SOFTWARE VALIDATION

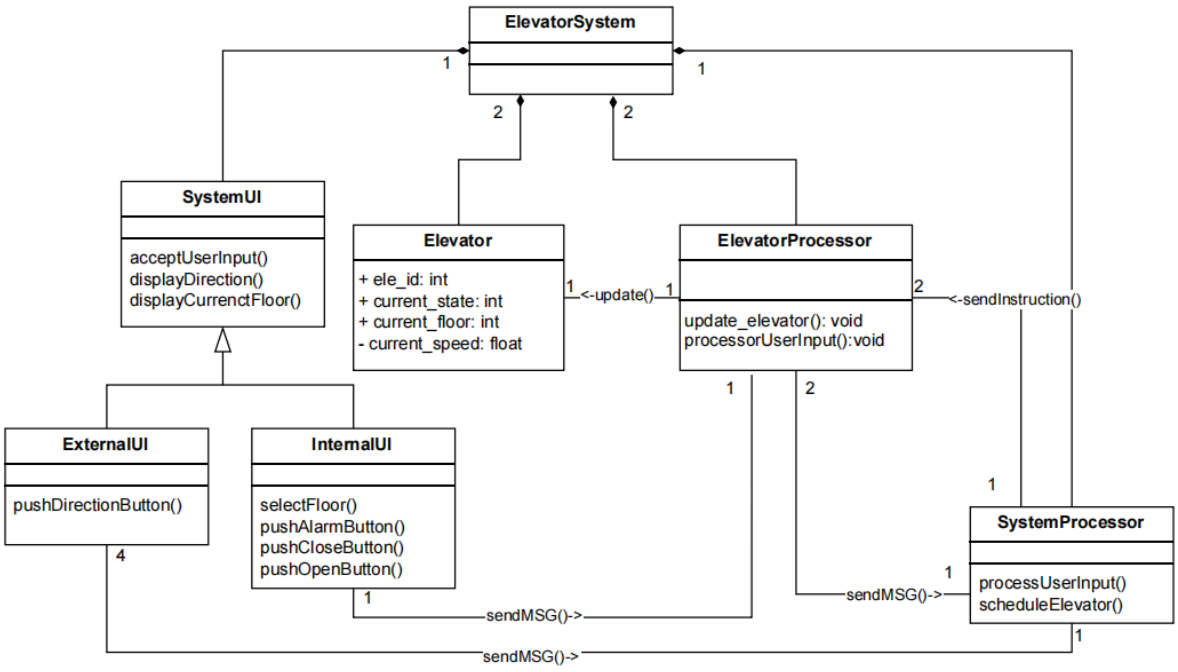
Elevator

Group 7
Author: Jintong Luo

Table of Contents

System Architecture

The system architecture of this Elevator system is shown below:



T1: Unit Test

This section provides information of unit tests for all the functions in the specfication we made for the Elevator System. You can find executable files in the corresponding files.

T1.1: ElevatorProcessor Unit Test

T1.1.1: Test getStaticDoor()

```
def getStaticDoor(self):
    return 2
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.1.1
Coverage Item	Tcover1.1.1.1
Input	
State	
Expected Output	2

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

T1.1.2: Test getStaticTimer()

```
def getStaticTimer(self):
    return 5
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.2.1
Coverage Item	Tcover1.1.2.1
Input	
State	
Expected Output	5

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

T1.1.3: Test update()

```
def update(self):
    self.update_door()
    self.update_floor()
    self.update_direction()
    self.update_state()
    if self.checkArrive():
        arrive_floor = self.elevator.current_floor
        moving_direction = self.elevator.direction
        if moving_direction == DirectionState.idle:
            moving_direction = ""
        elif moving_direction == DirectionState.up:
            moving_direction = "up_"
        elif moving_direction == DirectionState.down:
            moving_direction = "down_"
        if self.system_processor:
            self.smg_to_SystemProcessor(f"{moving_direction}floor_{arrive_flo
or}_arrived#{self.elevator.ele_id}")
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.3.1	Test Case T1.1.3.2
Coverage Item	Tcover1.1.3.1	Tcover1.1.3.2
Input		
State	processor = SystemProcessor().elevator _processors[0]	processor = SystemProcessor().elevator _processors[0]

	elevator = processor.elevator elevator.ele_id = 1 elevator.current_floor = 2 elevator.direction = DirectionState.idle processor.checkArrive.return_value = True	elevator = processor.elevator elevator.ele_id = 1 elevator.current_floor = 3 elevator.direction = DirectionState.up processor.checkArrive.return_value = True
Expected Output	processor.update_door(), processor.update_floor(), processor.update_direction(), processor.update_state(), processor.checkArrive(), processor.smg_to_SystemProcessor("floor_2_arrived#1") are called.	processor.update_door(), processor.update_floor(), processor.update_direction(), processor.update_state(), processor.checkArrive(), processor.smg_to_SystemProcessor("up_floor_3_arrived#1") are called.
	Test Case T1.1.3.3	Test Case T1.1.3.4
Coverage Item	Tcover1.1.3.3	Tcover1.1.3.4
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.ele_id = 1 elevator.current_floor = 1 elevator.direction = DirectionState.down processor.checkArrive.return_value = True	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.ele_id = 1 elevator.current_floor = 1 elevator.direction = DirectionState.up processor.checkArrive.return_value = False
Expected Output	processor.update_door(), processor.update_floor(), processor.update_direction(), processor.update_state(), processor.checkArrive(), processor.smg_to_SystemProcessor("down_floor_1_arrived#1") are called.	processor.update_door(), processor.update_floor(), processor.update_direction(), processor.update_state(), processor.checkArrive()are called. processor.smg_to_SystemProcessor() should not be called.

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

T1.1.4: Test update_floor()

```
    def update_floor(self):
        state = self.elevator.current_state
        direction = self.elevator.direction
        if state == ElevatorState.up or state == ElevatorState.down:
            if direction == DirectionState.up and self.elevator.current_floor <
3:
                if self.elevator.current_floor == -1:
                    self.elevator.current_floor = 1
                else:
                    self.elevator.current_floor = self.elevator.current_floor + 1
            elif direction == DirectionState.down and
self.elevator.current_floor > -1:
                if self.elevator.current_floor == 1:
                    self.elevator.current_floor = -1
                else:
                    self.elevator.current_floor = self.elevator.current_floor - 1
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.4.1	Test Case T1.1.4.2
Coverage Item	Tcover1.1.4.1	Tcover1.1.4.2
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.up elevator.current_state = ElevatorState.up	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = -1 elevator.direction = DirectionState.up elevator.current_state = ElevatorState.up
Expected Output	elevator.current_floor == 2	elevator.current_floor == 1
	Test Case T1.1.4.3	Test Case T1.1.4.4
Coverage Item	Tcover1.1.4.3	Tcover1.1.4.4
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 2

	elevator.direction = DirectionState.down elevator.current_state = ElevatorState.down	elevator.direction = DirectionState.down elevator.current_state = ElevatorState.down
Expected Output	elevator.current_floor == -1	elevator.current_floor == 1
	Test Case T1.1.4.5	
Coverage Item	Tcover1.1.4.5	
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.idle elevator.current_state = ElevatorState.stopped_door_closed	
Expected Output	elevator.current_floor == 1	

- Test coverage: 5 / 5 = 100%
- Test result: 5 passed

T1.1.5: Test update_door()

```
def update_door(self):
    state = self.elevator.current_state
    if state == ElevatorState.stopped_opening_door:
        if self.door_outside_length == 0:
            self.elevator.current_state = ElevatorState.stopped_door_opened
            self.open_timer = self.getStaticTimer()
        else:
            self.door_outside_length = self.door_outside_length - 1
    elif state == ElevatorState.stopped_closing_door:
        if self.door_outside_length == self.getStaticDoor():
            self.elevator.current_state = ElevatorState.stopped_door_closed
        else:
            self.door_outside_length = self.door_outside_length + 1
    elif state == ElevatorState.stopped_door_opened:
        if self.open_timer == 0:
            self.elevator.current_state = ElevatorState.stopped_closing_door
        else:
            self.open_timer = self.open_timer - 1
```

- Coverage Criteria: Branch coverage

- Test case

	Test Case T1.1.5.1	Test Case T1.1.5.2
Coverage Item	Tcover1.1.5.1	Tcover1.1.5.2
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_opening_door processor.door_outside_length = 0	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_opening_door processor.door_outside_length = 1
Expected Output	elevator.current_state == ElevatorState.stopped_door_opened processor.open_timer == processor.getStaticTimer()	processor.door_outside_length == 0
	Test Case T1.1.5.3	Test Case T1.1.5.4
Coverage Item	Tcover1.1.5.3	Tcover1.1.5.4
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_closing_door processor.door_outside_length = processor.getStaticDoor	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_closing_door processor.door_outside_length = 1
Expected Output	elevator.current_state == ElevatorState.stopped_door_closed	processor.door_outside_length == 2
	Test Case T1.1.5.5	Test Case T1.1.5.6
Coverage Item	Tcover1.1.5.5	Tcover1.1.5.6
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator

	elevator.current_state = ElevatorState.stopped_door_opened processor.open_timer = 0	elevator.current_state = ElevatorState.stopped_door_opened processor.open_timer = 3
Expected Output	elevator.current_state == ElevatorState.stopped_closing_door	process.open_timer == 2

