CS132: Software Engineering

Specification

Elevator

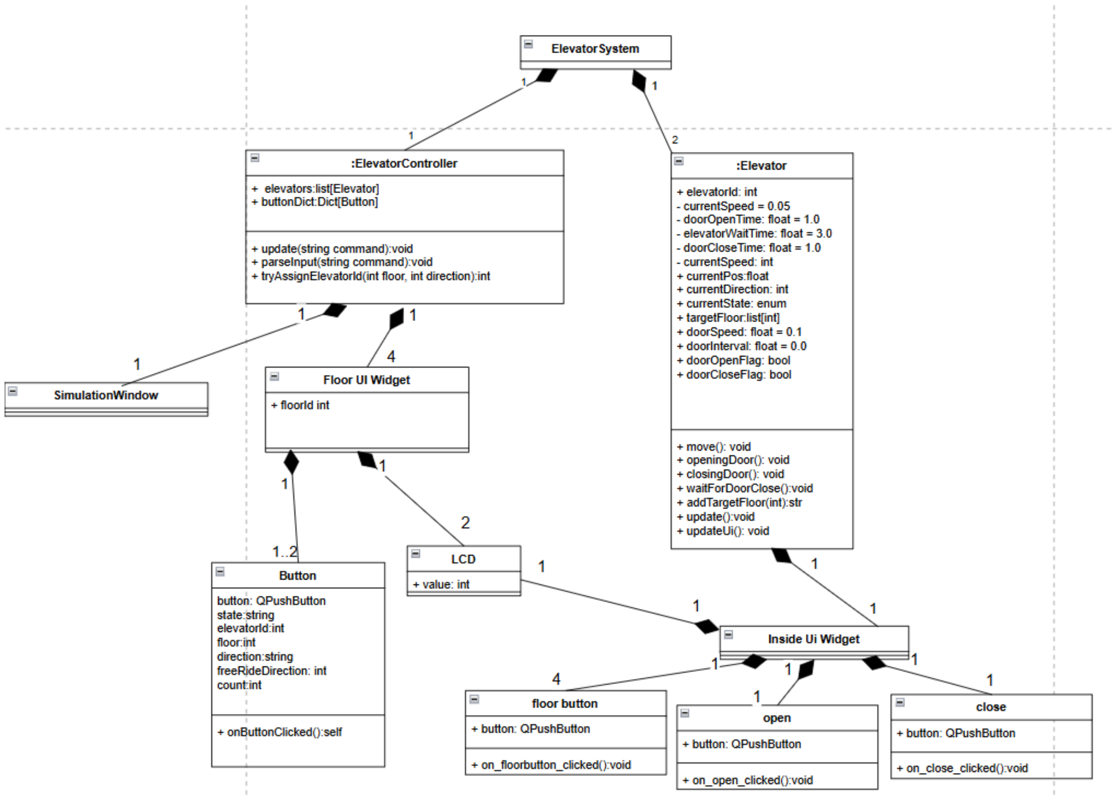
Group 15

Author: 祝宇航 王芸飞 贾舜康

Catalogue

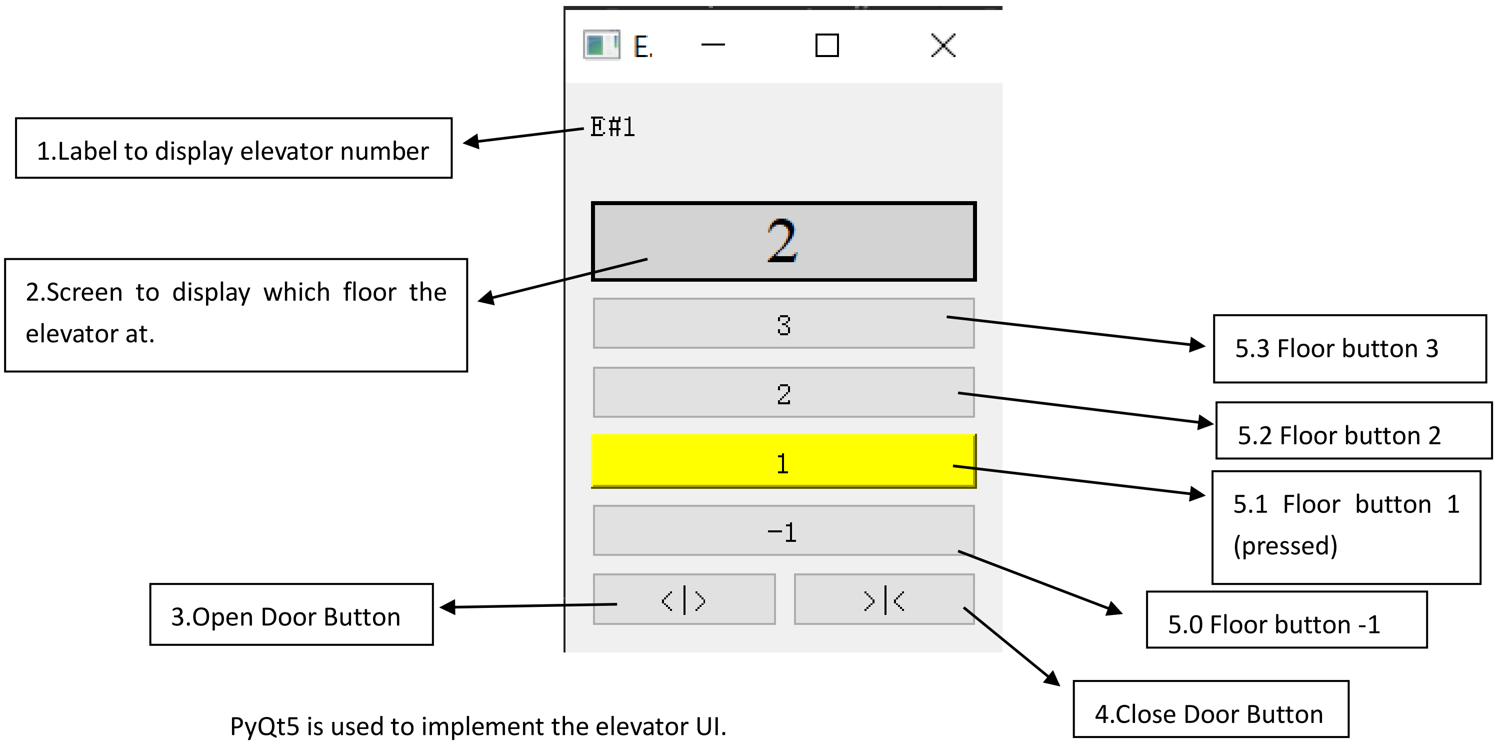
System Architecture

The system architecture is shown below:



1. Elevator Implementation

1.1 Elevator UI


E.
E#1
1.Label to display elevator number
2
2.Screen to display which floor the elevator at.
3
2
1
-1
<|>
>|<
5.3 Floor button 3
5.2 Floor button 2
5.1 Floor button 1 (pressed)
5.0 Floor button -1
3.Open Door Button
4.Close Door Button

