self.elevators.button_floor[elevator-1][floor]
10       self.elevators.inside_queue[elevator - 1].append(floor)
11       self.elevators.inside_queue[elevator - 1].sort()
12       button.setStyleSheet(
13           "QPushButton{border-image: url(./res/" + str(floor) + "
                 F_pressed.png);}")
14       button.setEnabled(False)
15
16   button_floor_clicked_(self, 1, 3)
17   floor_list = []
18   elevator_state_list = []
19   cnt = 0
20   while (cnt <= 60):
21       floor_list.append(self.elevators.current_floor[0])
22       elevator_state_list.append(self.elevators.elevator_state[0])
23       self.dispatcher.update()
24       cnt += 1
```

```
25
26    def removerepeat(list_):
27        new_list = list(set(list_))
28        new_list.sort(key=list_.index)
29        return new_list
30    self.assertEqual(removerepeat(floor_list), [0, 0.5, 1, 1.5, 2,
          2.5, 3])
31    self.assertEqual(removerepeat(elevator_state_list), [IDLE, UP])
```

Brief Descriptions: Elevators are initially at BF. Click on the inside 3 button of Elevator 1 and check the states.

Purposes: To check the functions of inside 3 button of elevator 1.

Expected Results: Floor from **0** to **0.5, 1, 1.5, 2, 2.5** to **3**. Elevator state from **IDLE** to **UP**.

Test Results: PASSED.

T11: Inside 2

Code:

```
1  def testInsideBotton_2_1(self):
2    self.elevators.elevator_state = [IDLE, IDLE]
3    self.elevators.current_floor = [0, 0]
4    self.elevators.elevator_door = [PAUSE, PAUSE]
5    self.elevators.elevator_execute_queue = [[], []]
6    self.elevators.inside_queue = [[], []]
7
8    def button_floor_clicked_(self, elevator, floor):
9        button = self.elevators.button_floor[elevator-1][floor]
10       self.elevators.inside_queue[elevator - 1].append(floor)
11       self.elevators.inside_queue[elevator - 1].sort()
12       button.setStyleSheet(
13           "QPushButton{border-image: url(./res/" + str(floor) + "
                 F_pressed.png);}")
14       button.setEnabled(False)
15
16   button_floor_clicked_(self, 2, 1)
17   floor_list = []
18   elevator_state_list = []
19   cnt = 0
20   while (cnt <= 20):
21       floor_list.append(self.elevators.current_floor[1])
22       elevator_state_list.append(self.elevators.elevator_state[1])
23       self.dispatcher.update()
```

```
24          cnt += 1
25
26      def removerepeat(list_):
27          new_list = list(set(list_))
28          new_list.sort(key=list_.index)
29          return new_list
30      self.assertEqual(removerepeat(floor_list), [0, 0.5, 1])
31      self.assertEqual(removerepeat(elevator_state_list), [IDLE, UP])
```

Brief Descriptions: Elevators are initially at BF. Click on the inside 1 button of Elevator 2 and check the states.

Purposes: Take button 1 as an example. Check the functions of inside buttons of elevator 2.

Expected Results: Floor from **0** to **0.5** to **1**. Elevator state from **IDLE** to **UP**.

Test Results: PASSED.

T12: Outside B

Code:

```
1  def testOutsideButton_b_1(self):
2      self.elevators.elevator_state = [IDLE, IDLE]
3      self.elevators.current_floor = [0, 1]
4      self.elevators.elevator_door = [PAUSE, PAUSE]
5      self.elevators.elevator_execute_queue = [[], []]
6      self.elevators.inside_queue = [[], []]
7
8      def button_outside_clicked_(self, floor, direction):
9          if direction == 'up':
10             button = self.elevators.button_up[floor if floor > 0 else
                   0]
11             self.elevators.outside_up_queue.append(floor)
12             self.elevators.outside_up_queue.sort()
13             button.setStyleSheet(
14                 "QPushButton{border-image: url(./res/up_pressed.png)
                       ;}")
15             button.setEnabled(False)
16         else:
17             button = self.elevators.button_down[floor-1]
18             self.elevators.outside_down_queue.append(floor)
19             self.elevators.outside_down_queue.sort()
20             button.setStyleSheet(
```

