Elevator System

Validation

CS132: Software Engineering

Group 15
Authors: Yuhang Zhu, Yunfei Wang, Shunkang Jia

Contents

1 Unit Test

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

1.1 Elevator

1.1.1 Test Case: Reset

- **Code Under Test**

```
33      def reset(self) -> None:
34          # Move related variables
35          self.currentPos: float = 1.0 # Initially stop at floor 1
36          self.__currentSpeed = 0.1
37          self.currentDirection: Direction = Direction.wait # Direction record
38          self.targetFloor: list[int] = []
39          # Door related variables
40          self.doorSpeed: float = 0.1
41          self.doorInterval: float = 0.0
42          self.doorOpenFlag: bool = False
43          self.doorCloseFlag: bool = False
44          # State
45          self.currentState: State = State.stopped_door_closed
46          # Reset UI
47          self.resetUi()
48          return
```

Figure 1: The reset function implementation

- T1.1 Results

```
test1_reset (__main__.TestElevator.test1_reset)
Test the reset functionality ... ok
```

Figure 2: Test results for reset

- Test coverage : $1/1 = 100\%$.

1.1.2 Test Case: Move

- **Code Under Test**

```python
    def move(self) -> None:
        if self.currentState == State.up:
            self.currentPos += self.__currentSpeed
            self.targetFloor.sort(reverse=False)
        elif self.currentState == State.down:
            self.currentPos -= self.__currentSpeed
            self.targetFloor.sort(reverse=True)

        # Check if the elevator has reached the target floor
        if self.currentPos > self.targetFloor[0]-0.01 and self.currentPos < self.targetFloor[0]+0.01:
            # Arrive! transfer state to stopped_door_opening
            arrivedFloor = self.targetFloor.pop(0)
            self.currentPos = float(arrivedFloor)
            self.floorArrivedMessage(arrivedFloor,self.elevatorId)
            #print("elevator: ",self.elevatorId," arrived at floor: ",arrivedFloor)
            # Clear floor ui
            self.clear_floor_ui(arrivedFloor)
            #print("door opening #"+str(self.elevatorId))
            self.currentState = State.stopped_opening_door
            if len(self.targetFloor) == 0:
                #print("direction reset to wait")
                self.currentDirection = Direction.wait

        return
```

Figure 3: The move function implementation

- T1.2.1, T1.2.2, T1.2.3, T1.2.4 Results

```
test2_1_move_up_not_arrive (__main__.TestElevator.test2_1_move_up_not_arrive)
Test the move functionality without changing status ... ok
test2_2_move_down_not_arrive (__main__.TestElevator.test2_2_move_down_not_arrive)
Test the move functionality when moving down ... ok
test2_3_move_up_arrive_at_floor_without_multiple_target (__main__.TestElevator.test2_3_move_up_arrive_at_floor_without_multiple_target) ... ok
test2_4_move_down_arrive_at_floor_with_multiple_target (__main__.TestElevator.test2_4_move_down_arrive_at_floor_with_multiple_target) ... ok
```

Figure 4: Test results for move

- Test coverage : $4/4 = 100\%$.

1.1.3 Test Case: Opening Door

- **Code Under Test**

```python
    def openingDoor(self) -> None:
        # Ignore Flag
        if self.doorOpenFlag:
            self.doorOpenFlag = False
        if self.doorCloseFlag:
            self.doorCloseFlag = False
        # Keep Opening the door
        self.doorInterval += self.doorSpeed
        if self.doorInterval >= Elevator.doorOpenTime:
            self.doorInterval = 0.0
            #print("door opened #"+str(self.elevatorId))
            self.doorOpenedMessage(self.elevatorId)
            self.currentState = State.stopped_door_opened
        return
```

Figure 5: The openingDoor function implementation

- T1.3.1, T1.3.2 Results

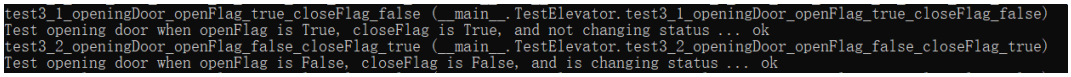

Figure 6: Test results for openingDoor

- Test coverage : $2/2 = 100\%$.