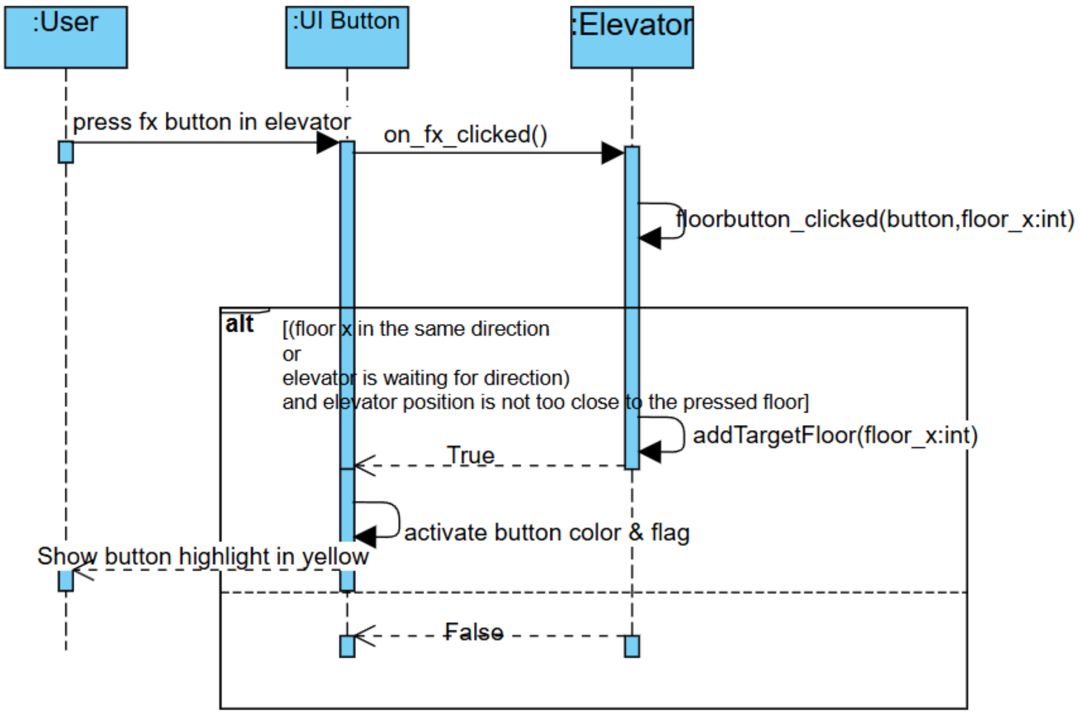
PyQt5 is used to implement the elevator UI.
Each elevator UI consists of:
1. One QLabel to display elevator number
2. A LCD Screen with one digit to display which floor the elevator is closest to. It is updated per iteration using `set_lcd_value(self,value)` and `getCurrentFloor(self)`.E.g. If the elevator is at position 1.4 floor, LCD displays 1. If the elevator is at position 2.7 floor, LCD screen displays 3.
3. Open Door Button: QPushButton with clicked event "`on_open_clicked(self)`" and it communicate with Elevator using a boolean flag: "`doorOpenFlag`"
4. Close Door Button: QPushButton with clicked event "`on_close_clicked(self)`" and it communicate with Elevator using a boolean flag: "`doorCloseFlag`"
5. Floor Door Button: QPushButton with clicked event "`on_f1_clicked(self)`", "`on_f2_clicked(self)`", "`on_f3_clicked(self)`", it communicate with Elevator using corresponding boolean flag: "`fx_activeFlag`".

Elevator UI is initialized in "`setupUI(self)`"
All functions mentioned above can be found at "project/src/elevator.py" in class Elevator.

1.2 Button Event

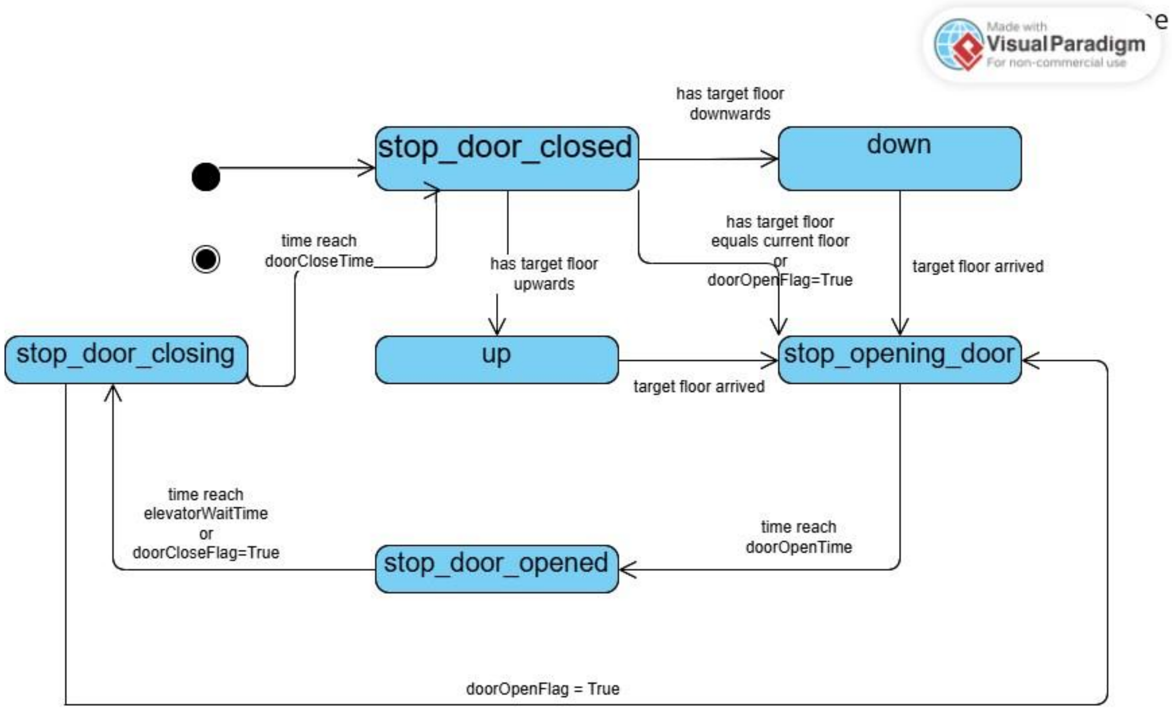
When user presses open or close button, a boolean flag will be set, and later in update, the