```
21                "QPushButton{border-image: url(./res/down_pressed.png
                      );}")
22            button.setEnabled(False)
23    button_outside_clicked_(self, 0, 'up')
24    floor_list_1 = []
25    elevator_state_list_1 = []
26    floor_list_2 = []
27    elevator_state_list_2 = []
28    cnt = 0
29    while (cnt <= 60):
30        floor_list_1.append(self.elevators.current_floor[0])
31        elevator_state_list_1.append(self.elevators.elevator_state
              [0])
32        floor_list_2.append(self.elevators.current_floor[1])
33        elevator_state_list_2.append(self.elevators.elevator_state
              [1])
34        self.dispatcher.update()
35        cnt += 1
36
37    def removerepeat(list_):
38        new_list = list(set(list_))
39        new_list.sort(key=list_.index)
40        return new_list
41    self.assertEqual(removerepeat(floor_list_1), [0])
42    self.assertEqual(removerepeat(elevator_state_list_1), [IDLE, UP])
43    self.assertEqual(removerepeat(floor_list_2), [1])
44    self.assertEqual(removerepeat(elevator_state_list_2), [IDLE])
45
46
47 def testOutsideButton_b_2(self):
48    self.elevators.elevator_state = [IDLE, IDLE]
49    self.elevators.current_floor = [2, 1]
50    self.elevators.elevator_door = [PAUSE, PAUSE]
51    self.elevators.elevator_execute_queue = [[], []]
52    self.elevators.inside_queue = [[], []]
53
54    def button_outside_clicked_(self, floor, direction):
55        if direction == 'up':
56            button = self.elevators.button_up[floor if floor > 0 else
                  0]
57            self.elevators.outside_up_queue.append(floor)
58            self.elevators.outside_up_queue.sort()
59            button.setStyleSheet(
60                "QPushButton{border-image: url(./res/up_pressed.png)
                      ;}")
61            button.setEnabled(False)
```

```
62          else:
63              button = self.elevators.button_down[floor-1]
64              self.elevators.outside_down_queue.append(floor)
65              self.elevators.outside_down_queue.sort()
66              button.setStyleSheet(
67                  "QPushButton{border-image: url(./res/down_pressed.png
                        );}")
68              button.setEnabled(False)
69      button_outside_clicked_(self, 0, 'up')
70      floor_list_1 = []
71      elevator_state_list_1 = []
72      floor_list_2 = []
73      elevator_state_list_2 = []
74      cnt = 0
75      while (cnt <= 60):
76          floor_list_1.append(self.elevators.current_floor[0])
77          elevator_state_list_1.append(self.elevators.elevator_state
                [0])
78          floor_list_2.append(self.elevators.current_floor[1])
79          elevator_state_list_2.append(self.elevators.elevator_state
                [1])
80          self.dispatcher.update()
81          cnt += 1
82
83      def removerepeat(list_):
84          new_list = list(set(list_))
85          new_list.sort(key=list_.index)
86          return new_list
87      self.assertEqual(removerepeat(floor_list_1), [2])
88      self.assertEqual(removerepeat(elevator_state_list_1), [IDLE])
89      self.assertEqual(removerepeat(floor_list_2), [1, 0.5, 0])
90      self.assertEqual(removerepeat(elevator_state_list_2),
91                       [IDLE, DOWN_UP, UP])
```

Brief Descriptions: Initial floors under the two circumstances are **[0,1]** and **[2,1]**. Click on outside B and check the states.

Purposes: To see which elevator will arrive and send a passenger when outside button B is clicked on under different circumstances.

Expected Results: [0,1]: Elevator 1 to send while **[2,1]**: Elevator 2 to send.

Test Results: PASSED.

T13: Outside 3

Code:

```
def testOutsideButton_3_1(self):
  self.elevators.elevator_state = [IDLE, IDLE]
  self.elevators.current_floor = [3, 2]
  self.elevators.elevator_door = [PAUSE, PAUSE]
  self.elevators.elevator_execute_queue = [[], []]
  self.elevators.inside_queue = [[], []]

  def button_outside_clicked_(self, floor, direction):
      if direction == 'up':
          button = self.elevators.button_up[floor if floor > 0 else
               0]
          self.elevators.outside_up_queue.append(floor)
          self.elevators.outside_up_queue.sort()
          button.setStyleSheet(
              "QPushButton{border-image: url(./res/up_pressed.png)
                  ;}")
          button.setEnabled(False)
      else:
          button = self.elevators.button_down[floor-1]
          self.elevators.outside_down_queue.append(floor)
          self.elevators.outside_down_queue.sort()
          button.setStyleSheet(
              "QPushButton{border-image: url(./res/down_pressed.png
                  );}")
          button.setEnabled(False)
  button_outside_clicked_(self, 3, 'down')
  floor_list_1 = []
  elevator_state_list_1 = []
  floor_list_2 = []
  elevator_state_list_2 = []
  cnt = 0
  while (cnt <= 60):
      floor_list_1.append(self.elevators.current_floor[0])
      elevator_state_list_1.append(self.elevators.elevator_state
          [0])
      floor_list_2.append(self.elevators.current_floor[1])
      elevator_state_list_2.append(self.elevators.elevator_state
          [1])
      self.dispatcher.update()
      cnt += 1