- Test coverage: 6 / 6 = 100%
- Test result: 6 passed

T1.1.6: Test update_direction()

```
def update_direction(self):
    floor = self.elevator.current_floor
    targets = self.target_floor
    targets_up = self.target_floor_up
    targets_down = self.target_floor_down
    direction = self.elevator.direction
    if direction == DirectionState.idle:
        for i in range(floor, 4):
            if targets_up[i] or targets_down[i]:
                self.elevator.direction = DirectionState.up
                return
        for i in range(floor, -1, -1):
            if targets_up[i] or targets_down[i]:
                self.elevator.direction = DirectionState.down
                return
        next_floor = floor
        distance = 10
        for i in range(4):
            if targets[i] and abs(i - floor) < distance:
                distance = abs(i - floor)
                next_floor = i
        if next_floor < floor:
            self.elevator.direction = DirectionState.down
        elif next_floor > floor:
            self.elevator.direction = DirectionState.up
    elif direction == DirectionState.up:
        for i in range(floor, 4):
            if targets[i] or targets_up[i] or targets_down[i]:
                return
        for i in range(floor, -1, -1):
            if targets[i] or targets_up[i] or targets_down[i]:
                self.elevator.direction = DirectionState.down
```

```
                return
            self.elevator.direction = DirectionState.idle
        elif direction == DirectionState.down:
            for i in range(floor, -1, -1):
                if targets[i] or targets_up[i] or targets_down[i]:
                    return
            for i in range(floor, 4):
                if targets[i] or targets_up[i] or targets_down[i]:
                    self.elevator.direction = DirectionState.up
                    return
            self.elevator.direction = DirectionState.idle
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.6.1	Test Case T1.1.6.2
Coverage Item	Tcover1.1.6.1	Tcover1.1.6.2
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.idle processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, True, False] processor.target_floor_down = [False, False, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 2 elevator.direction = DirectionState.idle processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, True, False, False]
Expected Output	elevator.direction == DirectionState.up	elevator.direction == DirectionState.down
	Test Case T1.1.6.3	Test Case T1.1.6.4
Coverage Item	Tcover1.1.6.3	Tcover1.1.6.4
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.idle	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 3 elevator.direction = DirectionState.idle

	processor.target_floor = [False, True, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, True, False]	processor.target_floor = [False, False, True, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, True, False, False]
Expected Output	elevator.direction == DirectionState.up	elevator.direction == DirectionState.down
	Test Case T1.1.6.5	Test Case T1.1.6.6
Coverage Item	Tcover1.1.6.5	Tcover1.1.6.6
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.up processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, True, False] processor.target_floor_down = [False, False, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 2 elevator.direction = DirectionState.up processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, True, False, False]
Expected Output	elevator.direction == DirectionState.up	elevator.direction == DirectionState.down
	Test Case T1.1.6.7	Test Case T1.1.6.8
Coverage Item	Tcover1.1.6.7	Tcover1.1.6.8
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.down processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, True, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.down processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, True, False] processor.target_floor_down = [False, False, False, False]

Expected Output	elevator.direction == DirectionState.down	elevator.direction == DirectionState.up
	Test Case T1.1.6.9	Test Case T1.1.6.8
Coverage Item	Tcover1.1.6.9	Tcover1.1.6.8
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.up processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_floor = 1 elevator.direction = DirectionState.down processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, False, False]
Expected Output	elevator.direction == DirectionState.idle	elevator.direction == DirectionState.idle

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

T1.1.7: Test update_state()

```
def update_state(self):
    direction = self.elevator.direction
    if not self.checkOpen():
        if direction == DirectionState.up:
            self.elevator.current_state = ElevatorState.up
        elif direction == DirectionState.down:
            self.elevator.current_state = ElevatorState.down
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.7.1	Test Case T1.1.7.2
Coverage Item	Tcover1.1.7.1	Tcover1.1.7.2
Input		
State	processor = SystemProcessor().elevator_processors[0]	processor = SystemProcessor().elevator_processors[0]

	elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_closed elevator.direction = DirectionState.up	elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_closed elevator.direction = DirectionState.down
Expected Output	elevator.current_state == ElevatorState.up	elevator.current_state == ElevatorState.down
	Test Case T1.1.7.3	
Coverage Item	Tcover1.1.7.3	
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_opened elevator.direction = DirectionState.up	
Expected Output	elevator.current_state == ElevatorState.stopped_door_opened	

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

T1.1.8: Test process_InternalUI_requests(InterUI_MSG)

```
def process_InternalUI_requests(self, InterUI_MSG = ""):
    if InterUI_MSG == "open_door":
        self.open_door()
    elif InterUI_MSG == "close_door":
        self.close_door()
    elif InterUI_MSG.startswith("select_floor"):
        select_floor = int(InterUI_MSG.split("@")[1].split("#")[0])
        select_floor = 0 if (select_floor == -1) else select_floor
        self.target_floor[select_floor] = True
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.8.1	Test Case T1.1.8.2
Coverage Item	Tcover1.1.8.1	Tcover1.1.8.2

Input	"open_door"	"close_door"
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, False, False]
Expected Output	processor.open_door() should be called, while processor.close_door() should not be called.	processor.close_door() should be called, while processor.open_door() should not be called.
	Test Case T1.1.8.3	Test Case T1.1.3.4
Coverage Item	Tcover1.1.8.3	Tcover1.1.3.4
Input	"select_floor@2#1"	"select_floor@-1#1"
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, False, False]
Expected Output	processor.target_floor[2] == True	processor.target_floor[0] == True

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

T1.1.9: Test open_door()

```
def open_door(self):
    state = self.elevator.current_state
    if state == ElevatorState.stopped_door_closed:
        self.elevator.current_state = ElevatorState.stopped_opening_door
        return True
    elif state == ElevatorState.stopped_closing_door:
```

```
            self.elevator.current_state = ElevatorState.stopped_opening_door
            return True
        else:
            return False
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.9.1	Test Case T1.1.9.2
Coverage Item	Tcover1.1.9.1	Tcover1.1.9.2
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_closed	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_closing_door
Expected Output	True. elevator.current_state == ElevatorState.stopped_opening_door	True. elevator.current_state == ElevatorState.stopped_opening_door
	Test Case T1.1.9.3	
Coverage Item	Tcover1.1.9.3	
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_opened	
Expected Output	False. elevator.current_state == ElevatorState.up	

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

T1.1.10: Test close_door()

```
    def close_door(self):
        state = self.elevator.current_state
        if state == ElevatorState.stopped_door_opened:
```

```
            self.elevator.current_state = ElevatorState.stopped_door_closed
            return True
        elif state == ElevatorState.stopped_opening_door:
            self.elevator.current_state = ElevatorState.stopped_door_closed
            return True
        else:
            return False
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.10.1	Test Case T1.1.10.2
Coverage Item	Tcover1.1.10.1	Tcover1.1.10.2
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_opened	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_opening_door
Expected Output	True. elevator.current_state == ElevatorState.stopped_door_closed	True. elevator.current_state == ElevatorState.stopped_door_closed
	Test Case T1.1.10.3	
Coverage Item	Tcover1.1.10.3	
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_closed	
Expected Output	False. elevator.current_state == ElevatorState.up	

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

T1.1.11: Test checkOpen()

```
def checkOpen(self):
    open1 = self.elevator.current_state == ElevatorState.stopped_opening_door
    open2 = self.elevator.current_state == ElevatorState.stopped_door_opened
    return open1 or open2
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.11.1	Test Case T1.1.11.2
Coverage Item	Tcover1.1.11.1	Tcover1.1.11.2
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_closed	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.current_state = ElevatorState.stopped_door_opened
Expected Output	False	True

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

T1.1.12: Test checkArrive()

```
def checkArrive(self):
    floor = self.elevator.current_floor
    floor = 0 if (floor == -1) else floor
    floors = self.target_floor
    floors_up = self.target_floor_up
    floors_down = self.target_floor_down
    flag = floors[floor]
    floors[floor] = False
    if floors_up[floor]:
        flag = True
        floors_up[floor] = False
    elif floors_down[floor]:
        flag = True
        floors_down[floor] = False
    return flag
```

- Coverage Criteria: Branch coverage

- Test case

	Test Case T1.1.12.1	Test Case T1.1.12.2
Coverage Item	Tcover1.1.12.1	Tcover1.1.12.2
Input		"close_door"
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.floor = 1 elevator.direction = DirectionState.idle processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.floor = 1 elevator.direction = DirectionState.idle processor.target_floor = [False, True, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, False, False, False]
Expected Output	False	True
	Test Case T1.1.12.3	Test Case T1.1.12.4
Coverage Item	Tcover1.1.12.3	Tcover1.1.12.4
Input		
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.floor = 1 elevator.direction = DirectionState.up processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, True, False, False] processor.target_floor_down = [False, False, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.floor = 1 elevator.direction = DirectionState.down processor.target_floor = [False, False, False, False] processor.target_floor_up = [False, False, False, False] processor.target_floor_down = [False, True, False, False]
Expected Output	True	True

