Group 7

Author: Jintong Luo

Software Validation

Elevator

Table of Contents

[System Architecture 5](#_Toc170082877)

[T1: Unit Test 5](#_Toc170082878)

[T1.1: ElevatorProcessor Unit Test 5](#_Toc170082879)

[T1.1.1: Test getStaticDoor() 5](#_Toc170082880)

[T1.1.2: Test getStaticTimer() 6](#_Toc170082881)

[T1.1.3: Test update() 6](#_Toc170082882)

[T1.1.4: Test update\_floor() 8](#_Toc170082883)

[T1.1.5: Test update\_door() 9](#_Toc170082884)

[T1.1.6: Test update\_direction() 11](#_Toc170082885)

[T1.1.7: Test update\_state() 14](#_Toc170082886)

[T1.1.8: Test process\_InternalUI\_requests(InterUI\_MSG) 15](#_Toc170082887)

[T1.1.9: Test open\_door() 16](#_Toc170082888)

[T1.1.10: Test close\_door() 17](#_Toc170082889)

[T1.1.11: Test checkOpen() 19](#_Toc170082890)

[T1.1.12: Test checkArrive() 19](#_Toc170082891)

[T1.1.13: Test compute\_callup\_time(floor) 21](#_Toc170082892)

[T1.1.14: Test compute\_calldown\_time(floor) 22](#_Toc170082893)

[T1.2: SystemProcessor Unit Test 23](#_Toc170082894)

[T1.2.1: Test update() 23](#_Toc170082895)

[T1.2.2: Test process\_ExternalUI\_requests(ExterUI\_MSG) 24](#_Toc170082896)

[T1.2.3: Test receive\_eleProcessor\_MSG(message) 27](#_Toc170082897)

[T1.2.4: Test getUpTime(floor) 28](#_Toc170082898)

[T1.2.5: Test getDownTime(floor) 29](#_Toc170082899)

[T1.3: ExternalUI Unit Test 29](#_Toc170082900)

[T1.3.1: Test update() 29](#_Toc170082901)

[T1.3.2: Test update\_time() 30](#_Toc170082902)

[T1.3.3: Test update\_state() 30](#_Toc170082903)

[T1.3.4: Test update\_floor() 31](#_Toc170082904)

[T1.3.5: Test update\_direction() 31](#_Toc170082905)

[T1.3.6: Test update\_button() 33](#_Toc170082906)

[T1.3.7: Test push\_up\_button() 34](#_Toc170082907)

[T1.3.8: Test push\_down\_button() 34](#_Toc170082908)

[T1.3.9: Test checkOpen(processor) 35](#_Toc170082909)

[T1.3.10: Test checkTargetUp(floor) 35](#_Toc170082910)

[T1.3.11: Test checkTargetDown(floor) 36](#_Toc170082911)

[T1.4: InternalUI Unit Test 37](#_Toc170082912)

[T1.4.1: Test update() 37](#_Toc170082913)

[T1.4.2: Test update\_time() 38](#_Toc170082914)

[T1.4.3: Test update\_state() 38](#_Toc170082915)

[T1.4.4: Test update\_floor() 39](#_Toc170082916)

[T1.4.5: Test update\_floor\_button() 39](#_Toc170082917)

[T1.4.6: Test update\_direction() 40](#_Toc170082918)

[T1.4.7: Test push\_open\_door\_button() 41](#_Toc170082919)

[T1.4.8: Test push\_close\_door\_button() 41](#_Toc170082920)

[T1.4.9: Test push\_floor\_button\_b() 42](#_Toc170082921)

[T1.4.10: Test push\_floor\_button\_1() 42](#_Toc170082922)

[T1.4.11: Test push\_floor\_button\_2() 43](#_Toc170082923)

[T1.4.12: Test push\_floor\_button\_3() 43](#_Toc170082924)

[T2: Integration Test 44](#_Toc170082925)