  def removerepeat(list_):
      new_list = list(set(list_))
```

```
39          new_list.sort(key=list_.index)
40          return new_list
41      self.assertEqual(removerepeat(floor_list_1), [3])
42      self.assertEqual(removerepeat(elevator_state_list_1), [IDLE, DOWN
            ])
43      self.assertEqual(removerepeat(floor_list_2), [2])
44      self.assertEqual(removerepeat(elevator_state_list_2), [IDLE])
45
46
47  def testOutsideButton_3_2(self):
48      self.elevators.elevator_state = [IDLE, IDLE]
49      self.elevators.current_floor = [1, 2]
50      self.elevators.elevator_door = [PAUSE, PAUSE]
51      self.elevators.elevator_execute_queue = [[], []]
52      self.elevators.inside_queue = [[], []]
53
54      def button_outside_clicked_(self, floor, direction):
55          if direction == 'up':
56              button = self.elevators.button_up[floor if floor > 0 else
                    0]
57              self.elevators.outside_up_queue.append(floor)
58              self.elevators.outside_up_queue.sort()
59              button.setStyleSheet(
60                  "QPushButton{border-image: url(./res/up_pressed.png)
                        ;}")
61              button.setEnabled(False)
62          else:
63              button = self.elevators.button_down[floor-1]
64              self.elevators.outside_down_queue.append(floor)
65              self.elevators.outside_down_queue.sort()
66              button.setStyleSheet(
67                  "QPushButton{border-image: url(./res/down_pressed.png
                        );}")
68              button.setEnabled(False)
69      button_outside_clicked_(self, 3, 'down')
70      floor_list_1 = []
71      elevator_state_list_1 = []
72      floor_list_2 = []
73      elevator_state_list_2 = []
74      cnt = 0
75      while (cnt <= 60):
76          floor_list_1.append(self.elevators.current_floor[0])
77          elevator_state_list_1.append(self.elevators.elevator_state
                [0])
78          floor_list_2.append(self.elevators.current_floor[1])
79          elevator_state_list_2.append(self.elevators.elevator_state
```

```
            [1])
80          self.dispatcher.update()
81          cnt += 1
82
83      def removerepeat(list_):
84          new_list = list(set(list_))
85          new_list.sort(key=list_.index)
86          return new_list
87      self.assertEqual(removerepeat(floor_list_1), [1])
88      self.assertEqual(removerepeat(elevator_state_list_1), [IDLE])
89      self.assertEqual(removerepeat(floor_list_2), [2, 2.5, 3])
90      self.assertEqual(removerepeat(elevator_state_list_2),
91                       [IDLE, UP_DOWN, DOWN])
92
93      def removerepeat(list_):
94          new_list = list(set(list_))
95          new_list.sort(key=list_.index)
96          return new_list
97      self.assertEqual(removerepeat(floor_list_1), [2])
98      self.assertEqual(removerepeat(elevator_state_list_1), [IDLE])
99      self.assertEqual(removerepeat(floor_list_2), [1, 0.5, 0])
100     self.assertEqual(removerepeat(elevator_state_list_2),
101                      [IDLE, DOWN_UP, UP])
```

Brief Descriptions: Initial floors under the two circumstances are **[3,2]** and **[1,2]**. Click on outside B and check the states.

Purposes: To see which elevator will arrive and send a passenger when outside button 3 is clicked on under different circumstances.

Expected Results: **[3,2]**: Elevator 1 to send while **[1,2]**: Elevator 2 to send.

Test Results: PASSED.

T14: Outside 1-up

Code:

```
1  def testOutsideButton_1_up(self):
2      self.elevators.elevator_state = [IDLE, IDLE]
3      self.elevators.current_floor = [0, 3]
4      self.elevators.elevator_door = [PAUSE, PAUSE]
5      self.elevators.elevator_execute_queue = [[], []]
6      self.elevators.inside_queue = [[], []]
7
8      def button_outside_clicked_(self, floor, direction):
```