1.1.4 Test Case: Closing Door

- **Code Under Test**

```
89      def closingDoor(self) -> None:
90          # Ignore Repeated Close Flag
91          if self.doorCloseFlag:
92              self.doorCloseFlag = False
93          # Pay attention to Open Flag
94          if self.doorOpenFlag:
95              # If press open button, reopen the door immediately
96              self.doorInterval = Elevator.doorOpenTime - self.doorInterval
97              self.doorOpenFlag = False
98              #print("door opening #"+str(self.elevatorId))
99              self.currentState = State.stopped_opening_door
100             return
101         # Keep Closing the door
102         self.doorInterval += self.doorSpeed
103         if self.doorInterval >= Elevator.doorCloseTime:
104             self.doorInterval = 0.0
105             #print("door closed #"+str(self.elevatorId))
106             self.doorClosedMessage(self.elevatorId)
107             self.currentState = State.stopped_door_closed
108         return
109
```

Figure 7: The closingDoor function implementation

- T1.4.1, T1.4.2, T1.4.3 Results

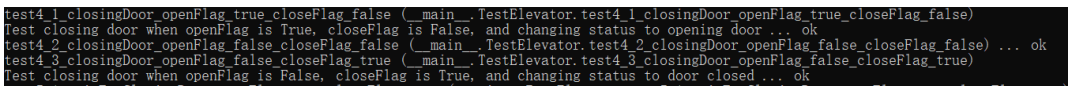

Figure 8: Test results for closingDoor

- Test coverage : $3/3 = 100\%$.

1.1.5 Test Case: Wait For Closing Door

- **Code Under Test**

```python
    def waitForClosingDoor(self) -> None:
        # Open Flag is on, keep opened
        if self.doorOpenFlag:
            self.doorInterval = 0.0
            self.doorOpenFlag = False
            self.doorCloseFlag = False
            return
        # Close? transfer state to closing door
        if self.doorCloseFlag:
            self.doorInterval = 0.0
            self.currentState = State.stopped_closing_door
            self.doorCloseFlag = False
            return

        # Keep waiting
        self.doorInterval += self.doorSpeed
        if self.doorInterval >= Elevator.elevatorWaitTime: ···

        return
```

Figure 9: The waitForClosingDoor function implementation

- T1.5.1, T1.5.2, T1.5.3 Results

```
test5_1_waitForClosingDoor_openFlag_true_closeFlag_true (__main__.TestElevator.test5_1_waitForClosingDoor_openFlag_true_closeFlag_true)
Test door opened(waiting to close) when openFlag is True, closeFlag is True, and changing status to door closed ... ok
test5_2_waitForClosingDoor_openFlag_false_closeFlag_false_larger_waiting (__main__.TestElevator.test5_2_waitForClosingDoor_openFlag_false_closeFlag_false_larger_waiting)
Test door opened(waiting to close) when openFlag is False, closeFlag is False, and changing status to door closed ... ok
test5_3_waitForClosingDoor_openFlag_false_closeFlag_false_less_waiting (__main__.TestElevator.test5_3_waitForClosingDoor_openFlag_false_closeFlag_false_less_waiting) ... ok
test5_4waitForClosingDoor_openFlag_false_closeFlag_true (__main__.TestElevator.test5_4waitForClosingDoor_openFlag_false_closeFlag_true) ... ok
```

Figure 10: Test results for waitForClosingDoor

- Test coverage : $3/3 = 100\%$.

1.1.6 Test Case: Floor Arrived Message

- **Code Under Test**

```python
# Sending Msg
    def floorArrivedMessage(self, floor: int, eid: int) -> None:
        directions = ["up", "down", ""]
        floors = ["-1", "1", "2", "3"]
        elevators = ["#1", "#2"]

        direction_str = directions[self.currentDirection.value]
        floor_str = floors[floor]
        elevator_str = elevators[eid - 1]  # Adjusting elevator index to start from 1

        message = f"floor_arrived@{floor_str}{elevator_str}"
        #print(message)
        self.zmqThread.sendMsg(message)

```

Figure 11: The floorArrivedMessage function implementation

- T1.6 Results

```
test6_floor_arrived_message_up (__main__.TestElevator.test6_floor_arrived_message_up)
Test the floorArrivedMessage when direction is up ... ok
```

Figure 12: Test results for floorArrivedMessage

- Test coverage : 1/1 = 100%.