[T2.1: ElevatorProcessor + 2 InternalUI Integration 44](#_Toc170082926)

[T2.1.1: Test Open Door Button 44](#_Toc170082927)

[T2.1.2: Test Close Door Button 45](#_Toc170082928)

[T2.1.3: Test Internal Floor Button & Elevator Move 45](#_Toc170082929)

[T2.2: ElevatorProcessor + 2 InternalUI + 4 ExternalUI Integration 46](#_Toc170082930)

[T3: Functional Test 50](#_Toc170082931)

[T3.1: Test Open Elevator Door 50](#_Toc170082932)

[T3.1.1: Press “Open Door” Button 50](#_Toc170082933)

[T3.1.2: Reach Target Floor 50](#_Toc170082934)

[T3.2: Test Close Elevator Door 51](#_Toc170082935)

[T3.2.1: Press “Close Door” Button 51](#_Toc170082936)

[T3.2.2: Reach Target Floor 51](#_Toc170082937)

[T3.3: Test Select Floor 52](#_Toc170082938)

[T3.3.1: Select Single Floor 52](#_Toc170082939)

[T3.3.2: Select Multiple Floor 52](#_Toc170082940)

[T3.3.3: Select Current Floor 52](#_Toc170082941)

[T3.4: Test Call Elevator Outside 53](#_Toc170082942)

[T3.5: Test Information Display 53](#_Toc170082943)

[T3.5.1: Inside Select Floor Button 53](#_Toc170082944)

[T3.5.2: Door Open Display 53](#_Toc170082945)

[T3.5.3: Floor Number Display 54](#_Toc170082946)

[T3.5.4: Outside Up & Down Button 54](#_Toc170082947)

[T3.5.5: Up & Down Display 54](#_Toc170082948)

[T3.6: Test Elevator Management 55](#_Toc170082949)

[T3.6.1: Multiple Calls Outside 55](#_Toc170082950)

[T3.6.2: Efficiency 55](#_Toc170082951)

[Model Checking 56](#_Toc170082952)

[Full Elevator Model 56](#_Toc170082953)

[The System Processor 56](#_Toc170082954)

[The User 56](#_Toc170082955)

[The Elevator 57](#_Toc170082956)

[The Elevator Door 57](#_Toc170082957)

[Check Properties 57](#_Toc170082958)

[P1.1 57](#_Toc170082959)

[P1.2 58](#_Toc170082960)

[P1.3 58](#_Toc170082961)

[P1.4 58](#_Toc170082962)

[Sub Elevator Model 58](#_Toc170082963)

[P2.1 58](#_Toc170082964)

[P2.2 59](#_Toc170082965)

[Sub Door Model 59](#_Toc170082966)

[P3.1 59](#_Toc170082967)

[P3.2 59](#_Toc170082968)

[P3.3 59](#_Toc170082969)

