Design Explanation

For the class diagrams, I created classes that align with my use case models and the specifications in the simulation requirements. My UML class diagram consists of all the components of the elevator system along with the simulation class. The SimulationControls class follows the singleton pattern, ensuring that only one instance controls the entire simulation. The LogConsole class follows the observer pattern, as it observes and logs system events. The SimulationControls class also implements the facade pattern, simplifying interactions with the complex elevator operation subsystem. Additionally, the MainWindow class follows the controller pattern, coordinating interactions between the building, elevators, and passengers.

For the sequence diagrams, I created four diagrams corresponding to various scenarios in the elevator system. Each diagram adheres to the system constraints, such as the number of floors and passengers.

- Sequence Diagram 1.1 (Normal Event): Depicts a single passenger requesting an elevator at floor 5 and traveling to floor 100.
- Sequence Diagram 1.2 (Normal Event): Represents multiple passengers on different floors, each traveling to distinct destinations.
- Sequence Diagram 1.3 (Safety Event): Illustrates a passenger pressing the help button during normal operations, triggering an emergency response.
- Sequence Diagram 1.4 (Safety Event): Depicts a power outage event where the system moves all the elevators to a safe floor.

For the state machine diagrams, I created two diagrams representing elevator and simulation controller behavior:

- Elevator State Machine Diagram: Represents the process of requesting an elevator, riding to a different floor, handling emergency cases, and allowing re-entry for additional travel.
- Simulation Controller State Machine Diagram: Represents initialization and management of the simulation, including running, pausing, stopping, and logging events before termination.

Regarding the C++ header files, most member function signatures and variables are self-documenting. For unintuitive methods, I provided comments to explain their role and significance within the system design.