1.1.7 Test Case: Door Opened Message

- **Code Under Test**

```
147     def doorOpenedMessage(self,eid: int) -> None:
148         elevators = ["#1", "#2"]
149         elevator_str = elevators[eid - 1]
150         message = f"door_opened{elevator_str}"
151         self.zmqThread.sendMsg(message)
```

Figure 13: The doorOpenedMessage function implementation

- T1.7 Results

```
test7_door_opened_message (__main__.TestElevator.test7_door_opened_message)
Test the doorOpenedMessage functionality ... ok
```

Figure 14: Test results for doorOpenedMessage

- Test coverage : 1/1 = 100%.

1.1.8 Test Case: Door Closed Message

- **Code Under Test**

```
153     def doorClosedMessage(self,eid: int) -> None:
154         elevators = ["#1", "#2"]
155         elevator_str = elevators[eid - 1]
156         message = f"door_closed{elevator_str}"
157         self.zmqThread.sendMsg(message)
```

Figure 15: The doorClosedMessage function implementation

- T1.8 Results

```
test8_door_closed_message (__main__.TestElevator.test8_door_closed_message)
Test the doorClosedMessage functionality ... ok
```

Figure 16: Test results for doorClosedMessage

- Test coverage : $1/1 = 100\%$.

1.1.9 Test Case: Check Target Floor

- **Code Under Test**

```
159     def checkTargetFloor(self) -> bool:
160         if len(self.targetFloor) == 0:
161             return False
162         # If there is a target floor, begin to move
163         if self.targetFloor[0] > self.currentPos:
164             #[3,2]
165             self.currentState = State.up
166             self.targetFloor.sort(reverse=(self.currentDirection == Direction.down))
167             #[2,3]
168         elif self.targetFloor[0] < self.currentPos:
169             #[2,1]
170             self.currentState = State.down
171             self.targetFloor.sort(reverse=(self.currentDirection == Direction.down))
172             #[1,2]
173         elif self.targetFloor[0] == self.currentPos:
174             self.targetFloor.remove(int(self.currentPos))
175             self.clear_floor_ui(floor=int(self.currentPos))
176             self.floorArrivedMessage(self.getCurrentFloor(),self.elevatorId)
177             self.currentState = State.stopped_opening_door
178         return True
```

Figure 17: The checkTargetFloor function implementation

- T1.9.1, T1.9.2, T1.9.3, T1.9.4 Results

```
test9_1_checkTargetFloor_none (__main__.TestElevator.test9_1_checkTargetFloor_none)
Test the check target floor functionality: no target floor ... ok
test9_2_checkTargetFloor_changeToUp (__main__.TestElevator.test9_2_checkTargetFloor_changeToUp)
Test the check target floor functionality: target floor upwards ... ok
test9_3_checkTargetFloor_changeToDown (__main__.TestElevator.test9_3_checkTargetFloor_changeToDown)
Test the check target floor functionality: target floor downwards ... ok
test9_4_checkTargetFloor_currentFloor (__main__.TestElevator.test9_4_checkTargetFloor_currentFloor)
Test the check target floor functionality: target floor current ... ok
```

Figure 18: Test results for checkTargetFloor

- Test coverage : $4/4 = 100\%$.

1.1.10 Test Case: Check Open Door

- **Code Under Test**

```
179     def checkOpenDoor(self) -> None:
180         if self.doorOpenFlag:
181             self.doorOpenFlag = False
182             self.currentState = State.stopped_opening_door
183         return
```

Figure 19: The checkOpenDoor function implementation

- T1.10 Results

```
test10_checkOpenDoor_whenDoorOpenFlagTrue (__main__.TestElevator.test10_checkOpenDoor_whenDoorOpenFlagTrue) ... ok
```

Figure 20: Test results for checkOpenDoor

- Test coverage : 1/1 = 100%.

1.1.11 Test Case: Get Current Floor

- **Code Under Test**

```
186 # Util function inside class
187     def getCurrentFloor(self) -> int:
188         return round(self.currentPos)
```

Figure 21: The getCurrentFloor function implementation

- T1.11 Results

```
test11_getCurrentFloor (__main__.TestElevator.test11_getCurrentFloor) ... ok
```

Figure 22: Test results for getCurrentFloor

- Test coverage : 1/1 = 100%.