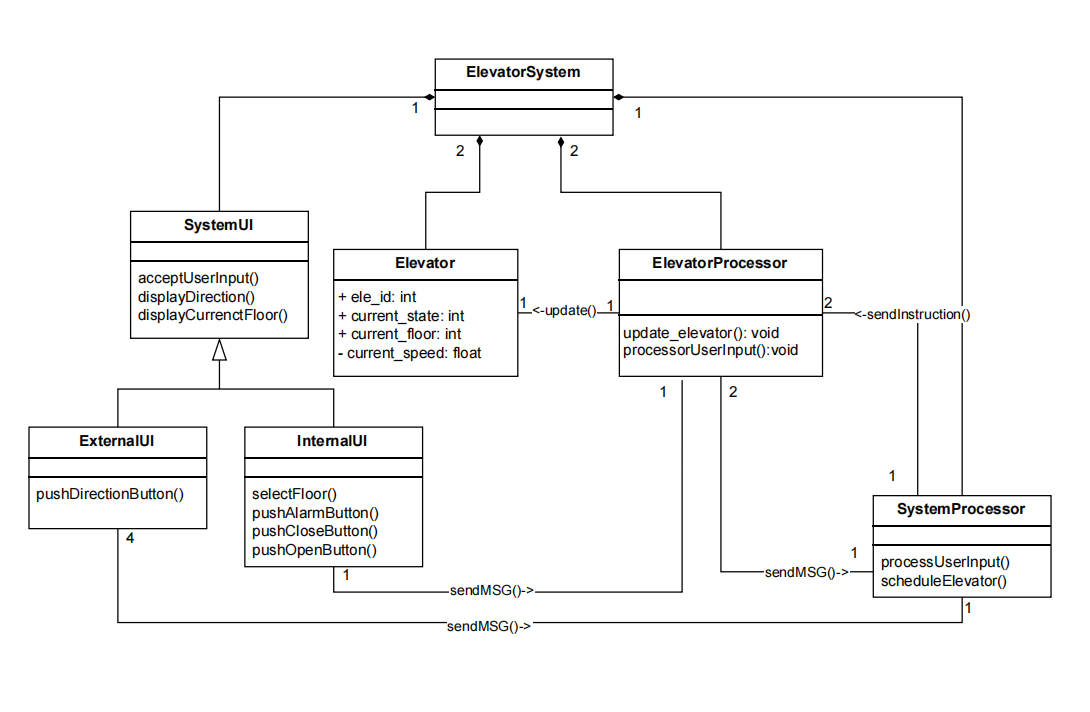
[P3.4 60](#_Toc170082970)

[P3.5 60](#_Toc170082971)

[P3.6 60](#_Toc170082972)

## 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\_floor}\_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]  elevator = processor.elevator  elevator.ele\_id = 1  elevator.current\_floor = 2  elevator.direction = DirectionState.idle  processor.checkArrive.return\_value = True | processor = SystemProcessor().elevator\_processors[0]  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  elevator.direction = DirectionState.down  elevator.current\_state = ElevatorState.down | processor = SystemProcessor().elevator\_processors[0]  elevator = processor.elevator  elevator.current\_floor = 2  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  elevator.current\_state = ElevatorState.stopped\_door\_opened  processor.open\_timer = 0 | processor = SystemProcessor().elevator\_processors[0]  elevator = processor.elevator  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.target\_floor = [False, True, False, False]  processor.target\_floor\_up = [False, False, False, False]  processor.target\_floor\_down = [False, False, True, False] | processor = SystemProcessor().elevator\_processors[0]  elevator = processor.elevator  elevator.current\_floor = 3  elevator.direction = DirectionState.idle  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]  elevator = processor.elevator  elevator.current\_state = ElevatorState.stopped\_door\_closed  elevator.direction = DirectionState.up | processor = SystemProcessor().elevator\_processors[0]  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  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)) | 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[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]  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)) | system\_processor = SystemProcessor()  processor1 = system\_processor.elevator\_processors[0]  processor2 = system\_processor.elevator\_processors[1]  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.down\_button.styleSheet() == circle\_button\_style | ui.up\_button.styleSheet() == circle\_button\_style\_on, 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]  processor.elevator\_processors[1].target\_floor\_up = [False, False, 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] |
| 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.processor.elevator.direction = DirectionState.idle | ui = InternalUI(SystemProcessor().elevator\_processors[0])  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.processor.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.processor.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.processor.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 | 1. 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