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

T1.1.13: Test compute_callup_time(floor)

```
    def compute_callup_time(self, floor):
        curr_floor = self.elevator.current_floor
        direction = self.elevator.direction
        floor = 0 if (floor == -1) else floor
        curr_floor = 0 if (curr_floor == -1) else curr_floor
        if direction == DirectionState.idle:
            return abs(curr_floor - floor)
        elif direction == DirectionState.down:
            min_floor = curr_floor
            for i in range(curr_floor):
                if self.target_floor[i] or self.target_floor_down[i] or
self.target_floor_up[i]:
                    min_floor = i
                    break
            return abs(floor - min_floor) + (curr_floor - min_floor)
        elif direction == DirectionState.up:
            if floor >= curr_floor:
                return floor - curr_floor
            else:
                max_floor = curr_floor
                for i in range(curr_floor, 4):
                    if self.target_floor[i] or self.target_floor_down[i] or
self.target_floor_up[i]:
                        max_floor = i
                return (max_floor - curr_floor) + (max_floor - floor)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.13.1	Test Case T1.1.13.2
Coverage Item	Tcover1.1.13.1	Tcover1.1.13.2
Input	2	3
State	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.direction = DirectionState.idle elevator.current_floor = 1 processor.target_floor = [False, False, False, False]	processor = SystemProcessor().elevator_processors[0] elevator = processor.elevator elevator.direction = DirectionState.down elevator.current_floor = 2 processor.target_floor = [True, False, False, False]
Expected Output	1	5
	Test Case T1.1.13.3	Test Case T1.1.13.4

Coverage Item	Tcover1.1.13.3	Tcover1.1.13.4
Input	2	-1
State	processor = SystemProcessor().elevator _processors[0] elevator = processor.elevator elevator.direction = DirectionState.up elevator.current_floor = 1 processor.target_floor = [True, False, False, False]	processor = SystemProcessor().elevator _processors[0] elevator = processor.elevator elevator.direction = DirectionState.up elevator.current_floor = 1 processor.target_floor = [False, False, False, True]
Expected Output	1	5

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

T1.1.14: Test compute_calldown_time(floor)

```
    def compute_calldown_time(self, floor):
        curr_floor = self.elevator.current_floor
        direction = self.elevator.direction
        floor = 0 if (floor == -1) else floor
        curr_floor = 0 if (curr_floor == -1) else curr_floor
        if direction == DirectionState.idle:
            return abs(curr_floor - floor)
        elif direction == DirectionState.down:
            if floor <= curr_floor:
                return curr_floor - floor
            else:
                min_floor = curr_floor
                for i in range(curr_floor):
                    if self.target_floor[i] or self.target_floor_down[i] or
self.target_floor_up[i]:
                        min_floor = i
                        break
                return abs(floor - min_floor) + (curr_floor - min_floor)
        elif direction == DirectionState.up:
            max_floor = curr_floor
            for i in range(curr_floor, 4):
                    if self.target_floor[i] or self.target_floor_down[i] or
self.target_floor_up[i]:
                        max_floor = i
            return (max_floor - curr_floor) + abs(max_floor - floor)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.1.14.1	Test Case T1.1.14.2
Coverage Item	Tcover1.1.14.1	Tcover1.1.14.2
Input	2	3
State	processor = SystemProcessor().elevator _processors[0] elevator = processor.elevator elevator.direction = DirectionState.idle elevator.current_floor = 1 processor.target_floor = [False, False, False, False]	processor = SystemProcessor().elevator _processors[0] elevator = processor.elevator elevator.direction = DirectionState.down elevator.current_floor = 2 processor.target_floor = [True, False, False, False]
Expected Output	1	5
	Test Case T1.1.14.3	Test Case T1.1.14.4
Coverage Item	Tcover1.1.14.3	Tcover1.1.14.4
Input	1	-1
State	processor = SystemProcessor().elevator _processors[0] elevator = processor.elevator elevator.direction = DirectionState.down elevator.current_floor = 2 processor.target_floor = [True, False, False, False]	processor = SystemProcessor().elevator _processors[0] elevator = processor.elevator elevator.direction = DirectionState.up elevator.current_floor = 1 processor.target_floor = [False, False, False, True]
Expected Output	1	5

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

T1.2: SystemProcessor Unit Test

T1.2.1: Test update()

```
def update(self):
    self.elevator_processors[0].update()
    self.elevator_processors[1].update()
```

- Coverage Criteria: Branch coverage

- Test case

	Test Case T1.2.1.1
Coverage Item	Tcover1.2.1.1
Input	
State	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1]
Expected Output	processor1.update(), processor2.update() should be called.

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

T1.2.2: Test process_ExternalUI_requests(ExterUI_MSG)

```
    def process_ExternalUI_requests(self, ExterUI_MSG = ""):
        ele_processors = self.elevator_processors
        call_floor = int(ExterUI_MSG.split("@")[1])
        call_floor = 0 if (call_floor == -1) else call_floor
        if ExterUI_MSG.startswith("call_up"):
            if ele_processors[0].target_floor_up[call_floor] or
ele_processors[1].target_floor_up[call_floor]:
                return
            if ele_processors[0].target_floor_down[call_floor]:
                ele_processors[1].target_floor_up[call_floor] = True
                return
            elif ele_processors[1].target_floor_down[call_floor]:
                ele_processors[0].target_floor_up[call_floor] = True
                return
            arrive_time1, arrive_time2 = self.getUpTime(call_floor)
            if arrive_time1 < arrive_time2:
                ele_id = 0
            else:
                ele_id = 1
            ele_processors[ele_id].target_floor_up[call_floor] = True
        elif ExterUI_MSG.startswith("call_down"):
            if ele_processors[0].target_floor_down[call_floor] or
ele_processors[1].target_floor_down[call_floor]:
                return
            if ele_processors[0].target_floor_up[call_floor]:
                ele_processors[1].target_floor_down[call_floor] = True
                return
            elif ele_processors[1].target_floor_up[call_floor]:
                ele_processors[0].target_floor_down[call_floor] = True
                return
```

```
arrive_time1, arrive_time2 = self.getDownTime(call_floor)
if arrive_time1 < arrive_time2:
    ele_id = 0
else:
    ele_id = 1
ele_processors[ele_id].target_floor_down[call_floor] = True
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.2.2.1	Test Case T1.2.2.2
Coverage Item	Tcover1.2.2.1	Tcover1.2.2.2
Input	"call_up@2"	"call_up@2"
State	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.target_floor_up[2] = True processor2.target_floor_up[2] = False processor1.target_floor_down[2] = False processor2.target_floor_down[2] = False	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.target_floor_up[2] = False processor2.target_floor_up[2] = False processor1.target_floor_down[2] = True processor2.target_floor_down[2] = False
Expected Output	processor1.target_floor_up[2] == True, processor2.target_floor_up[2] == False	processor1.target_floor_up[2] == False, processor2.target_floor_up[2] == True
	Test Case T1.2.2.3	Test Case T1.2.2.4
Coverage Item	Tcover1.2.2.3	Tcover1.2.2.4
Input	"call_up@3"	"call_down@-1"
State	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.target_floor_up[3] = False	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.target_floor_up[0] = True

	processor2.target_floor_up[3] = False processor1.target_floor_down[3] = False processor2.target_floor_down[3] = False system_processor.getUpTime = MagicMock(return_value=(5, 10))	processor2.target_floor_up[0] = False processor1.target_floor_down[0] = False processor2.target_floor_down[0] = False
Expected Output	processor1.target_floor_up[3] == True, processor2.target_floor_up[3] == False	processor1.target_floor_up[0] == True, processor2.target_floor_up[0] == False
	Test Case T1.2.2.5	Test Case T1.2.2.6
Coverage Item	Tcover1.2.2.5	Tcover1.2.2.6
Input	"call_down@3"	"call_down@3"
State	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.target_floor_down[3] = True processor2.target_floor_down[3] = False processor1.target_floor_up[3] = False processor2.target_floor_up[3] = False	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.target_floor_down[3] = False processor2.target_floor_down[3] = False processor1.target_floor_up[3] = True processor2.target_floor_up[3] = False
Expected Output	processor1.target_floor_down[3] == True, processor2.target_floor_down[3] == False	processor1.target_floor_down[3] == False, processor2.target_floor_down[3] == True
	Test Case T1.2.2.7	Test Case T1.2.2.8
Coverage Item	Tcover1.2.2.7	Tcover1.2.2.8
Input	"call_down@2"	"call_down@-1"
State	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1]	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1]

	processor1.target_floor_down[2] = False processor2.target_floor_down[2] = False processor1.target_floor_up[2] = False processor2.target_floor_up[2] = False system_processor.getDownTime = MagicMock(return_value=(5, 3))	processor1.target_floor_down[0] = True processor2.target_floor_down[0] = False processor1.target_floor_up[0] = False processor2.target_floor_up[0] = False
Expected Output	processor1.target_floor_down[2] == False, processor2.target_floor_down[2] == True	processor1.target_floor_down[0] == True, processor2.target_floor_down[0] == False

- Test coverage: 8 / 8 = 100%
- Test result: 8 passed

T1.2.3: Test receive_eleProcessor_MSG(message)

```
    def receive_eleProcessor_MSG(self, message):
        print(f"System Processor received update: {message}")
        '''Deal with
f"{moving_direction}_floor_{arrive_floor}_arrived#{self.elevator.ele_id}"'''
        ele_processor = self.elevator_processors
        if message.startswith("up_floor") or message.startswith("down_floor"):
            ele_id = int(message.split("#")[1])
            ele_processor[ele_id - 1].elevator.current_state =
ElevatorState.stopped_door_closed
            ele_processor[ele_id - 1].open_door()
        elif message.startswith("floor_"):
            ele_id = int(message.split("#")[1])
            ele_processor[ele_id - 1].elevator.current_state =
ElevatorState.stopped_door_closed
            ele_processor[ele_id - 1].open_door()
        for ele_processor in ele_processor:
            if(ele_processor.checkOpen()):
                print(f"System Processor received update:
door_opened#{ele_processor.elevator.ele_id}")
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.2.3.1	Test Case T1.2.3.2
Coverage Item	Tcover1.2.3.1	Tcover1.2.3.2
Input	"up_floor_2_arrived#1"	"floor_1_arrived#2"

State	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.checkOpen = MagicMock(return_value=True)	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor2.checkOpen = MagicMock(return_value=True)
Expected Output	processor1.elevator.current_state == ElevatorState.stopped_door_closed processor1.open_door() should be called.	processor2.elevator.current_state == ElevatorState.stopped_door_closed processor2.open_door() should be called.