```
9         if direction == 'up':
10            button = self.elevators.button_up[floor if floor > 0 else
                  0]
11            self.elevators.outside_up_queue.append(floor)
12            self.elevators.outside_up_queue.sort()
13            button.setStyleSheet(
14                "QPushButton{border-image: url(./res/up_pressed.png)
                      ;}")
15            button.setEnabled(False)
16        else:
17            button = self.elevators.button_down[floor-1]
18            self.elevators.outside_down_queue.append(floor)
19            self.elevators.outside_down_queue.sort()
20            button.setStyleSheet(
21                "QPushButton{border-image: url(./res/down_pressed.png
                      );}")
22            button.setEnabled(False)
23    button_outside_clicked_(self, 1, 'up')
24    floor_list_1 = []
25    elevator_state_list_1 = []
26    floor_list_2 = []
27    elevator_state_list_2 = []
28    cnt = 0
29    while (cnt <= 60):
30        floor_list_1.append(self.elevators.current_floor[0])
31        elevator_state_list_1.append(self.elevators.elevator_state
              [0])
32        floor_list_2.append(self.elevators.current_floor[1])
33        elevator_state_list_2.append(self.elevators.elevator_state
              [1])
34        self.dispatcher.update()
35        cnt += 1
36
37    def removerepeat(list_):
38        new_list = list(set(list_))
39        new_list.sort(key=list_.index)
40        return new_list
41    self.assertEqual(removerepeat(floor_list_1), [0, 0.5, 1])
42    self.assertEqual(removerepeat(elevator_state_list_1), [IDLE, UP])
43    self.assertEqual(removerepeat(floor_list_2), [3])
44    self.assertEqual(removerepeat(elevator_state_list_2),
45                     [IDLE])
```

Brief Descriptions: Elevators are initially at 0F and 3F. Click on the outside 1-up button to see which elevator is to arrive.

Purposes: Check outside 1-up button functions.

Expected Results: Elevator 1 comes.

Test Results: PASSED.

T15: Outside 1-down

Code:

```
def testOutsideButton_1_down(self):
  self.elevators.elevator_state = [IDLE, IDLE]
  self.elevators.current_floor = [0, 3]
  self.elevators.elevator_door = [PAUSE, PAUSE]
  self.elevators.elevator_execute_queue = [[], []]
  self.elevators.inside_queue = [[], []]

  def button_outside_clicked_(self, floor, direction):
      if direction == 'up':
          button = self.elevators.button_up[floor if floor > 0 else
               0]
          self.elevators.outside_up_queue.append(floor)
          self.elevators.outside_up_queue.sort()
          button.setStyleSheet(
              "QPushButton{border-image: url(./res/up_pressed.png)
                  ;}")
          button.setEnabled(False)
      else:
          button = self.elevators.button_down[floor-1]
          self.elevators.outside_down_queue.append(floor)
          self.elevators.outside_down_queue.sort()
          button.setStyleSheet(
              "QPushButton{border-image: url(./res/down_pressed.png
                  );}")
          button.setEnabled(False)
  button_outside_clicked_(self, 1, 'down')
  floor_list_1 = []
  elevator_state_list_1 = []
  floor_list_2 = []
  elevator_state_list_2 = []
  cnt = 0
  while (cnt <= 60):
      floor_list_1.append(self.elevators.current_floor[0])
      elevator_state_list_1.append(self.elevators.elevator_state
          [0])
      floor_list_2.append(self.elevators.current_floor[1])
      elevator_state_list_2.append(self.elevators.elevator_state
          [1])
      self.dispatcher.update()
```

```
35          cnt += 1
36
37      def removerepeat(list_):
38          new_list = list(set(list_))
39          new_list.sort(key=list_.index)
40          return new_list
41      self.assertEqual(removerepeat(floor_list_1), [0, 0.5, 1])
42      self.assertEqual(removerepeat(elevator_state_list_1),
43                       [IDLE, UP_DOWN, DOWN])
44      self.assertEqual(removerepeat(floor_list_2), [3])
45      self.assertEqual(removerepeat(elevator_state_list_2),
46                       [IDLE])
```

Brief Descriptions: Elevators are initially at 0F and 3F. Click on the outside 1-down button to see which elevator is to arrive.

Purposes: Check outside 1-down button functions.

Expected Results: Elevator 1 comes.

Test Results: PASSED.

T16: Outside 2-up

Code:

```
1   def testOutsideButton_2_up(self):
2   self.elevators.elevator_state = [IDLE, IDLE]
3   self.elevators.current_floor = [0, 3]
4   self.elevators.elevator_door = [PAUSE, PAUSE]
5   self.elevators.elevator_execute_queue = [[], []]
6   self.elevators.inside_queue = [[], []]
7
8   def button_outside_clicked_(self, floor, direction):
9       if direction == 'up':
10          button = self.elevators.button_up[floor if floor > 0 else
                0]
11          self.elevators.outside_up_queue.append(floor)
12          self.elevators.outside_up_queue.sort()
13          button.setStyleSheet(
14              "QPushButton{border-image: url(./res/up_pressed.png)
                    ;}")
15          button.setEnabled(False)
16      else:
17          button = self.elevators.button_down[floor-1]
18          self.elevators.outside_down_queue.append(floor)
19          self.elevators.outside_down_queue.sort()
20          button.setStyleSheet(
```