corresponding function will handle the flag.
When user presses a floor button, however, the case will be more complicated. Every time a floor button is clicked, `floorbutton_clicked(self,button,floor)->bool` will be called to check if the floor that the button represents is align with the direction of the elevator. If it is checked to be true, then the floor will be added to the `targetFloor` list by a function `addTargetFloor(floor)->string`.
A following sequence diagram show a series of events when a floor button fx is clicked.


:User
:UI Button
:Elevator
press fx button in elevator
on_fx_clicked()
floorbutton_clicked(button,floor_x:int)
alt
[(floor x in the same direction
or
elevator is waiting for direction)
and elevator position is not too close to the pressed floor]
addTargetFloor(floor_x:int)
True
activate button color & flag
Show button highlight in yellow
False

1.3 Elevator State Machine Overview

The following state machine diagram shows the overview of the elevator state machine. There are overall 6 states: stop_door_closed; up; down; stop_closing_door; stop_opening_door; stop_door_opened. Transition conditions are listed on the edges. The state enum definition can be found in “project/src/elevatorState.py” in class State.



1.4 State: stop_door_closed

In this state, elevator itself is idle. The only thing it needs to do is to check if it is required to transfer to other state in every update iteration. The functions related to this state are “chekTargetFloor(self)->bool” and “checkOpenDoor(self)” in “project/src/elevator.py”.

1.5 State: up and down

In these two states, the elevator moves according to the direction specified by the currentDirection attribute. When the elevator is in the State.up state, it moves upward by incrementing its position (currentPos) by the current speed (__currentSpeed). Conversely, when in the State.down state, the elevator moves downward by decrementing its position by the current speed. The `move(self)` function handles this behavior and also checks if the elevator has reached the target floor. More detailed information about this state is given below.

Function: move(self) -> None

- If currentState is State.up, the elevator's position (currentPos) is increased by __currentSpeed.
- If currentState is State.down, the elevator's position is decreased by __currentSpeed.
- The function then checks if the elevator has reached the target floor (within a small margin of error). If it has, the elevator transitions to the State.stopped_opening_door state, sets the current position to the target floor, sends an arrival message, clears the floor UI, and if there are no more target floors, resets the direction to Direction.wait.

1.6 State: stopped_opening_door

In this state, the elevator's doors are opening. The function `openingDoor(self)` manages the transition of door states from closed to fully opened.
Function: `openingDoor(self) -> None`
- The door interval (`doorInterval`) is incremented by the door speed (`doorSpeed`).
- Once the door interval reaches the predefined door open time (`Elevator.doorOpenTime`), the door is considered fully opened. A door opened message is sent, and the state is changed to `State.stopped_door_opened`.

1.7 State: stopped_door_opened

In this state, the elevator waits for a command to close the door or a timeout. The function `waitForClosingDoor(self)` handles this state.

Function: `waitForClosingDoor(self) -> None`
-If the door open flag (`doorOpenFlag`) is set, the function keeps the door open.
-If the door close flag (`doorCloseFlag`) is set, the state transitions to `State.stopped_closing_door`.
- If neither flag is set, the door interval is incremented by the door speed. If the interval reaches the predefined wait time (`Elevator.elevatorWaitTime`), the state transitions to `State.stopped_closing_door`.

1.8 State: stopped_closing_door

In this state, the elevator's doors are closing. The function `closingDoor(self)` manages the transition from fully opened to fully closed doors.