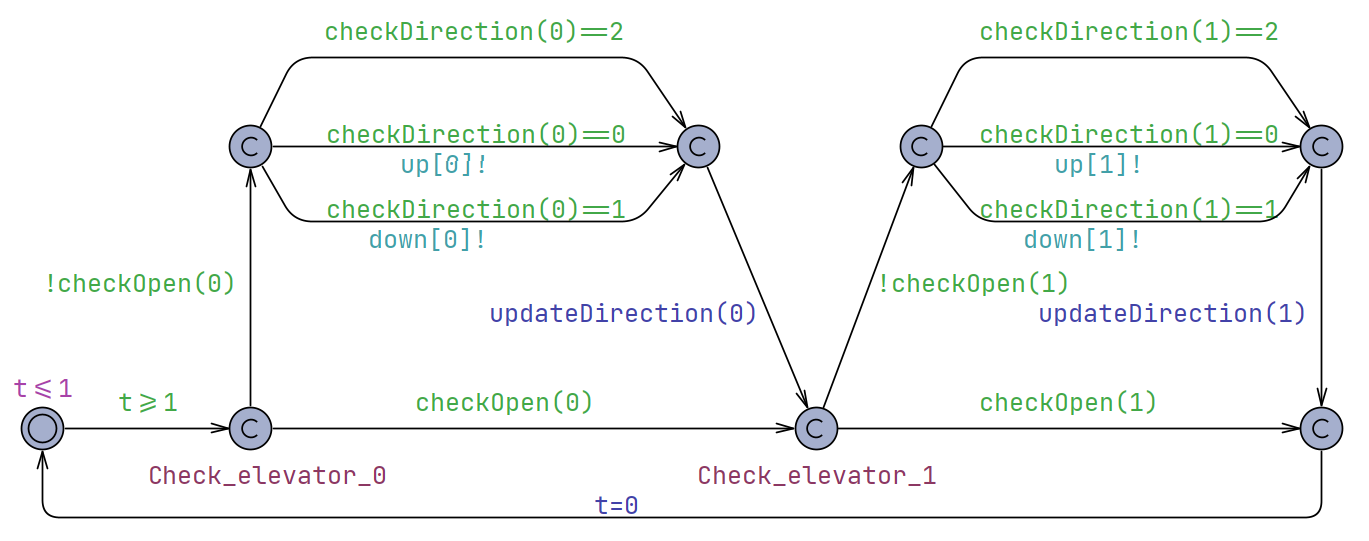
A UPPAAL model of this Elevator system is built for model checking. You could find corresponding files in the validation folder.

### 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.

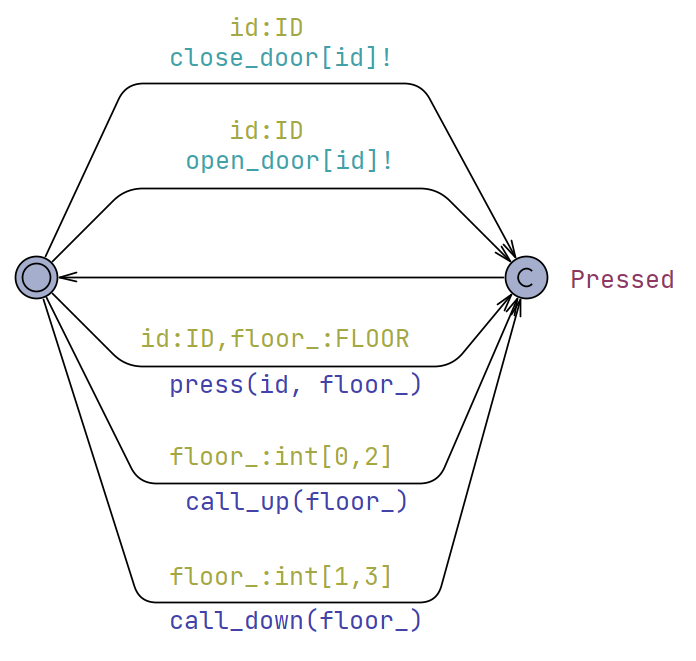
The full model consists of 1 system processor, 1 user, 2 elevators and 2 elevator doors.