```
21                "QPushButton{border-image: url(./res/down_pressed.png
                      );}")
22            button.setEnabled(False)
23    button_outside_clicked_(self, 2, 'up')
24    floor_list_1 = []
25    elevator_state_list_1 = []
26    floor_list_2 = []
27    elevator_state_list_2 = []
28    cnt = 0
29    while (cnt <= 60):
30        floor_list_1.append(self.elevators.current_floor[0])
31        elevator_state_list_1.append(self.elevators.elevator_state
              [0])
32        floor_list_2.append(self.elevators.current_floor[1])
33        elevator_state_list_2.append(self.elevators.elevator_state
              [1])
34        self.dispatcher.update()
35        cnt += 1
36
37    def removerepeat(list_):
38        new_list = list(set(list_))
39        new_list.sort(key=list_.index)
40        return new_list
41    self.assertEqual(removerepeat(floor_list_1), [0])
42    self.assertEqual(removerepeat(elevator_state_list_1),
43                     [IDLE])
44    self.assertEqual(removerepeat(floor_list_2), [3, 2.5, 2])
45    self.assertEqual(removerepeat(elevator_state_list_2),
46                     [IDLE, DOWN_UP, UP])
```

Brief Descriptions: Elevators are initially at 0F and 3F. Click on the outside 2-up button to see which elevator is to arrive.

Purposes: Check outside 2-up button functions.

Expected Results: Elevator 2 comes.

Test Results: PASSED.

T17: Outside 2-down

Code:

```
1    def testOutsideButton_2_down(self):
2    self.elevators.elevator_state = [IDLE, IDLE]
3    self.elevators.current_floor = [0, 3]
4    self.elevators.elevator_door = [PAUSE, PAUSE]
5    self.elevators.elevator_execute_queue = [[], []]
```