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

T1.2.4: Test getUpTime(floor)

```
    def getUpTime(self, floor):
        ele_processors = self.elevator_processors
        return ele_processors[0].compute_callup_time(floor),
ele_processors[1].compute_callup_time(floor)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.2.4.1
Coverage Item	Tcover1.2.4.1
Input	2
State	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.compute_callup_time = MagicMock(return_value=5) processor2.compute_callup_time = MagicMock(return_value=10)
Expected Output	(5, 10)

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

T1.2.5: Test getDownTime(floor)

```
    def getDownTime(self, floor):
        ele_processors = self.elevator_processors
        return ele_processors[0].compute_calldown_time(floor),
ele_processors[1].compute_calldown_time(floor)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.2.5.1
Coverage Item	Tcover1.2.5.1
Input	3
State	system_processor = SystemProcessor() processor1 = system_processor.elevator_processors[0] processor2 = system_processor.elevator_processors[1] processor1.compute_calldown_time = MagicMock(return_value=8) processor2.compute_calldown_time = MagicMock(return_value=6)
Expected Output	(8, 6)

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

T1.3: ExternalUI Unit Test

T1.3.1: Test update()

```
    def update(self):
        self.update_time()
        self.update_floor()
        self.update_state()
        self.update_direction()
        self.update_button()
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.1.1
Coverage Item	Tcover1.3.1.1
Input	
State	ui = ExternalUI(2, SystemProcessor())
Expected Output	ui.update_time(), ui.update_floor(), ui.update_state(), ui.update_direction(), ui.update_button() should be called.

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

T1.3.2: Test update_time()

```
def update_time(self):
    current_time = QDateTime.currentDateTime().toString('hh:mm:ss')
    self.time_label.setText(f'Time: {current_time}')
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.2.1
Coverage Item	Tcover1.3.2.1
Input	
State	ui = ExternalUI(2, SystemProcessor())
Expected Output	ui.time_label.text() == f'Time: {QDateTime.currentDateTime().toString('hh:mm:ss')}'

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

T1.3.3: Test update_state()

```
def update_state(self):
    ele_processors = self.processor.elevator_processors
    if self.checkOpen(ele_processors[0]):
        self.elevator_1_open_indicator.setText("<|>")
        self.elevator_1_open_indicator.setStyleSheet("color:green;")
    else:
        self.elevator_1_open_indicator.setText(">|<")
        self.elevator_1_open_indicator.setStyleSheet("color:black;")

    if self.checkOpen(ele_processors[1]):
        self.elevator_2_open_indicator.setText("<|>")
        self.elevator_2_open_indicator.setStyleSheet("color:green;")
    else:
        self.elevator_2_open_indicator.setText(">|<")
        self.elevator_2_open_indicator.setStyleSheet("color:black;")
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.3.1	Test Case T1.3.3.2
Coverage Item	Tcover1.3.3.1	Tcover1.3.3.2
Input		
State	ui = ExternalUI(2, SystemProcessor()) ui.checkOpen.return_value = True	ui = ExternalUI(2, SystemProcessor())

		ui.checkOpen.return_value = False
Expected Output	ui.elevator_1_open_indicator.text() == "<|>", ui.elevator_2_open_indicator.text() == "<|>", ui.elevator_1_open_indicator.styleSheet() == "color:green;", ui.elevator_2_open_indicator.styleSheet() == "color:green;"	ui.elevator_1_open_indicator.text() == ">|<", ui.elevator_2_open_indicator.text() == ">|<", ui.elevator_1_open_indicator.styleSheet() == "color:black;", ui.elevator_2_open_indicator.styleSheet() == "color:black;"

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

T1.3.4: Test update_floor()

```
    def update_floor(self):
        ele_processors = self.processor.elevator_processors
        self.elevator_1_floor_label.setText(f'Floor
{ele_processors[0].elevator.current_floor}')
        self.elevator_2_floor_label.setText(f'Floor
{ele_processors[1].elevator.current_floor}')
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.4.1
Coverage Item	Tcover1.3.4.1
Input	
State	ui = ExternalUI(2, SystemProcessor()) ui.processor.elevator_processors[0].elevator.current_floor = 3 ui.processor.elevator_processors[1].elevator.current_floor = 1
Expected Output	ui.elevator_1_floor_label.text() == 'Floor 3', ui.elevator_2_floor_label.text() == 'Floor 1'

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

T1.3.5: Test update_direction()

```
    def update_direction(self):
        ele_processors = self.processor.elevator_processors
        direction1 = ele_processors[0].elevator.direction
        if direction1 == DirectionState.up:
            self.elevator_1_up_indicator.setStyleSheet(on)
            self.elevator_1_down_indicator.setStyleSheet(off)
```

```
        elif direction1 == DirectionState.down:
            self.elevator_1_up_indicator.setStyleSheet(off)
            self.elevator_1_down_indicator.setStyleSheet(on)
        elif direction1 == DirectionState.idle:
            self.elevator_1_up_indicator.setStyleSheet(off)
            self.elevator_1_down_indicator.setStyleSheet(off)

        direction2 = ele_processors[1].elevator.direction
        if direction2 == DirectionState.up:
            self.elevator_2_up_indicator.setStyleSheet(on)
            self.elevator_2_down_indicator.setStyleSheet(off)
        elif direction2 == DirectionState.down:
            self.elevator_2_up_indicator.setStyleSheet(off)
            self.elevator_2_down_indicator.setStyleSheet(on)
        elif direction2 == DirectionState.idle:
            self.elevator_2_up_indicator.setStyleSheet(off)
            self.elevator_2_down_indicator.setStyleSheet(off)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.5.1	Test Case T1.3.5.2
Coverage Item	Tcover1.3.5.1	Tcover1.3.5.2
Input		
State	ui = ExternalUI(2, SystemProcessor()) ui.processor.elevator_processors[0].elevator.direction = DirectionState.idle ui.processor.elevator_processors[1].elevator.direction = DirectionState.idle	ui = ExternalUI(2, SystemProcessor()) ui.processor.elevator_processors[0].elevator.direction = DirectionState.up ui.processor.elevator_processors[1].elevator.direction = DirectionState.up
Expected Output	ui.elevator_1_up_indicator.styleSheet() == off, ui.elevator_1_down_indicator.styleSheet() == off, ui.elevator_2_up_indicator.styleSheet() == off, ui.elevator_2_down_indicator.styleSheet() == off	ui.elevator_1_up_indicator.styleSheet() == on, ui.elevator_1_down_indicator.styleSheet() == off, ui.elevator_2_up_indicator.styleSheet() == on, ui.elevator_2_down_indicator.styleSheet() == off
	Test Case T1.3.5.3	
Coverage Item	Tcover1.3.5.3	
Input		
State	ui = ExternalUI(2, SystemProcessor())	

	ui.processor.elevator_processors[0].elevator.direction = DirectionState.down ui.processor.elevator_processors[1].elevator.direction = DirectionState.down	
Expected Output	ui.elevator_1_up_indicator.styleSheet() == off, ui.elevator_1_down_indicator.styleSheet() == on, ui.elevator_2_up_indicator.styleSheet() == off, ui.elevator_2_down_indicator.styleSheet() == on	

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

T1.3.6: Test update_button()

```
def update_button(self):
    if self.checkTargetUp(self.floor):
        self.up_button.setStyleSheet(circle_button_style_on)
    else:
        self.up_button.setStyleSheet(circle_button_style)

    if self.checkTargetDown(self.floor):
        self.down_button.setStyleSheet(circle_button_style_on)
    else:
        self.down_button.setStyleSheet(circle_button_style)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.6.1	Test Case T1.3.6.2
Coverage Item	Tcover1.3.6.1	Tcover1.3.6.2
Input		
State	ui = ExternalUI(2, SystemProcessor()) ui.checkTargetUp.return_value = False ui.checkTargetDown.return_value = False	ui = ExternalUI(2, SystemProcessor()) ui.checkTargetUp.return_value = True ui.checkTargetDown.return_value = True
Expected Output	ui.up_button.styleSheet() == circle_button_style,	ui.up_button.styleSheet() == circle_button_style_on,

	ui.down_button.styleSheet() == circle_button_style	ui.down_button.styleSheet() == circle_button_style_on

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

T1.3.7: Test push_up_button()

```
def push_up_button(self):
    self.processor.process_ExternalUI_requests(f"call_up@{self.floor}")
    self.up_button.setStyleSheet(circle_button_style_on)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.7.1
Coverage Item	Tcover1.3.7.1
Input	
State	ui = ExternalUI(2, SystemProcessor())
Expected Output	ui.up_button.styleSheet() == circle_button_style_on ui.processor.process_ExternalUI_requests(f"call_up@{ui.floor}") should be called.

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

T1.3.8: Test push_down_button()

```
def push_down_button(self):
    self.processor.process_ExternalUI_requests(f"call_down@{self.floor}")
    self.down_button.setStyleSheet(circle_button_style_on)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.8.1
Coverage Item	Tcover1.3.8.1
Input	
State	ui = ExternalUI(2, SystemProcessor())
Expected Output	ui.down_button.styleSheet() == circle_button_style_on ui.processor.process_ExternalUI_requests(f"call_down@{ui.floor}") should be called.