1.1.12 Test Case: Add Target Floor

- **Code Under Test**

```python
# Utility Functions for controller & button panel inside this elevator
    # Reveive outer request from controller
    def addTargetFloor(self, floor: int) -> str:
        if floor in self.targetFloor:
            return "OK"
        # Determine the direction of the elevator when adding the first target floor
        if self.currentDirection == Direction.wait:
            if(self.currentPos < floor):
                self.currentDirection = Direction.up # up
            elif(self.currentPos > floor): # down
                self.currentDirection = Direction.down
        self.targetFloor.append(floor)
        self.targetFloor.sort(reverse=(self.currentDirection == Direction.down))
        #print("current target floor: ",self.targetFloor)
        return "OK"
```

Figure 23: The addTargetFloor function implementation

- T1.12.1, T1.12.2, T1.12.3 Results

```
test12_1_addTargetFloor_whenFloorAlreadyInTarget (__main__.TestElevator.test12_1_addTargetFloor_whenFloorAlreadyInTarget) ... ok
test12_2_addTargetFloor_whenDirectionWaitAndFloorAbove (__main__.TestElevator.test12_2_addTargetFloor_whenDirectionWaitAndFloorAbove) ... ok
test12_3_addTargetFloor_whenDirectionWaitAndFloorBelow (__main__.TestElevator.test12_3_addTargetFloor_whenDirectionWaitAndFloorBelow) ... ok
```

Figure 24: Test results for addTargetFloor

- Test coverage : 3/3 = 100%.

1.1.13 Test Case: Set Open Door Flag

- **Code Under Test**

```python
    def setOpenDoorFlag(self) -> None:
        self.doorOpenFlag = True
        return
```

Figure 25: The setOpenDoorFlag function implementation

- T1.13 Results

```
test13_setOpenDoorFlag (__main__.TestElevator.test13_setOpenDoorFlag) ... ok
```

Figure 26: Test results for setOpenDoorFlag

- Test coverage : 1/1 = 100%.

1.1.14 Test Case: Set Close Door Flag

- **Code Under Test**

```
209     def setCloseDoorFlag(self) -> None:
210         self.doorCloseFlag = True
211         return
```

Figure 27: The setCloseDoorFlag function implementation

- T1.14 Results

```
test14_setCloseDoorFlag (__main__.TestElevator.test14_setCloseDoorFlag) ... ok
```

Figure 28: Test results for setCloseDoorFlag

- Test coverage : 1/1 = 100%.

1.1.15 Test Case: Get Door Percentage

- **Code Under Test**

```
212     def getDoorPercentage(self) -> float:
213         # opened -> 1.0; closed -> 0.0
214         if self.currentState == State.stopped_closing_door:
215             return 1.0 - self.doorInterval/Elevator.doorCloseTime
216         elif self.currentState == State.stopped_opening_door:
217             return self.doorInterval/Elevator.doorOpenTime
218         elif self.currentState == State.stopped_door_opened:
219             return 1.0
220         else:
221             return 0.0
```

Figure 29: The getDoorPercentage function implementation

- T1.15.1, T1.15.2, T1.15.3, T1.15.4 Results

```
test15_1_getDoorPercentage_whenStoppedClosingDoor (__main__.TestElevator.test15_1_getDoorPercentage_whenStoppedClosingDoor) ... ok
test15_2_getDoorPercentage_whenStoppedOpeningDoor (__main__.TestElevator.test15_2_getDoorPercentage_whenStoppedOpeningDoor) ... ok
test15_3_getDoorPercentage_whenStoppedDoorOpened (__main__.TestElevator.test15_3_getDoorPercentage_whenStoppedDoorOpened) ... ok
test15_4_getDoorPercentage_whenOtherStates (__main__.TestElevator.test15_4_getDoorPercentage_whenOtherStates) ... ok
```

Figure 30: Test results for getDoorPercentage

- Test coverage : 4/4 = 100%.

1.1.16 Test Case: Update

- **Code Under Test**

```
222     def update(self) -> None:
223         self.updateUi()
224         if self.currentState == State.up or self.currentState == State.down:
225             #print("elevator: ",self.elevatorId," is moving",self.currentState.name)
226             self.move()
227             pass
228         elif self.currentState == State.stopped_opening_door:
229             self.openingDoor()
230             pass
231         elif self.currentState == State.stopped_door_opened:
232             self.waitForClosingDoor()
233             pass
234         elif self.currentState == State.stopped_closing_door:
235             self.closingDoor()
236             pass
237         elif self.currentState == State.stopped_door_closed:
238             # Find if Controller give new command to this elevator
239             hasTarget = self.checkTargetFloor()
240             # If door is already closed,  user can still exit or enter by request opening door.
241             if not hasTarget:
242                 self.checkOpenDoor()
243             pass
244         return
```

Figure 31: The update function implementation

- T1.16 Results

```
test16_update (__main__.TestElevator.test16_update)
Test the update functionality ... ok
```

Figure 32: Test results for update

- Test coverage : 1/1 = 100%.