```
6   self.elevators.inside_queue = [[], []]
7
8   def button_outside_clicked_(self, floor, direction):
9       if direction == 'up':
10          button = self.elevators.button_up[floor if floor > 0 else
                0]
11          self.elevators.outside_up_queue.append(floor)
12          self.elevators.outside_up_queue.sort()
13          button.setStyleSheet(
14              "QPushButton{border-image: url(./res/up_pressed.png)
                    ;}")
15          button.setEnabled(False)
16      else:
17          button = self.elevators.button_down[floor-1]
18          self.elevators.outside_down_queue.append(floor)
19          self.elevators.outside_down_queue.sort()
20          button.setStyleSheet(
21              "QPushButton{border-image: url(./res/down_pressed.png
                    );}")
22          button.setEnabled(False)
23  button_outside_clicked_(self, 2, 'down')
24  floor_list_1 = []
25  elevator_state_list_1 = []
26  floor_list_2 = []
27  elevator_state_list_2 = []
28  cnt = 0
29  while (cnt <= 60):
30      floor_list_1.append(self.elevators.current_floor[0])
31      elevator_state_list_1.append(self.elevators.elevator_state
            [0])
32      floor_list_2.append(self.elevators.current_floor[1])
33      elevator_state_list_2.append(self.elevators.elevator_state
            [1])
34      self.dispatcher.update()
35      cnt += 1
36
37  def removerepeat(list_):
38      new_list = list(set(list_))
39      new_list.sort(key=list_.index)
40      return new_list
41  self.assertEqual(removerepeat(floor_list_1), [0])
42  self.assertEqual(removerepeat(elevator_state_list_1),
43                   [IDLE])
44  self.assertEqual(removerepeat(floor_list_2), [3, 2.5, 2])
45  self.assertEqual(removerepeat(elevator_state_list_2),
46                   [IDLE, DOWN])
```

Brief Descriptions: Elevators are initially at 0F and 3F. Click on the outside 2-down button to see which elevator is to arrive.

Purposes: Check outside 2-down button functions.

Expected Results: Elevator 2 comes.

Test Results: PASSED.

3 Functional Test

Functional testing refers to testing a software application by evaluating its functionality against specified requirements. This type of testing involves checking the system's components, such as user interfaces, databases, security features, and more, to ensure they work as expected. Functional tests are typically conducted from the end user's perspective, verifying that the application performs its intended tasks and produces the correct outcomes. This practice helps identify discrepancies between the actual and expected behavior of the system, ensuring that the software meets the specified requirements and provides a good user experience.

Here we mainly include the displaying part and the buttons part.

Display

T18: Floor Display

Purposes: To check whether the LCD screens display correctly.

Test Results: Through random simulation, correct display and elevator floor correspondence.

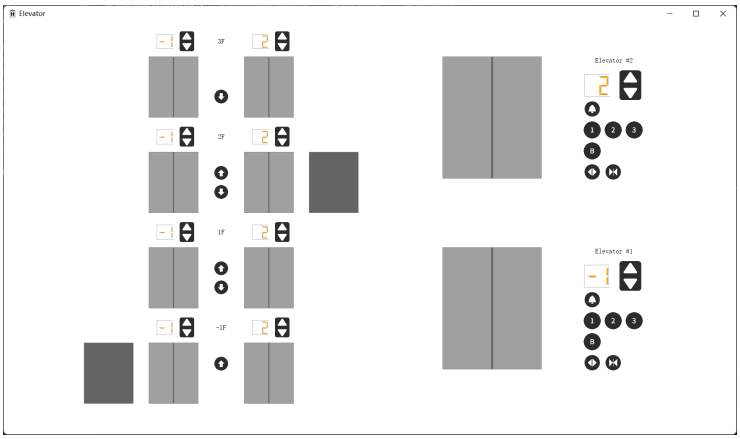

图 1: T18

T19: Button Display

Purposes: To check whether the button display changes correctly.

Test Results: Randomly click on buttons many times, correct button colour display.

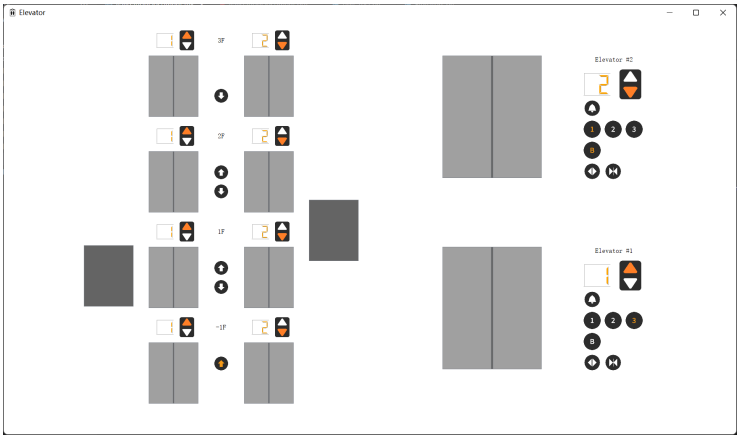

图 2: T19

T20: Direction Display

Purposes: To check whether the button elevator directions display correctly.

Test Results: Randomly click on buttons many times, correct directions display and elevator direction correspondence.

T21: Door Animation Display

Purposes: To check whether the door open and close animations display correctly.

Test Results: Correctly and it's continuous in order to allow door opening and closing without delay.

T22: Moving Elevators Animation Display

Purposes: To check elevator moving animations display correct positions.

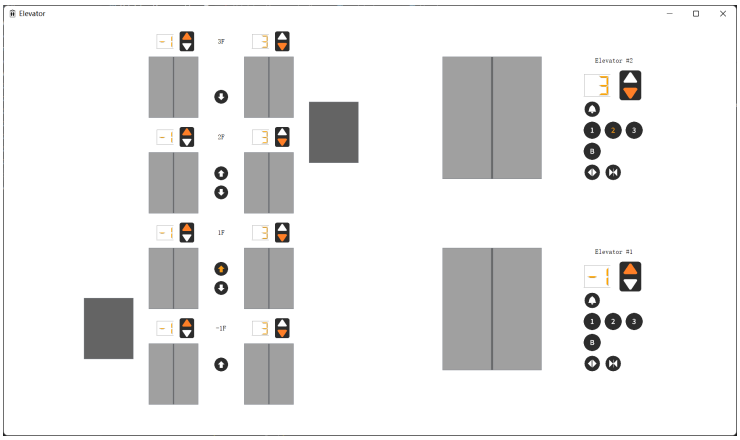
Elevator
3F
2F
1F
-1F
Elevator #2
Elevator #1

图 3: T20

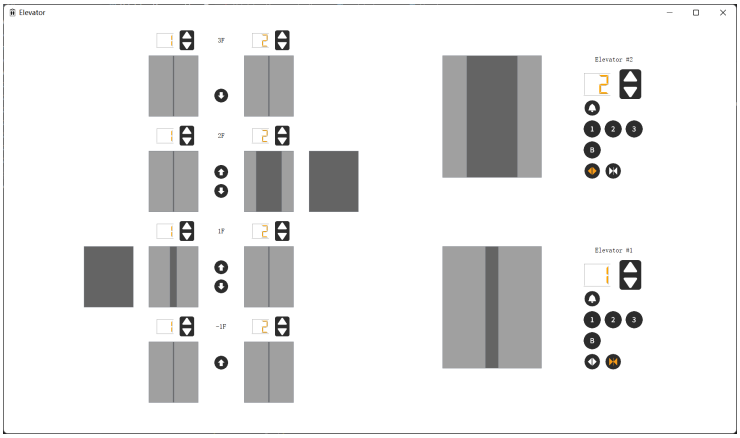
Elevator
3F
2F
1F
-1F
Elevator #2
Elevator #1

图 4: T21

Test Results: Correct.

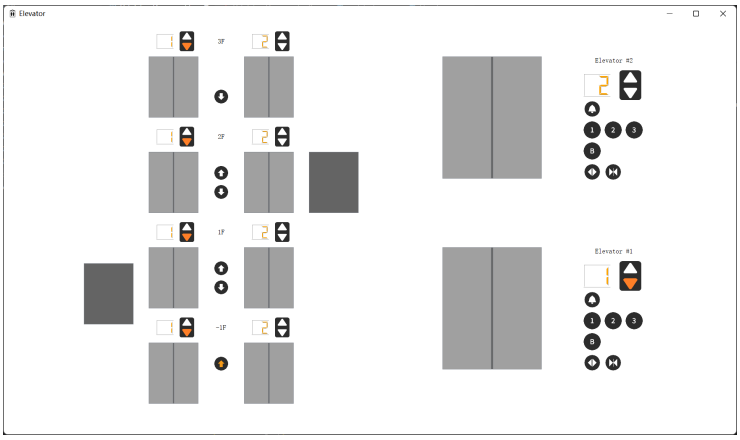

图 5: T22

Buttons

T23: Calling up

Purposes: Test the call-up buttons in the interface.

Test Results: Each simple call-up request will be handled in limited time.

T24: Calling down

Purposes: Test the call-down buttons in the interface.

Test Results: Each simple call-down request will be handled in limited time.

T25: Inside floors

Purposes: Test the inside floor buttons of both elevators in the interface.

Test Results: Each simple floor call request will be handled in limited time.

T26: Inside open and close

Purposes: Test the valid and invalid open and close.

Test Results: Operation will be handled when stopping at a floor(valid), will not be handled when moving(invalid).

T27: Cancelling

Purposes: There is a lately added function: click on a clicked button again to cancel. Test its function correctness.

Test Results: Will cancel the operation and stop at the following floor.

4 Integration Test

Integration testing involves testing the interactions between different components or modules of a software application to ensure they work together correctly. This type of testing focuses on identifying issues that occur when integrating various parts of the system, such as data flow, interface compatibility, and overall interaction between modules. Integration tests help ensure that combined components function as expected, detect faults that may arise from module dependencies, and verify that the system's integrated parts operate smoothly together. This practice helps to uncover problems that unit tests might miss, ensuring a cohesive and reliable software system.

Here the integration tests include all the perspectives, including the dispatcher.

I will show by using commands similar but not same as that of API to show inputs and outputs.

T28