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

T1.3.9: Test checkOpen(processor)

```
    def checkOpen(self, processor):
        state = processor.elevator.current_state
        door_open = state == ElevatorState.stopped_door_opened or state ==
ElevatorState.stopped_opening_door
        same_floor = processor.elevator.current_floor == self.floor
        return door_open and same_floor
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.9.1	Test Case T1.3.9.2
Coverage Item	Tcover1.3.9.1	Tcover1.3.9.2
Input	processor = ui.processor.elevator_processors[0]	processor = ui.processor.elevator_processors[0]
State	ui = ExternalUI(2, SystemProcessor()) processor.elevator.current_floor = 2 processor.elevator.current_state = ElevatorState.stopped_door_opened	ui = ExternalUI(2, SystemProcessor()) processor.elevator.current_floor = 2 processor.elevator.current_state = ElevatorState.stopped_door_closed
Expected Output	True	False

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

T1.3.10: Test checkTargetUp(floor)

```
    def checkTargetUp(self, floor):
        floor = 0 if (floor == -1) else floor
        ele_processors = self.processor.elevator_processors
        return ele_processors[0].target_floor_up[floor] or
ele_processors[1].target_floor_up[floor]
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.10.1	Test Case T1.3.10.2
Coverage Item	Tcover1.3.10.1	Tcover1.3.10.2
Input	1	1
State	ui = ExternalUI(2, SystemProcessor()) processor = ui.processor processor.elevator_processors[0].target_floor_up = [False, True, False, False]	ui = ExternalUI(2, SystemProcessor()) processor = ui.processor processor.elevator_processors[0].target_floor_up = [False, False, False, False]

	processor.elevator_processors[1].target_floor_up = [False, False, False, False]	processor.elevator_processors[1].target_floor_up = [False, False, False, False]
Expected Output	True	False
	Test Case T1.3.10.3	
Coverage Item	Tcover1.3.10.3	
Input	-1	
State	ui = ExternalUI(2, SystemProcessor()) processor = ui.processor processor.elevator_processors[0].target_floor_up = [False, False, False, False] processor.elevator_processors[1].target_floor_up = [True, False, False, False]	
Expected Output	True	

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

T1.3.11: Test checkTargetDown(floor)

```
    def checkTargetDown(self, floor):
        floor = 0 if (floor == -1) else floor
        ele_processors = self.processor.elevator_processors
        return ele_processors[0].target_floor_down[floor] or
ele_processors[1].target_floor_down[floor]
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.3.11.1	Test Case T1.3.11.2
Coverage Item	Tcover1.3.11.1	Tcover1.3.11.2
Input	1	1
State	ui = ExternalUI(2, SystemProcessor()) processor = ui.processor processor.elevator_processors[0].target_floor_down = [False, True, False, False] processor.elevator_processors[1].target_floor_down = [False, False, False, False]	ui = ExternalUI(2, SystemProcessor()) processor = ui.processor processor.elevator_processors[0].target_floor_down = [False, False, False, False] processor.elevator_processors[1].target_floor_down = [False, False, False, False]
Expected Output	True	False
	Test Case T1.3.11.3	

Coverage Item	Tcover1.3.11.3	
Input	-1	
State	ui = ExternalUI(2, SystemProcessor()) processor = ui.processor processor.elevator_processors[0].target_floor_down = [False, False, False, False] processor.elevator_processors[1].target_floor_down = [True, False, False, False]	
Expected Output	True	

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

T1.4: InternalUI Unit Test

T1.4.1: Test update()

```
def update(self):
    self.update_time()
    self.update_state()
    self.update_floor()
    self.update_direction()
    self.update_floor_button()
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.1.1
Coverage Item	Tcover1.4.1.1
Input	
State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.update_time(), ui.update_state(), ui.update_floor(), ui.update_direction(), ui.update_floor_button() should be called.

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

T1.4.2: Test update_time()

```
def update_time(self):
    current_time = QDateTime.currentDateTime().toString('hh:mm:ss')
    self.time_label.setText(f"Time: {current_time}")
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.2.1
Coverage Item	Tcover1.4.2.1
Input	
State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.time_label.text() == f'Time: {QDateTime.currentDateTime().toString('hh:mm:ss')}'

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

T1.4.3: Test update_state()

```
def update_state(self):
    if self.processor.checkOpen():
        self.open_close_state.setText("<|>")
        self.open_close_state.setStyleSheet("color:green;")
    else:
        self.open_close_state.setText(">|<")
        self.open_close_state.setStyleSheet("color:black;")
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.3.1	Test Case T1.4.3.2
Coverage Item	Tcover1.4.3.1	Tcover1.4.3.2
Input		
State	ui = InternalUI(SystemProcessor().elevator_processors[0]) ui.processor.checkOpen.return_value = True	ui = InternalUI(SystemProcessor().elevator_processors[0]) ui.processor.checkOpen.return_value = False
Expected Output	ui.open_close_state.text() == "<|>", ui.open_close_state.styleSheet() == "color:green;"	ui.open_close_state.text() == ">|<", ui.open_close_state.styleSheet() == "color:black;"

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

T1.4.4: Test update_floor()

```
    def update_floor(self):
        self.floor_label.setText(f"Floor:
{self.processor.elevator.current_floor}")
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.4.1
Coverage Item	Tcover1.4.4.1
Input	
State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.floor_label.text() == f'Floor: {ui.processor.elevator.current_floor}'

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

T1.4.5: Test update_floor_button()

```
    def update_floor_button(self):
        if self.processor.target_floor[0]:
            self.floor_button_b.setStyleSheet(circle_button_style_on)
        else:
            self.floor_button_b.setStyleSheet(circle_button_style)

        if self.processor.target_floor[1]:
            self.floor_button_1.setStyleSheet(circle_button_style_on)
        else:
            self.floor_button_1.setStyleSheet(circle_button_style)

        if self.processor.target_floor[2]:
            self.floor_button_2.setStyleSheet(circle_button_style_on)
        else:
            self.floor_button_2.setStyleSheet(circle_button_style)

        if self.processor.target_floor[3]:
            self.floor_button_3.setStyleSheet(circle_button_style_on)
        else:
            self.floor_button_3.setStyleSheet(circle_button_style)
```

- Coverage Criteria: Branch coverage

- Test case

	Test Case T1.4.5.1	Test Case T1.4.5.2
Coverage Item	Tcover1.4.5.1	Tcover1.4.5.2
Input		
State	ui = InternalUI(SystemProcessor().elevator_processors[0]) ui.processor.target_floor = [False, False, False, False]	ui = InternalUI(SystemProcessor().elevator_processors[0]) ui.processor.target_floor = [True, True, True, True]
Expected Output	ui.floor_button_b.styleSheet() == circle_button_style, ui.floor_button_1.styleSheet() == circle_button_style, ui.floor_button_2.styleSheet() == circle_button_style, ui.floor_button_3.styleSheet() == circle_button_style	ui.floor_button_b.styleSheet() == circle_button_style_on, ui.floor_button_1.styleSheet() == circle_button_style_on, ui.floor_button_2.styleSheet() == circle_button_style_on, ui.floor_button_3.styleSheet() == circle_button_style_on

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

T1.4.6: Test update_direction()

```
def update_direction(self):
    direction = self.processor.elevator.direction
    if direction == DirectionState.up:
        self.direction_label_up.setStyleSheet(on)
        self.direction_label_down.setStyleSheet(off)
    elif direction == DirectionState.down:
        self.direction_label_up.setStyleSheet(off)
        self.direction_label_down.setStyleSheet(on)
    elif direction == DirectionState.idle:
        self.direction_label_up.setStyleSheet(off)
        self.direction_label_down.setStyleSheet(off)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.6.1	Test Case T1.4.6.2
Coverage Item	Tcover1.4.6.1	Tcover1.4.6.2
Input		
State	ui = InternalUI(SystemProcessor().elevator_processors[0])	ui = InternalUI(SystemProcessor().elevator_processors[0])

	ui.processor.elevator.direction = DirectionState.idle	ui.processor.elevator.direction = DirectionState.up
Expected Output	ui.direction_label_up.styleSheet() == off, ui.direction_label_down.styleSheet() == off	ui.direction_label_up.styleSheet() == on, ui.direction_label_down.styleSheet() == off
	Test Case T1.4.6.3	
Coverage Item	Tcover1.4.6.3	
Input		
State	ui = InternalUI(SystemProcessor().elevator_processors[0]) ui.processor.elevator.direction = DirectionState.down	
Expected Output	ui.direction_label_up.styleSheet() == off, ui.direction_label_down.styleSheet() == on	

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

T1.4.7: Test push_open_door_button()

```
def push_open_door_button(self):
    self.processor.process_InternalUI_requests("open_door")
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.7.1
Coverage Item	Tcover1.4.7.1
Input	
State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.processor.process_InternalUI_requests("open_door") should be called.

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

T1.4.8: Test push_close_door_button()

```
def push_close_door_button(self):
    self.processor.process_InternalUI_requests("close_door")
```

- Coverage Criteria: Branch coverage

- Test case

	Test Case T1.4.8.1
Coverage Item	Tcover1.4.8.1
Input	
State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.processor.process_InternalUI_requests("close_door") should be called.

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

T1.4.9: Test push_floor_button_b()

```
    def push_floor_button_b(self):
        self.processor.process_InternalUI_requests(f"select_floor@-
1#{self.processor.elevator.ele_id}")
        self.floor_button_b.setStyleSheet(circle_button_style_on)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.9.1
Coverage Item	Tcover1.4.9.1
Input	
State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.floor_button_b.styleSheet() == circle_button_style_on ui.processor.process_InternalUI_requests(f"select_floor@-1#{ui.processor.elevator.ele_id}") should be called.

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

T1.4.10: Test push_floor_button_1()

```
    def push_floor_button_1(self):
        self.processor.process_InternalUI_requests(f"select_floor@1#{self.process
or.elevator.ele_id}")
        self.floor_button_1.setStyleSheet(circle_button_style_on)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.10.1
Coverage Item	Tcover1.4.10.1
Input	

State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.floor_button_1.styleSheet() == circle_button_style_on ui.processor.process_InternalUI_requests(f"select_floor@1#{ui.processor.elevator.ele_id}") should be called.

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

T1.4.11: Test push_floor_button_2()

```
    def push_floor_button_2(self):
        self.processor.process_InternalUI_requests(f"select_floor@2#{self.process
or.elevator.ele_id}")
        self.floor_button_2.setStyleSheet(circle_button_style_on)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.11.1
Coverage Item	Tcover1.4.11.1
Input	
State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.floor_button_2.styleSheet() == circle_button_style_on ui.processor.process_InternalUI_requests(f"select_floor@2#{ui.processor.elevator.ele_id}") should be called.