1.2 Elevator Controller

Due to space constraints, we only show the final results of here.
{T2.1.1-T2.1.19, T2.2, T2.3, T2.4, T2.5, T2.6, T2.7, T2.8}

```
.up_floor_arrived@1#1
.....1_down clicked
.2_down clicked
.3_down clicked
.1_up clicked
.2_up clicked
.-1_up clicked
..............
----------------------------------------------------------------------
Ran 26 tests in 0.088s

OK
```

Figure 33: Test results for Elevator Controller

Test coverage :26/26 = 100% .

2 Integration Test

2.1 Controller + Elevator

2.1.1 Synchronization

To ensure the elevator control system components work together smoothly, we designed several integration test cases. These tests verify the correctness of the elevator scheduling algorithm and operation when multiple passengers call elevators from different floors simultaneously. Below are two key test cases:

T3.1.1: Free Ride

- **Scenario**:
  1. Passenger A calls the elevator at the 1st floor and selects the 3rd floor.
  2. Passenger B calls the elevator at the 2nd floor, wanting to go up.
  3. The same elevator should go to the 2nd floor and pick up passenger B.
- **Messages**:
  - `"call_up@1"` - Passenger A calls the elevator at the 1st floor.
  - `"select_floor@3#1"` - Passenger A selects the 3rd floor.
  - `"call_up@2"` - Passenger B calls the elevator at the 2nd floor.
- **Validation**:
  - Check if the elevator arrives at the 2nd and 3rd floors.
  - Expected results:

```
"up_floor_arrived@2#1"
"floor_arrived@3#1"
```

T3.1.2: Parallel Moving

- **Scenario**:
  1. Passenger A calls the elevator at the 1st floor and selects the 3rd floor.
  2. Passenger B calls the elevator at the 2nd floor, wanting to go down.
  3. The other elevator should go to the 2nd floor to pick up passenger B.
- **Messages**:
  - `"call_up@1"` - Passenger A calls the elevator at the 1st floor.
  - `"call_down@2"` - Passenger B calls the elevator at the 2nd floor.

  - `"select_floor@3#1"` - Passenger A selects the 3rd floor.
  - `"select_floor@1#2"` - Passenger B selects the 1st floor.
- **Validation**:
  - Check if the elevators arrive at the 2nd and 3rd floors.
  - Expected results:

```
"floor_arrived@2#2"
"floor_arrived@3#1"
```

2.1.2 Passenger Validation

This test ensures that passengers are correctly picked up and dropped off at their target floors, even when passing intermediate floors.

T3.2.1: Passenger Route Validation

- **Scenario**:
  1. Passenger A calls the elevator at the 3rd floor, targeting the ground floor.
  2. Passenger B calls the elevator at the 2nd floor, targeting the 3rd floor.
  3. Passenger C calls the elevator at the 1st floor, targeting the 3rd floor.
  4. The elevators should pick up and drop off each passenger at their respective target floors.
- **Messages**:
  - `"call_down@3"` - Passenger A calls the elevator at the 3rd floor.
  - `"call_up@2"` - Passenger B calls the elevator at the 2nd floor.
  - `"call_up@1"` - Passenger C calls the elevator at the 1st floor.
- **Validation**:
  - Ensure each passenger reaches their target floor and exits the elevator.
  - Expected result:

```
"Passenger A has arrived at the target floor."
"Passenger B has arrived at the target floor."
"Passenger C has arrived at the target floor."
```

3 Risk Management for Functional Testing

3.1 Risks

1. **Incorrect Scheduling Algorithm Implementation**
   - **Impact**: The elevator might not pick up passengers efficiently, leading to delays and user frustration.
   - **Mitigation**: Implement comprehensive unit tests for the scheduling algorithm. Perform code reviews and pair programming during development.
2. **Concurrency Issues**
   - **Impact**: Simultaneous requests might cause race conditions, leading to incorrect elevator behavior.
   - **Mitigation**: Use thread-safe data structures and concurrency control mechanisms. Conduct stress testing to identify and fix concurrency issues.
3. **Edge Cases**
   - **Impact**: Unhandled edge cases, such as simultaneous opposite requests on the same floor, can lead to unexpected behavior.
   - **Mitigation**: Identify and document potential edge cases. Ensure test cases cover all identified edge cases.