```
1  call-up@1
2  Re: open-close@1
3  select-floor3@2
4  select-floor2@2
5  Re: floor-arrived2@2
6  Re: floor-arrived3@2
7  call-down@1
8  Re: open-close@1
9  select-floorB@1
10 Re: floor-arrivedB@1
11 call-up@2
12 Re: floor-arrived2@2
```

Explanations: The testcase mainly covers basic functions and dispatcher. First, from 1 to 3 when having not reached 2, press 2, the elevator will stop at 2. Second, when both elevators are idle, regardless of the direction, the closer one will go and pick up passengers.

T29

```
call-up@B
Re: floor-arrived2@B
call-down@3
Wait for elevator 1 to move past floor2 but have not reached floor3
call-down@2
Re: floor-arrived1@3
Re: floor-arrived2@2
```

Explanations: Apart from basic functions, this testcase verifies that from 1 to 3 when having reached 2 above, press 2, the elevator won't handle the command immediately.

T30

```
call-up@B
Re: floor-arrived2@B
select-floor3@1
call-down@2
Re: floor-arrived3@1
Re: floor-arrived2@2
select-floorB@1
Wait for elevator 1 to move past floor2 but have not reached floor3
select-floor2@1
Re: floor-arrivedB@1
Re: floor-arrived2@1
```

Explanations: Apart from basic functions, this case verifies invalid call from outside buttons.

T31