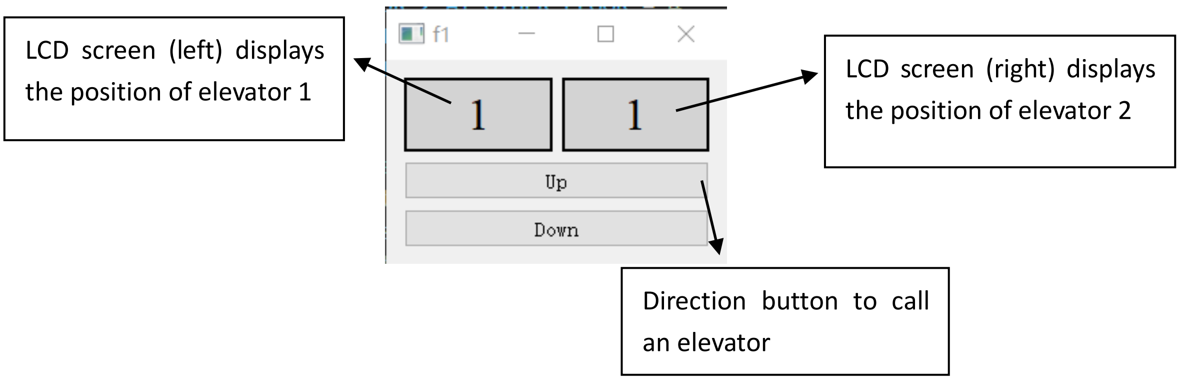
Function: `closingDoor(self) -> None`
- The door interval is incremented by the door speed.
- If the door interval reaches the predefined door close time (`Elevator.doorCloseTime`), the door is considered fully closed. A door closed message is sent, and the state is changed to `State.stopped_door_closed`.

1.9 Update Method

The `update(self)` method orchestrates the state machine by calling the appropriate function based on the current state.

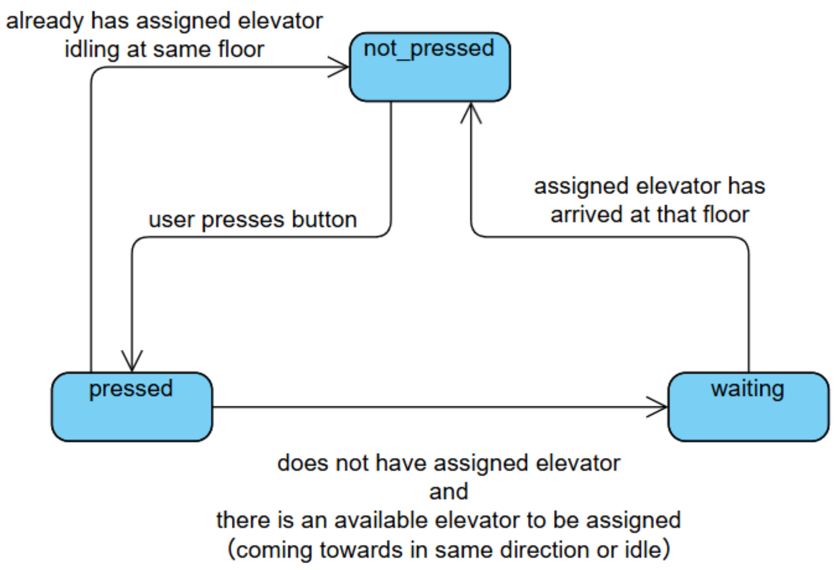
2. Elevator Controller Implementation

2.1 UI Overview


LCD screen (left) displays the position of elevator 1
f1
1
1
Up
Down
LCD screen (right) displays the position of elevator 2
Direction button to call an elevator