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

T1.4.12: Test push_floor_button_3()

```
    def push_floor_button_3(self):
        self.processor.process_InternalUI_requests(f"select_floor@3#{self.process
or.elevator.ele_id}")
        self.floor_button_3.setStyleSheet(circle_button_style_on)
```

- Coverage Criteria: Branch coverage
- Test case

	Test Case T1.4.12.1
Coverage Item	Tcover1.4.12.1
Input	
State	ui = InternalUI(SystemProcessor().elevator_processors[0])
Expected Output	ui.floor_button_3.styleSheet() == circle_button_style_on ui.processor.process_InternalUI_requests(f"select_floor@3#{ui.processor.elevator.ele_id}") should be called.

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

T2: Integration Test

This section provides information of integration tests we made for the Elevator System. You can find executable files in the corresponding files.

T2.1: ElevatorProcessor + 2 InternalUI Integration

This section tests the integration between the elevator processor and its corresponding internal UI.

T2.1.1: Test Open Door Button

```
    def test_open_door_button(self):
        """Test the internal open door button of Elevator 1."""
        self.processor.elevator.current_state = ElevatorState.stopped_door_closed
        self.internalUI.open_door_button.click()
        self.processor.update()
        self.internalUI.update()
        self.assertTrue(self.processor.checkOpen())
        self.assertEqual(self.internalUI.open_close_state.text(), "<|>")
        self.assertEqual(self.internalUI.open_close_state.styleSheet(),
"color:green;")
        QTimer.singleShot(2500, self.verify_door_closed)

    def verify_door_closed(self):
        self.assertEqual(self.processor.elevator.current_state,
ElevatorState.stopped_door_closed)
        self.assertEqual(self.internalUI.open_close_state.text(), ">|<")
        self.assertEqual(self.internalUI.open_close_state.styleSheet(),
"color:black;")
```

- Test case

	Test Case T2.1.1.1
Coverage Item	Tcover1.1.5, Tcover1.1.7, Tcover1.1.8, Tcover1.1.9, Tcover1.1.11, Tcover1.4.3
Input	Press internal open door button of Elevator 1.
State	Elevator 1 door is closed.
Expected Output	Elevator 1 door opens, then if the button is released, the door will close in some time.

- Test coverage: 6 / 6 = 100%
- Test result: 1 passed

T2.1.2: Test Close Door Button

```
    def test_close_door_button(self):
        """Test the internal close door button of Elevator 1."""
        self.processor.elevator.current_state = ElevatorState.stopped_door_opened
        self.internalUI.close_door_button.click()
        self.assertEqual(self.processor.elevator.current_state,
ElevatorState.stopped_door_closed)
        self.assertEqual(self.internalUI.open_close_state.text(), ">|<")
```

- Test case

	Test Case T2.1.2.1
Coverage Item	Tcover1.1.5, Tcover1.1.7, Tcover1.1.8, Tcover1.1.10, Tcover1.1.11, Tcover1.4.3
Input	Press internal close door button of Elevator 1.
State	Elevator 1 door is open.
Expected Output	Elevator 1 door closes at once.

- Test coverage: 6 / 6 = 100%
- Test result: 1 passed

T2.1.3: Test Internal Floor Button & Elevator Move

```
    def test_floor_buttons(self):
        """Test the floor buttons of Elevator 1."""
        self.processor.elevator.current_floor = 1
        self.internalUI.floor_button_3.click()
        self.assertEqual(self.internalUI.floor_button_3.styleSheet(),
circle_button_style_on)
        self.internalUI.floor_button_1.click()
        self.assertEqual(self.internalUI.floor_button_1.styleSheet(),
circle_button_style_on)
        QTimer.singleShot(2500, self.verify_floor)

    def verify_floor(self):
        self.assertEqual(self.processor.elevator.current_floor, 3)
        self.assertEqual(self.internalUI.floor_label.text(), "Floor: 3")
        self.assertEqual(self.internalUI.floor_button_3.styleSheet(),
circle_button_style)
        self.assertEqual(self.internalUI.floor_button_1.styleSheet(),
circle_button_style)
```

- Test case

	Test Case T2.1.3.1
Coverage Item	Tcover1.1.4, Tcover1.1.6, Tcover1.1.7, Tcover1.1.8, Tcover1.1.12, TcoverT1.1.13, T1.1.14, Tcover1.2.1, Tcover1.2.2, Tcover1.2.3, Tcover1.2.4, Tcover1.2.5, Tcover1.4.3, Tcover1.4.4, Tcover1.4.10, Tcover1.4.12
Input	Press floor 3 button in Elevator 1, then press floor 1 button.
State	Elevator 1 stops on floor 1.
Expected Output	Elevator 1 goes up and ignore the requirement to go to floor 1.

  - Test coverage: 16 / 16 = 100%
  - Test result: 1 passed

T2.2: ElevatorProcessor + 2 InternalUI + 4 ExternalUI Integration

This section tests the integration between the elevator processor and all the UI, i.e. internal UI and external UI.

```
class TestElevatorProcessor_InternalUI_ExternalUI(unittest.TestCase):

    @classmethod
    def setUpClass(cls):
        """Initialize the test environment and set class-level variables."""
        cls.app = QApplication(sys.argv)


    def setUp(self):
        """Initialization of each test case."""
        self.system_processor = SystemProcessor()
        self.processor1 = self.system_processor.elevator_processors[0]
        self.processor2 = self.system_processor.elevator_processors[1]
        self.externalUIb = ExternalUI(-1, self.system_processor)
        self.externalUI1 = ExternalUI(1, self.system_processor)
        self.externalUI2 = ExternalUI(2, self.system_processor)
        self.externalUI3 = ExternalUI(3, self.system_processor)
        self.internalUI1 = InternalUI(self.processor1)
        self.internalUI2 = InternalUI(self.processor2)
        self.externalUIb.show()
        self.externalUI1.show()
        self.externalUI2.show()
        self.externalUI3.show()
        self.internalUI1.show()
        self.internalUI2.show()


    def tearDown(self):
        """Cleanup after each test case."""
```

```
        self.externalUIb.close()
        self.externalUI1.close()
        self.externalUI2.close()
        self.externalUI3.close()
        self.internalUI1.close()
        self.internalUI2.close()


    @classmethod
    def tearDownClass(cls):
        """Cleanup work after all test cases are executed."""
        cls.app.quit()

    def update(self):
        '''Simulate the update of the processor and UI.'''
        self.system_processor.update()
        self.externalUIb.update()
        self.externalUI1.update()
        self.externalUI2.update()
        self.externalUI3.update()
        self.internalUI1.update()
        self.internalUI2.update()

    def test_elevator_integration(self):
        """Test the full integration scenario with detailed steps."""

        # T2.2.1: Press down button outside on floor 2.
        self.externalUI2.down_button.click()
        while self.processor2.elevator.current_floor != 2:
            self.update()
        # T2.2.1. Expected Output: Elevator 2 is called and the door will open
when it arrives.
        self.assertEqual(self.processor2.elevator.current_floor, 2)
        self.assertTrue(self.processor2.checkOpen())
        self.assertEqual(self.internalUI2.open_close_state.text(), "<|>")
        self.assertEqual(self.internalUI2.open_close_state.styleSheet(),
"color:green;")
        self.assertEqual(self.externalUI2.elevator_2_open_indicator.text(),
"<|>")
        self.assertEqual(self.externalUI2.elevator_2_open_indicator.styleSheet(),
"color:green;")

        # T2.2.2: When the door of elevator 2 is about to close, press open door
button.
        while self.processor2.elevator.current_state != \
```

```
                (ElevatorState.stopped_closing_door or
ElevatorState.stopped_door_closed):
            self.update()
        self.internalUI2.open_door_button.click()
        self.update()
        # T2.2.2. Expected Output: The door of elevator 2 opens.
        self.assertTrue(self.processor2.checkOpen())
        self.assertEqual(self.internalUI2.open_close_state.text(), "<|>")
        self.assertEqual(self.internalUI2.open_close_state.styleSheet(),
"color:green;")
        self.assertEqual(self.externalUI2.elevator_2_open_indicator.text(),
"<|>")
        self.assertEqual(self.externalUI2.elevator_2_open_indicator.styleSheet(),
"color:green;")

        # T2.2.3: Press floor -1 button in elevator 2, then press floor 3 button.
        self.internalUI2.floor_button_b.click()
        self.update()
        self.internalUI2.floor_button_3.click()
        self.update()
        self.assertEqual(self.internalUI2.floor_button_b.styleSheet(),
circle_button_style_on)
        self.assertEqual(self.internalUI2.floor_button_3.styleSheet(),
circle_button_style_on)

        # T2.2.3. Expected Output: Elevator 2 moves to floor -1 at first.
        while self.processor2.elevator.current_floor != -1:
            self.update()
        self.assertEqual(self.internalUI2.floor_label.text(), "Floor: -1")
        self.assertEqual(self.externalUIb.elevator_2_floor_label.text(), "Floor -
1")
        self.assertEqual(self.externalUI1.elevator_2_floor_label.text(), "Floor -
1")
        self.assertEqual(self.externalUI2.elevator_2_floor_label.text(), "Floor -
1")
        self.assertEqual(self.externalUI3.elevator_2_floor_label.text(), "Floor -
1")

        # T2.2.4: When elevator 2 is on floor -1, press up button outside on
floor 2.
        self.externalUI2.up_button.click()

        # T2.2.4. Expected Output: Elevator 1 is called.
        while self.processor1.elevator.current_floor != 2:
            self.update()
```

```
        self.assertEqual(self.internalUI1.floor_label.text(), "Floor: 2")
        self.assertEqual(self.externalUIb.elevator_1_floor_label.text(), "Floor
2")
        self.assertEqual(self.externalUI1.elevator_1_floor_label.text(), "Floor
2")
        self.assertEqual(self.externalUI2.elevator_1_floor_label.text(), "Floor
2")
        self.assertEqual(self.externalUI3.elevator_1_floor_label.text(), "Floor
2")

        # T2.2.5: Press close door button of elevator 1 when door is open.
        while not self.processor1.checkOpen():
            self.update()
        self.internalUI1.push_close_door_button()
        self.update()
        # T2.2.5. Expected Output: Elevator 1 closes the door.
        self.assertEqual(self.processor1.elevator.current_state,
ElevatorState.stopped_door_closed)
        self.assertEqual(self.internalUI1.open_close_state.text(), ">|<")
        self.assertEqual(self.internalUI1.open_close_state.styleSheet(),
"color:black;")
        self.assertEqual(self.externalUI2.elevator_2_open_indicator.text(),
">|<")
        self.assertEqual(self.externalUI2.elevator_2_open_indicator.styleSheet(),
"color:black;")

        # T2.2.3. Expected Output: Elevator 1 then moves to floor 3.
        while self.processor2.elevator.current_floor != 3:
            self.update()
        self.assertEqual(self.internalUI2.floor_label.text(), "Floor: 3")
        self.assertEqual(self.externalUIb.elevator_2_floor_label.text(), "Floor
3")
        self.assertEqual(self.externalUI1.elevator_2_floor_label.text(), "Floor
3")
        self.assertEqual(self.externalUI2.elevator_2_floor_label.text(), "Floor
3")
        self.assertEqual(self.externalUI3.elevator_2_floor_label.text(), "Floor
3")
```