#### The System Processor



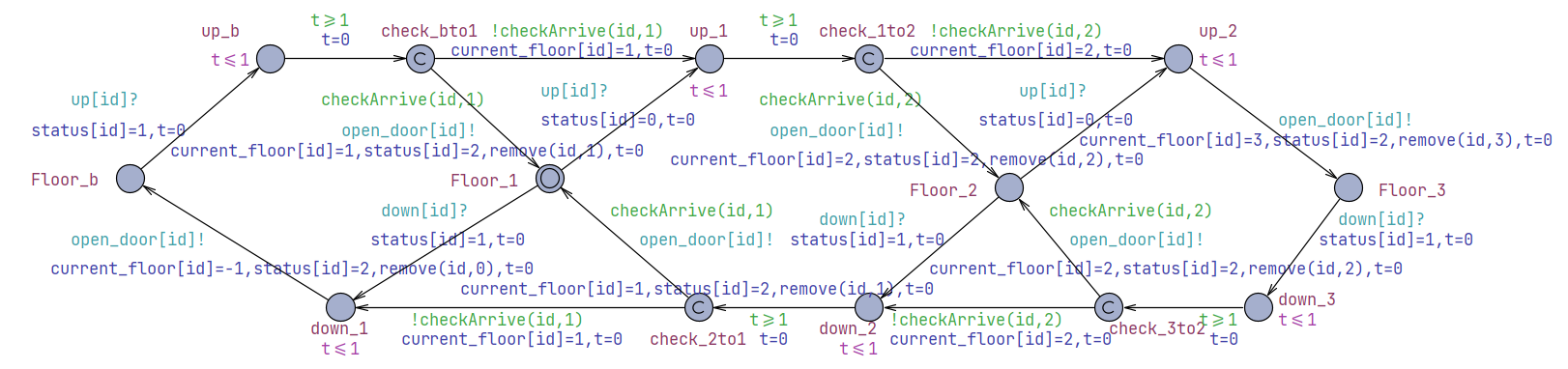
The system processor would check and update the direction of the elevators every tick, which simulates the procedure of assigning tasks.

#### The User



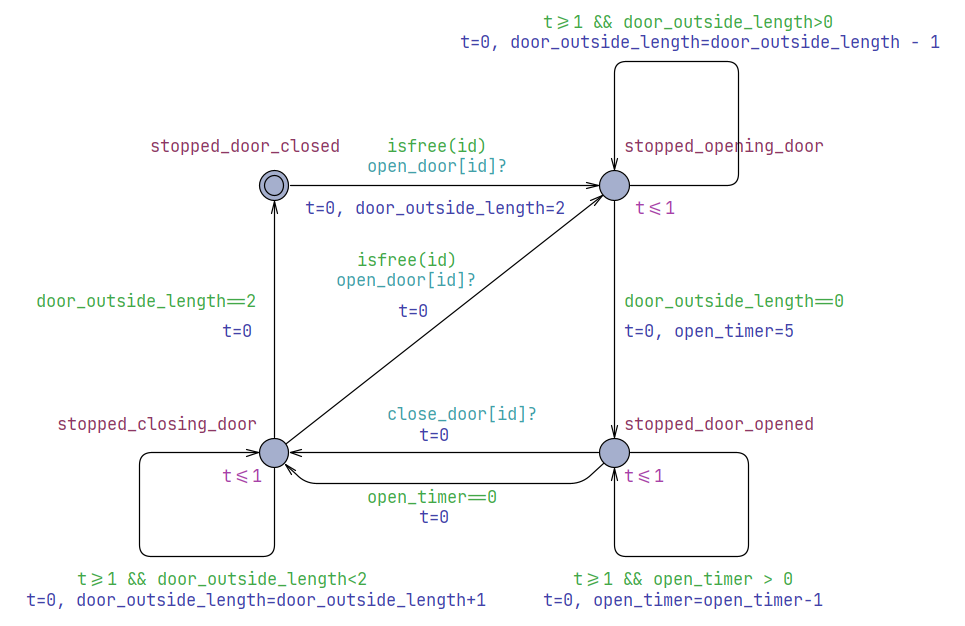
This model simulates the action of users, i.e. opening & closing elevator door, selecting floor inside and calling up & down outside. The user is able to press the buttons at any time.