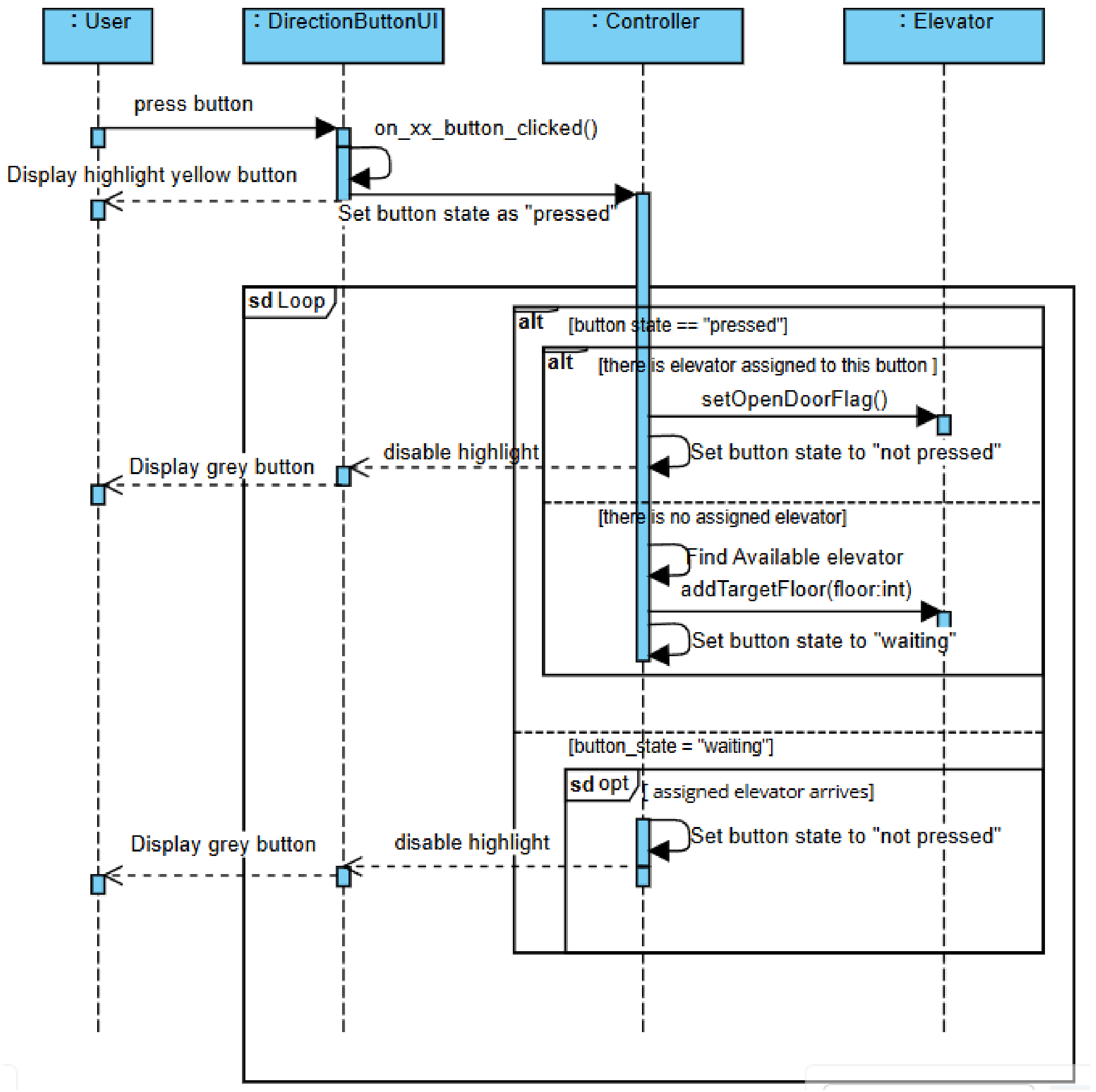
2.2 Direction Button State Machine

User can press the direction button to call the elevator according to the direction he wants to go. The button will be lit up after pressing. When the elevator arrives, the light will be off.
When elevator has already arrived, user can press the direction button again to hold the door.
Every direction button has 3 states, and their state machine is shown below.


already has assigned elevator idling at same floor
not_pressed
assigned elevator has arrived at that floor
user presses button
pressed
waiting
does not have assigned elevator and there is an available elevator to be assigned (coming towards in same direction or idle)

2.3 Direction Button Sequence Diagram

When user clicked a button at any floor, a series of events happen in both controller and elevators. The following sequence diagram gives a glimpse into how function calls are performed between different classes. The more detailed explanation for schedule algorithms is in the next section.



: User
: DirectionButtonUI
: Controller
: Elevator
press button
on_xx_button_clicked()
Display highlight yellow button
Set button state as "pressed"
sd Loop
alt [button_state == "pressed"]
alt [there is elevator assigned to this button]
setOpenDoorFlag()
disable highlight
Set button state to "not pressed"
Display grey button
[there is no assigned elevator]
Find Available elevator
addTargetFloor(floor:int)
Set button state to "waiting"
[button_state = "waiting"]
sd opt [assigned elevator arrives]
Set button state to "not pressed"
Display grey button
disable highlight

2.4 Schedule Algorithm

In this section, the details of schedule algorithm will be introduced. Before we actually schedule an elevator to meet the need of the passenger pressing the outside panel, we first check if there is already a elevator assigned to that button, if yes, we set open door flag for that elevator directly.

If there is no assigned elevator yet, we start to schedule. The process is done in `tryAssignElevatorId(self,floor,direction:Direction)` in elevatorController.py.

