# Elevator Simulator Project Documentation

## Overview

This project is an elevator simulator that mimics the operation of multiple elevators in a multi-story building. The simulator graphically displays elevator movement and allows users to call elevators from different floors.

## Project Structure

### 1. Main File (`main.py`)

- Initializes the building object

- Runs the main game loop

### 2. Building Class (`building.py`)

- Manages the overall logic of the building

- Initializes floors and elevators

- Handles user requests for elevators

- Updates the building state each frame

### 3. Elevator Class (`elevator.py`)

- Defines elevator behavior

- Calculates arrival times

- Updates elevator position

### 4. Floor Class (`floor.py`)

- Represents a single floor in the building

- Includes a button and timer display

### 5. Button Class (`button.py`)

- Defines the appearance and functionality of buttons on each floor

### 6. Timer Classes (`timer.py`, `show\_timer.py`)

- Handle time calculations and time display on screen

### 7. Settings (`settings.py`)

- Contains constants and predefined values for the project

## Key Algorithms

### Elevator Selection

- Calculates the estimated arrival time of each elevator to the requested floor

- Chooses the elevator with the shortest arrival time

### Elevator Position Update

- Calculates the maximum distance an elevator can move in each frame

- Updates the elevator position based on speed and elapsed time

## Simulation Process

1. Initialize the building with a defined number of floors and elevators

2. Main game loop:

- Update building state

- Handle user events (button presses)

- Update elevator positions

- Draw the building and elevators on screen

## Event Handling

- Clicking a floor button triggers the `elevator\_controller`

- `elevator\_controller` selects the most suitable elevator and adds the floor to its order list

## Possible Extensions

1. Adding more advanced elevator routing algorithms

2. Including statistics (such as average waiting times)

3. Creating an advanced user interface for controlling simulation parameters

## Developer Notes

- Ensure all required files are in the project directory

- Check `settings.py` before running to adjust parameters as needed

- Use `pygame.time.Clock()` to control the frame rate of the simulation

## Core Functions

### `Building.elevator\_controller(floor)`

```python

def elevator\_controller(self, floor):

min\_time = float('inf')

for index, elevator in enumerate(self.elevators):

time\_elev = elevator.calc\_elev\_time(floor)

if time\_elev < min\_time:

min\_time = time\_elev

min\_idx = index

if not floor.is\_pressed:

floor.is\_pressed = True

floor.timer.reaching\_time = min\_time

self.elevators[min\_idx].orders.append(floor)

def update\_position(self, elapsed\_time):

target\_y = self.current\_floor.y

distance = target\_y - self.y

max\_move = FLOOR\_HEIGHT \* elapsed\_time / 0.5

if abs(distance) <= max\_move:

self.y = target\_y

else:

direction = 1 if distance > 0 else -1

self.y += direction \* max\_move

This documentation provides an in-depth overview of the project structure, key algorithms, and simulation process. It's designed to help developers understand the logic behind the project and facilitate easier maintenance and expansion.