#### 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 |

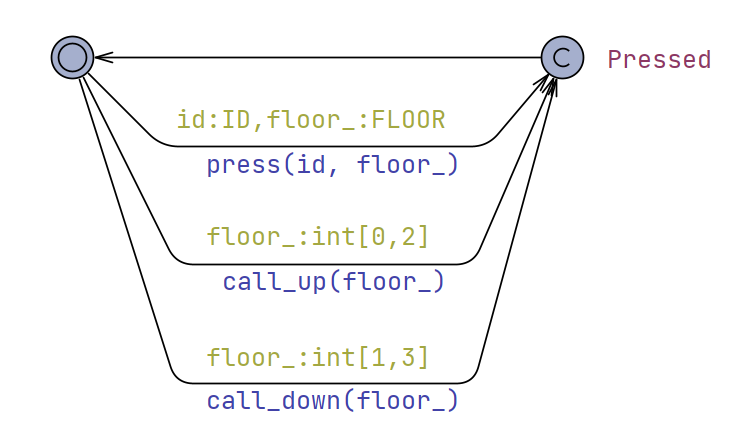
#### P1.4

|  |  |
| --- | --- |
| Property | E<> Elevator(0).Floor\_3 |
| Description | The elevator could move to floor 3. |
| Result | Passed |

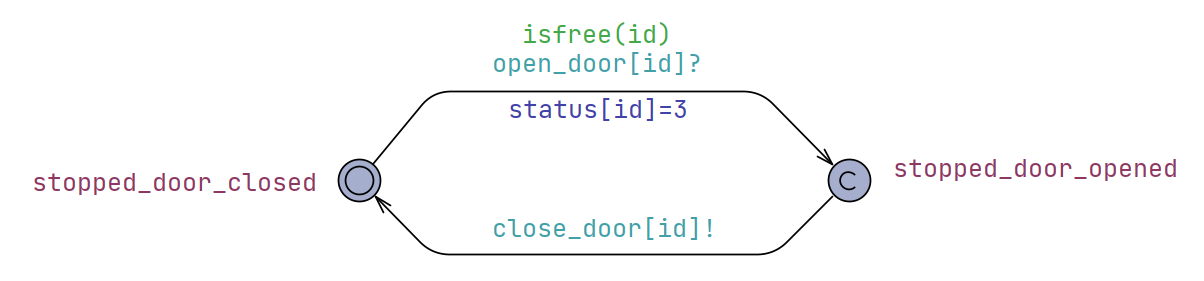
### Sub Elevator Model

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

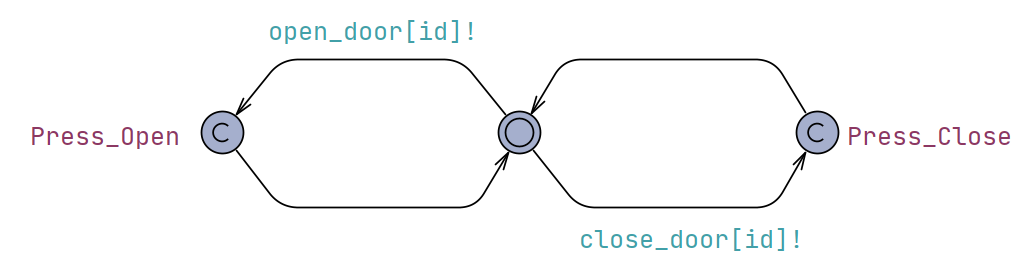
|  |  |
| --- | --- |
| Property | A[] not deadlock |
| Description | The system will not crash and has no deadlock. |
| Result | Passed |

#### 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 |