- Test case

	Test Case T2.2.1 – T2.2.5
Coverage Item	Tcover1.1.3 - T1.1.14, Tcover1.2.1 - Tcover1.2.5, Tcover1.3.1 - Tcover1.3.11, Tcover1.4.1 - Tcover1.4.12
Input	1. Press down button outside on floor 2.

	2. When the door of elevator 2 is about to close, press open door button. 3. Press floor -1 button in elevator 2, then press floor 3 button. 4. When elevator 2 is on floor -1, press up button outside on floor 2. 5. Press close door button of elevator 1 when door is open.
State	Elevator 1 stops on floor 1, elevator 2 stops on floor 3.
Expected Output	1. Elevator 2 is called and the door will open when it arrives. 2. The door of elevator 2 opens. 3. Elevator 2 moves to floor -1, then moves to floor 3. 4. Elevator 1 is called. 5. Elevator 1 closes the door.

- Test coverage: 30 / 30 = 100%
- Test result: 1 passed

T3: Functional Test

This section provides information of functional tests we made for the Elevator System. There are executable files as starter for functional tests with initial states set correctly in the corresponding folders for your convenience.

T3.1: Test Open Elevator Door

T3.1.1: Press "Open Door" Button

- Test case

	Test Case T3.1.1.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press "open door" button in both elevators. 2. Wait 3 seconds. 3. Release "open door" button in both elevators (Release the clicks to simulate this). 4. Wait 5 seconds.
Expected Behavior	2. Elevator doors open. 4. Elevator doors close.

- Test result: 1 passed

T3.1.2: Reach Target Floor

- Test case

	Test Case T3.1.2.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Elevator 1 door opens. 2. Press "call up" button outside on floor 1.Wait for elevator 1 coming up.

	3. Wait 3 seconds.
Expected Behavior	1. Elevator 1 is called. 2. Elevator 1 moves to floor 1. 3. Elevator 1 door opens.

- Test result: 1 passed

T3.2: Test Close Elevator Door

T3.2.1: Press "Close Door" Button

- Test case

	Test Case T3.2.1.1	Test Case T3.3.1.2
State	Elevator 1 on floor -1, elevator 2 on floor 3.	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press "open door" button in both elevators. 2. Wait 3 seconds. 3. Release "open door" button in both elevators (Release the clicks to simulate this). 4. Press "close door" button in both elevators.	1. Press "open door" button in both elevators. 2. Release "open door" button in both elevators (Release the clicks to simulate this). 3. Press "close door" button in both elevators.
Expected Behavior	2. Elevator doors open. 4. Elevator doors close.	3. Elevator doors close when the doors are opening.

- Test result: 2 passed

T3.2.2: Reach Target Floor

- Test case

	Test Case T3.2.2.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press "call down" button outside on floor 2. 2. Wait for elevator 2 coming down.
Expected Behavior	2. Elevator 2 door keeps closed when moving.

- Test result: 1 passed

T3.3: Test Select Floor

T3.3.1: Select Single Floor

- Test case

	Test Case T3.3.1.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press floor 2 button in elevator 1, and press floor 1 button in elevator 2.
Expected Behavior	1. Elevator 1 moves to floor 2 and then stops. Elevator 2 moves to floor 1 and then stops.

- Test result: 1 passed

T3.3.2: Select Multiple Floor

- Test case

	Test Case T3.3.2.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press floor 2 button and floor 3 button in elevator 1, and press floor 1 button and floor -1 button in elevator 2.
Expected Behavior	1. Elevator 1 moves to floor 2, 3 and then stops. Elevator 2 moves to floor 1, -1 and then stops.

- Test result: 1 passed

T3.3.3: Select Current Floor

- Test case

	Test Case T3.3.3.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press floor -1 button in elevator 1, and press floor 3 button in elevator 2.
Expected Behavior	1. Elevator 1 and elevator 2 door open.

- Test result: 1 passed

T3.4: Test Call Elevator Outside

- Test case

	Test Case T3.4.1.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press floor 2 button in elevator 1, and press floor 1 button in elevator 2.
Expected Behavior	1. Elevator 1 moves to floor 2 and elevator 2 moves to floor 1 eventually.

- Test result: 1 passed

T3.5: Test Information Display

T3.5.1: Inside Select Floor Button

- Test case

	Test Case T3.5.1.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press floor 2 button in elevator 1, and press floor 1 button in elevator 2. 2. Wait for elevator 1 and elevator 2 arriving.
Expected Behavior	1. Elevator 1 floor 2 button light on, elevator 2 floor 1 button light on. 2. Elevator 1 floor 2 button light off, elevator 2 floor 1 button light off.

- Test result: 1 passed

T3.5.2: Door Open Display

- Test case

	Test Case T3.5.2.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press “open door” button in both elevators. 2. Press “close door” button in both elevators.
Expected Behavior	1. Both elevators have indicator “<|>” light on. The indicator “<|>” for elevator 1 on floor -1 outside light on. The indicator “<|>” for elevator 2 on floor 3 outside light on. 2. Both elevators have indicator “>|<” light on. The indicator for elevator 1 on floor -1 outside becomes “>|<”. The indicator for elevator 2 on floor 3 outside becomes “>|<”.

- Test result: 1 passed

T3.5.3: Floor Number Display

- Test case

	Test Case T3.5.3.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press floor 2 button in elevator 1, and press floor 1 button in elevator 2. 2. Wait for elevator 1 and elevator 2 arriving.
Expected Behavior	0. Elevator 1 shows floor -1, while elevator 2 shows floor 3. All the outside UI show elevator 1 on floor -1 and elevator 2 on floor 3. 2. Elevator 1 shows floor 2, while elevator 2 shows floor 1. All the outside UI show elevator 1 on floor 2 and elevator 2 on floor 1.

- Test result: 1 passed

T3.5.4: Outside Up & Down Button

- Test case

	Test Case T3.5.4.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press “call up” button on floor 1, and press “call down” button on floor 2. 2. Wait until both elevators open door.
Expected Behavior	1. The “call up” button on floor 1 lights on, and the “call down” button on floor 2 lights on. 2. The “call up” button on floor 1 lights off, and the “call down” button on floor 2 lights off.

- Test result: 1 passed

T3.5.5: Up & Down Display

- Test case

	Test Case T3.5.5.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press floor 1 button in elevator 1, and press floor 2 button in elevator 2. 2. Wait until both elevators open door.
Expected Behavior	1. Elevator 1 up indicator lights on, and elevator 2 down indicator lights on. All the outside UI show that elevator 1 up and elevator 2 down. 2. All the indicators light off.

- Test result: 1 passed

T3.6: Test Elevator Management

T3.6.1: Multiple Calls Outside

- Test case

	Test Case T3.6.1.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press both “call up” and “call down” button on floor 2.
Expected Behavior	1. Both the elevators move to floor 2 eventually.

- Test result: 1 passed

T3.6.2: Efficiency

- Test case

	Test Case T3.6.2.1
State	Elevator 1 on floor -1, elevator 2 on floor 3.
Operation	1. Press floor 2 button in elevator 1, and press “call up” button outside on floor 1.
Expected Behavior	1. Elevator 1 moves to floor 1 to deal with the call up and then moves to floor 2.