1. Scheduler finds if there is any elevator stopped with no target floor at the same floor as the

pressed button. It is done by `getElevatorIdleAtSameFloor(self,floor,dist)` in elevatorController.py.

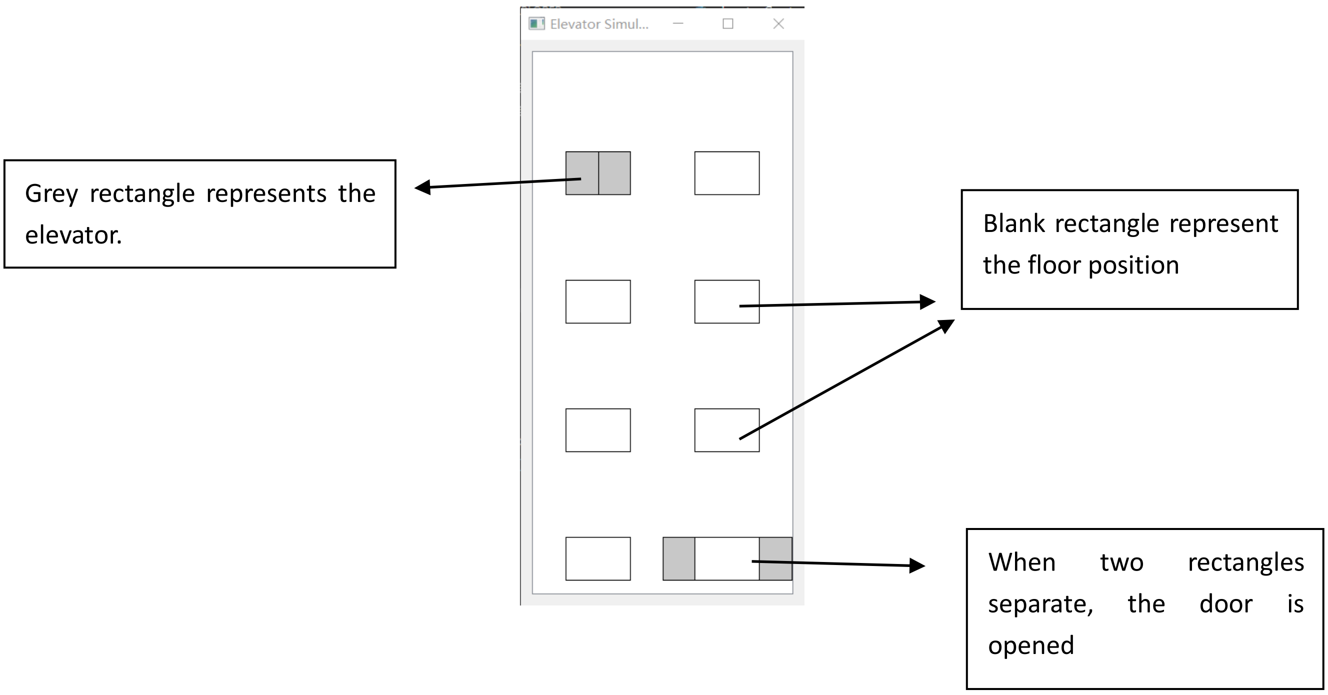
2. Scheduler checks if there is an elevator moving towards this floor with the same direction the passenger want to go. For example, if a passenger presses “up” at F1, then the scheduler check if there is an elevator is moving upwards and its position is lower than F1 (to be more specific, scheduler also ensures only when elevator position is more than 0.5 floor away from the current floor, will it be allowed to take the free ride). It is done by `getNearestElevatorWithDirect(floor,direction,dist)` in in elevatorController.py.
3. Finally, scheduler checks if there is an elevator stop with no task at other floors. It is done by `getNearestStopElevator(floor,dist)` in elevatorController.py.

The schedule function `tryAssignElevatorId` is called once per update for every button that has “pressed” state, after the button has been assigned to one elevator, its state will be set to “waiting”.

Moreover, to avoid the situation of one button can control the two elevator at same time, we also consider the situation case by case in `tryAssignElevatorId` and remove the bind between the button and the elevator once the elevator leaves that floor.

3.Simulation Window

A simulation window is provided to monitor the movement of elevator. It is implemented by mapping the position and door open proportion of elevator to the position in the UI window in elevatorController.py



Grey rectangle represents the elevator.

Blank rectangle represent the floor position

When two rectangles separate, the door is opened