3.2 Test Cases and Mitigation

- **TestCase1**: Both elevators start on the first floor, and a passenger presses the up button.
  - **Mitigation**: Ensure the algorithm assigns the nearest available elevator.
- **TestCase2**: An elevator is near a floor when a request is made.
  - **Mitigation**: The scheduling algorithm should reassign the closest elevator without causing significant delays.
- **TestCase3**: Multiple requests are made on the same floor with different directions.
  - **Mitigation**: Ensure the algorithm handles simultaneous requests correctly without prioritizing one direction unfairly.
- **TestCase4**: Simultaneous requests from different floors.
  - **Mitigation**: Ensure the algorithm distributes requests evenly among elevators to optimize efficiency.
- **TestCase5**: Requests from different floors with overlapping destinations.
  - **Mitigation**: Ensure the algorithm handles overlapping requests efficiently.

4 Door Control System

4.1 Risks

1. **Door Not Opening/Closing Properly**
   - **Impact**: Passengers might be unable to enter or exit, causing safety concerns and operational delays.
   - **Mitigation**: Implement fail-safes and sensors to detect and correct door malfunctions. Conduct regular maintenance checks.
2. **Door Obstruction Detection Failure**
   - **Impact**: The door might close on passengers, causing injuries.
   - **Mitigation**: Install and regularly test obstruction detection sensors. Implement an emergency stop mechanism.
3. **Simultaneous Door Control Commands**
   - **Impact**: Conflicting commands (e.g., open and close) might cause erratic door behavior.
   - **Mitigation**: Ensure door control logic prioritizes safety and resolves conflicting commands systematically.

4.2 Test Cases and Mitigation

- **TestCase1**: Passenger presses the up button and tries to open the door just before it closes.
  - **Mitigation**: Ensure the door reopens when the open button is pressed during closing.
- **TestCase2**: Passenger continuously presses the open button.
  - **Mitigation**: Ensure the door remains open as long as the button is pressed and closes only after a timeout.
- **TestCase3**: Passenger presses the close button while the door is opening.
  - **Mitigation**: Ensure the door completes the opening cycle before attempting to close.
- **TestCase4**: Simultaneous door open requests from inside and outside the elevator.
  - **Mitigation**: Ensure the door control system prioritizes opening for passenger safety.

5 Functionality Test

5.1 Scheduling Algorithm Test

- **T4.1.1**

  1. Both elevators start at floor 1. Passenger A presses the up button at floor 1, enters the elevator, and presses button 3.
  2. After 1s of elevator 1 moving up, Passenger B presses the up button at floor 2.
  3. Elevator 1 stops at floor 2, Passenger B enters.
  4. Elevator 1 reaches floor 3, Passengers A and B reach their destination.

- **T4.1.2**

  1. Both elevators start at floor 1. Passenger A presses the up button at floor 1, enters the elevator, and presses button 3.
  2. After 1.8s of elevator 1 moving up, Passenger B presses the up button at floor 2.
  3. Elevator 1 continues moving up as it is close to floor 2.
  4. Elevator 2 moves up to floor 2, Passenger B enters and presses button 3.
  5. Elevators 1 and 2 reach floor 3, Passengers A and B reach their destination.

- **T4.1.3**

  1. Both elevators start at floor 1. Passengers A and B at floor 2 press the up and down buttons 0.5s apart.
  2. Both elevators move to floor 2. After doors close, Passengers A and B press the up and down buttons again 0.5s apart.
  3. Passenger B enters elevator 2 and presses button -1. Passenger A enters elevator 1 and presses button 3.
  4. Elevator 2 reaches floor -1, and elevator 1 reaches floor 3. Passengers A and B reach their destinations.

- **T4.1.4**

  1. Passengers A and B in elevators 1 and 2 press button 3 with a 0.5s interval. Both elevators move up.
  2. Passenger C at floor 2 presses the up button.
  3. Elevator 1 reaches floor 3 and then moves to floor 2, Passenger C enters.
  4. Elevator 2 stops at floor 3.

- **T4.1.5**

  1. Passenger B in elevator 2 presses button -1, elevator 2 moves to floor -1.
  2. Passenger A in elevator 1 presses button 3, elevator 1 moves to floor 3.