- Test result: 1 passed

Model Checking

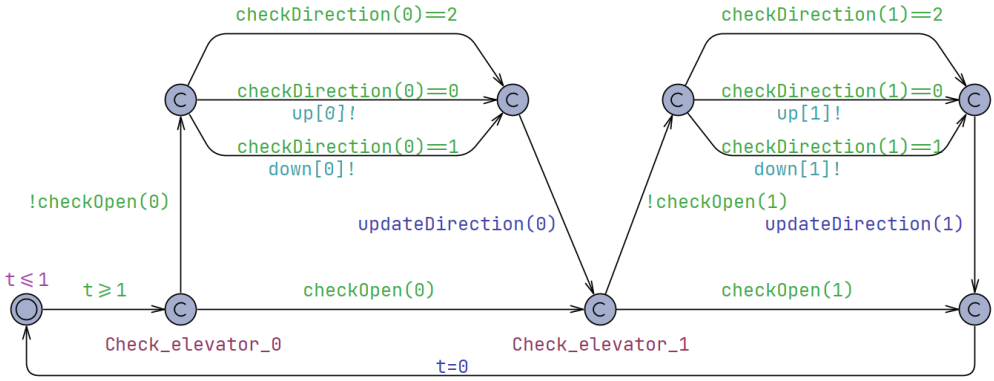
A UPPAAL model of this Elevator system is built for model checking. You could find corresponding files in the validation folder.

Full Elevator Model

The full UPPAAL model consists of 4 parts: 1. The system processor template; 2. The user template; 3. The elevator template; 4. The elevator door template.

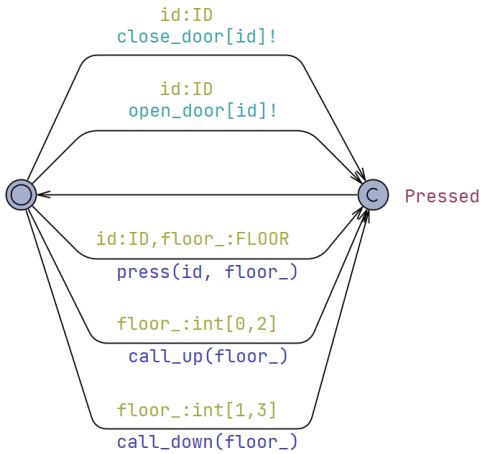
The full model consists of 1 system processor, 1 user, 2 elevators and 2 elevator doors.

The System Processor



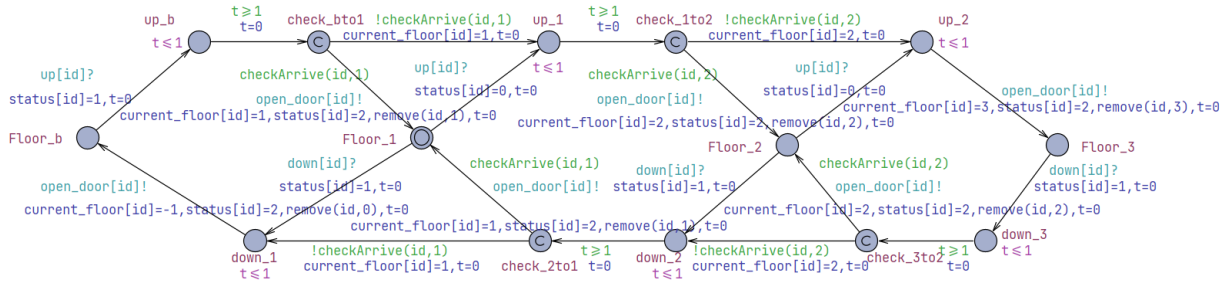
The system processor would check and update the direction of the elevators every tick, which simulates the procedure of assigning tasks.

The User



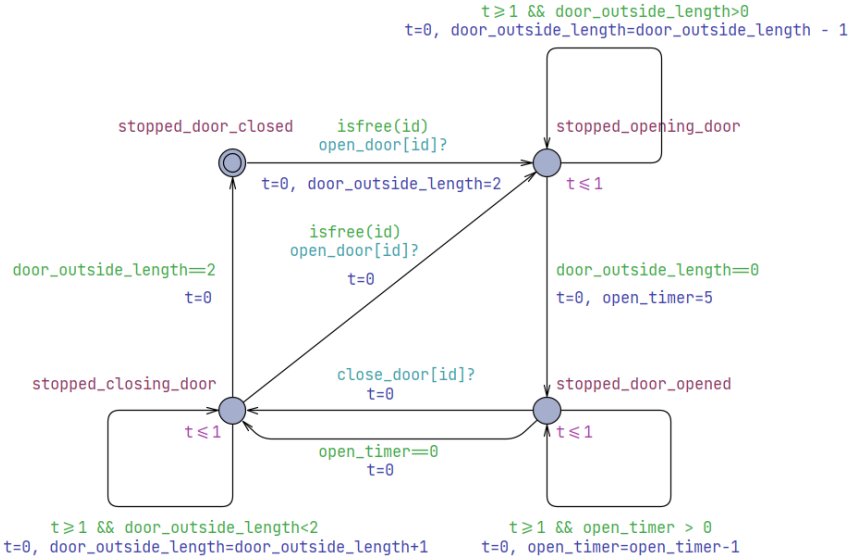
This model simulates the action of users, i.e. opening & closing elevator door, selecting floor inside and calling up & down outside. The user is able to press the buttons at any time.

The Elevator



This model simulates the elevator object. The elevator moves among the 4 floors (we use Floor_b with index 0 to simulate floor -1 here). The elevator could check the tasks to move and open door automatically.

The Elevator Door



This model simulates the action of elevator door. It will open if it receives open request and the situation is valid, that is the elevator is not moving. The elevator door will close if it opens for 5 seconds.

Check Properties

The full model is so large that it cost a lot of time to run some property. We just choose some properties that would not run out of memory. Other checks will be included in sub model checks.

P1.1

Property	E<> ElevatorDoor(1).stopped_door_opened
Description	The elevator could open the door.
Result	Passed

P1.2

Property	E<> Elevator(0).Floor_b
Description	The elevator could move to floor -1.
Result	Passed

P1.3

Property	E<> Elevator(0).Floor_2
Description	The elevator could move to floor 2.
Result	Passed

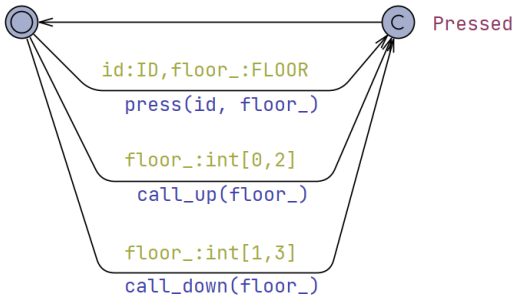
P1.4

Property	E<> Elevator(0).Floor_3
Description	The elevator could move to floor 3.
Result	Passed

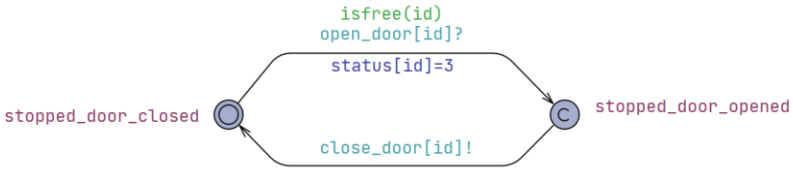
Sub Elevator Model

This sub model focus on the function of elevator movement. This model only consists 2 elevator, 2 elevator doors and 1 user. The elevator model is the same as the full model.

We simplify the user model to focus on the movement of elevator, i.e. we ban the open & close door request in this sub model.



We also simplify the elevator door, so the door will close immediately when it opens.



The checked properties are as follows:

P2.1

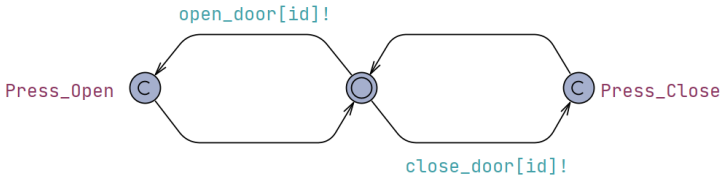
Property	A[] not deadlock
Description	The system will not crash and has no deadlock.
Result	Passed

P2.2

Property	A[] forall(i:ID) ElevatorDoor(i).stopped_door_opened imply Elevator(i).Floor_b or Elevator(i).Floor_1 or Elevator(i).Floor_2 or Elevator(i).Floor_3
Description	Whenever the elevator door is open or opening, it should be stopped at some floor to ensure sercurity.
Result	Passed

Sub Door Model

This sub model focuses on the function of elevator door. This model only consists 2 elevator doors and 2 users, each user controls one elevator door. The elevator door model is the same as the full model. We implement a sub model for user to focus on opening & closing door requests.



The declaration of the environment does not change, but this sub model is more efficient and costs less memory for model checking.

The checked properties are as follows:

P3.1

Property	A[] not deadlock
Description	The system will not crash and has no deadlock.
Result	Passed

P3.2

Property	E<> ElevatorDoor(1).stopped_door_opened
Description	The elevator could open the door.
Result	Passed

P3.3

Property	A[] forall(i:ID) User(i).Press_Open imply not ElevatorDoor(i).stopped_door_closed and not ElevatorDoor(i).stopped_closing_door
Description	If the user presses the “Open” button, the door should try to open, i.e. it should not be closing or closed.
Result	Passed

P3.4

Property	A[] forall(i:ID) User(i).Press_Close imply not ElevatorDoor(i).stopped_door_opened
Description	If the user presses the “Close” button, the door will try to close, i.e. it should not be opened.
Result	Passed

P3.5

Property	A[] ElevatorDoor(0).door_outside_length>=0 and ElevatorDoor(0).door_outside_length<=2
Description	The door never opens or closes so much that exceed the door size.
Result	Passed

P3.6

Property	A[] ElevatorDoor(0).open_timer>=0 and ElevatorDoor(0).open_timer<=5
Description	The elevator would never keep opening for more than 5 seconds without user pressing the “Open” button.
Result	Passed