```
call-down@1
Re: open-close@2
select-floorB@2
Re: floor-arrivedB@2
call-up@1
Re: open-close@1
select-floor3@2
call-up@2
Re: floor-arrived2@1
Re: floor-arrived3@1
```

Explanations: This case additionally verifies the dispatching processes when both elevators are moving.

5 Model Checking

UPPAAL model checking involves using the UPPAAL tool to verify the correctness of real-time systems by creating and analyzing models of system behavior. This type of verification ensures that the system meets specified temporal properties and constraints, such as timing requirements and concurrency conditions. UPPAAL allows users to model systems using timed automata, simulate their behavior, and check properties expressed in temporal logic. This practice helps identify design flaws, verify system performance, and ensure that the system adheres to critical timing constraints, ultimately contributing to the development of reliable and robust real-time systems.

Here I use models that are not completely the same as that of development, but are similar to them. I use these simplified models to check some basic properties.

Risk Management

1. Make sure each passenger that is picked up will be sent to target floor.

2. Make sure the elevator will not reach floors higher than 3 or lower than B.

3. Make sure each arrival to the following destination does not take too long.

FTA:

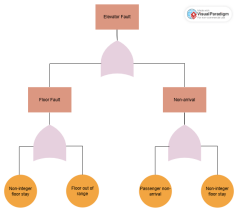

图 6: FTA

UPPAAL Models

T32

The first model assumes that there is only one elevator and passengers enter and exit the elevator in turn.

First define the system:ElevatorController.

Define variables:

```
int current_floor=1;
int out_button=-1;
int in_button=-1;
int minus_times=0;
int each_interval=0;
```

where minustimes represents difference of pickups and sendings of passengers, each_interval represents the time intervals between being picked and being sent.

Below is the state machine of UPPAAL:

图 7: T32 State Machine

Below is the verifications:

图 8: T32 Verification

The three verifications ensure the nonexistence of the three risks in this simplified model.

T33

This models add the interactions between the elevator controller and the passengers. I add 4 passengers totally.

The first model assumes that there is only one elevator too, but passengers not necessarily enter and exit the elevator in turn.

Here is the system definition:

```
passenger1 = Passenger();
passenger2 = Passenger();
passenger3 = Passenger();
passenger4 = Passenger();

system ElevatorController, passenger1,passenger2,passenger3,
    passenger4;
```

Controller variables remain unchanged and the global declarations:

```
broadcast chan can_send, can_pick;
broadcast chan p1, p2, p3, p4;
broadcast chan s1, s2, s3, s4;

int interval=0;
```

Passengers and controller interact through the 8 **broadcast chan**s.

Below is the controller state machine:

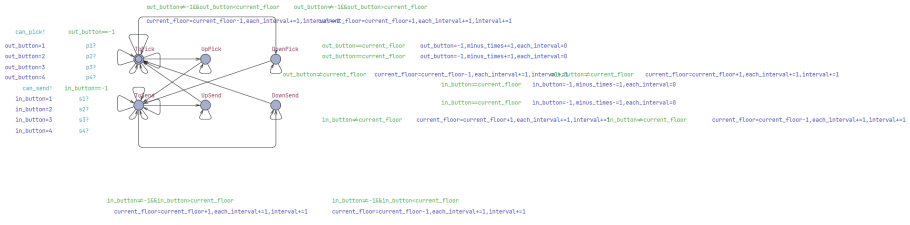

图 9: T33 Controller State Machine

Below is the passenger state machine:

Below is the verification:

We can see that we can always pick and send the four passengers within limited elevator state transfers.

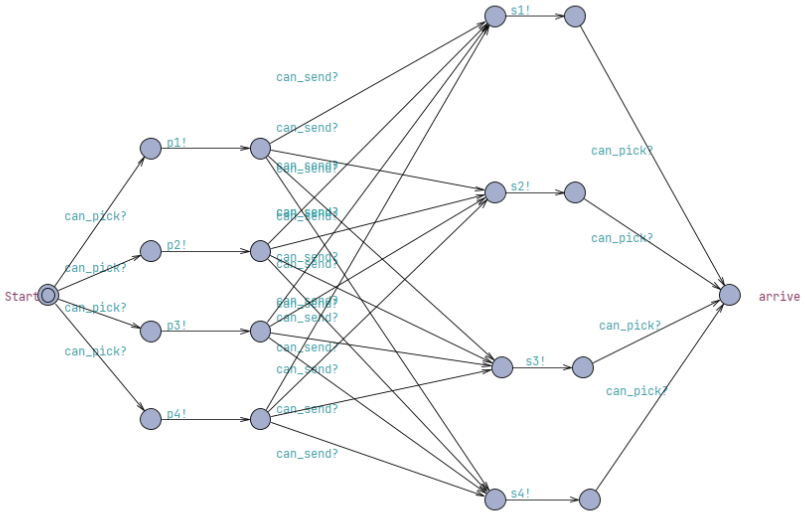

图 10: T33 Passenger State Machine

图 11: T33 Verification