3. Passenger A presses button -1 and Passenger B presses button 3, both elevators start moving.
4. Passenger D at floor 1 presses the down button, Passenger C at floor 2 presses the up button.
5. Elevator 1 stops at floor 1, Passenger D enters.
6. Elevator 2 stops at floor 2, Passenger C enters.
7. Elevator 1 reaches floor -1, and elevator 2 reaches floor 3. Passengers A, B, C, and D reach their destinations.

- **T4.1.6**

1. Passenger A in elevator 1 presses button 3, elevator moves up.
2. After passing floor 2, Passenger A presses button 2, no response.
3. Elevator 1 continues to floor 3.

- **T4.1.7**

1. Passenger A in elevator 1 presses button 3, elevator moves up.
2. After 1.5s, Passenger A presses button 2, no response as the elevator is close to floor 2.
3. Elevator 1 continues to floor 3.

- **T4.1.8**

1. Passenger A in elevator 1 presses button 3, elevator moves up.
2. After 1s, Passenger A presses button 2, floor 2 button lights up.
3. Elevator 1 stops at floor 2, then continues to floor 3.

5.2 Door Control System

- **T4.2.1**

1. Passenger A at floor 1 presses the up button, elevator 1 door opens, A enters and presses button 2.
2. As the door is closing, press the open button, the door reopens.
3. Wait for the door to close, elevator 1 moves up, A reaches floor 2.

- **T4.2.2**

1. Passenger A at floor 1 presses the up button, elevator 1 door opens, A enters and presses button 2.
2. While the door is open, press the open button 3 times, the door stays open.
3. Wait for the door to close, elevator 1 moves up, A reaches floor 2.

- **T4.2.3**

1. Passenger A at floor 1 presses the up button, elevator 1 door opens, A enters and presses button 2.
2. As the door is opening, press the close button, the door continues opening.
3. Wait for the door to close, elevator 1 moves up, A reaches floor 2.

- **T4.2.4**

1. Passengers A and B in elevators 1 and 2 press button 3 with a 0.5s interval, both elevators move up.
2. Passenger C at floor 2 presses the up button.
3. Elevator 1 reaches floor 3, A presses the open button 3 times.
4. Elevator 2 moves to floor 2, C enters.

6 Model Checking

6.1 Introduction

The model consists of an abstract passenger, 4 button on 3 floors: floor_3_down, floor_2_up, floor_2_down,floor_1_up, and 2 elevators.

6.1.1 Assumption

In this model, we assume that

- The abstract passenger can call at any floor.
- Button has three state: Not pressed, pressed, assigned (this is align with our actual implementation)
- Elevator has six main state (except for some intermediate state): stop_door_closed, up, down, stop_opening_door, stop_door_opened, stop_closing door. (this is align with our actual implementation)
- We only do the model checking for 3 floors since it is can be extended to 4 floor with 1 extra basement.

6.2 Properties Validation

- **No deadlock**

$$A[]\ !\ deadlock$$

Description: The whole system should not have deadlock.
Result: Passed

- **Elevator position safety constraint**

$$A[]\ elevator1.currentPos \geq 10\ and\ elevator1.currentPos \leq 30$$

$$A[]\ elevator2.currentPos \geq 10\ and\ elevator2.currentPos \leq 30$$

Description: Elevator should stay within the floor threshold.
Result: Passed

- **Elevator door safety constraint**

$$A[]\ ((elevator1.up\ or\ elevator1.down)\ imply\ elevator1.doorOpen == false)$$

$$A[]\ ((elevator2.up\ or\ elevator2.down)\ imply\ elevator2.doorOpen == false)$$

Description: If elevator is moving, the door should always be closed.
Result: Passed

- **Elevator only stop at the exact floor position**

A[] (elevator1.stop_door_closed or elevator1.stop_opening_door or elevator1.stop_door_opened or elevator1.stop_closing_door) imply (elevator1.currentPos == 10 or elevator1.currentPos == 20 or elevator1.currentPos == 30)

$A[]$ ($elevator2.stop\_door\_closed$ *or* $elevator2.stop\_opening\_door$ *or* $elevator2.stop\_door\_opened$ *or* $elevator2.stop\_closing\_door$) *imply* ($elevator2.currentPos == 10$ *or* $elevator2.currentPos == 20$ *or* $elevator2.currentPos == 30$)

Description: If elevator is stopped, then it must always stop at the exact floor position, not elsewhere between floors.
Result: Passed