# Elevator Simulator 电梯模拟器

## 操作系统课程作业

同济大学软件学院

指导老师：张慧娟

1652795 王陆洋

# 目录

[1. 概述 4](#_Toc513539805)

[1.1 项目简介 4](#_Toc513539806)

[1.2 项目目的 4](#_Toc513539807)

[1.3 项目主要运用到的技术 4](#_Toc513539808)

[2. 项目内容 5](#_Toc513539809)

[2.1 类简介 5](#_Toc513539810)

[2.1.1 cn.leonwong.ElevatorSimulator.Model 包 5](#_Toc513539811)

[2.1.2 cn.leonwong.ElevatorSimultor中的其他类 13](#_Toc513539812)

[2.2 项目运行机制简介 22](#_Toc513539813)

[3. 项目操作示例 22](#_Toc513539814)

[3.1 新建建筑 22](#_Toc513539815)

[3.2 增加一个乘客 25](#_Toc513539816)

[3.3 随机批量增加乘客 29](#_Toc513539817)

[3.4 修改分配策略。 31](#_Toc513539818)

[4. 程序测试 32](#_Toc513539819)

[4.1 边界测试 32](#_Toc513539820)

[4.1.1 建筑参数在边界上。 32](#_Toc513539821)

[4.2 出错测试 33](#_Toc513539822)

[4.2.1 建筑物参数错误 33](#_Toc513539823)

[4.2.2 乘客参数错误 34](#_Toc513539824)

[5. 项目心得 35](#_Toc513539825)

[5.1 线程的运用 36](#_Toc513539826)

[5.2 GUI的编写 36](#_Toc513539827)

[5.3 MVC模式的运用 36](#_Toc513539828)

[5.4 项目的不足和改进空间 37](#_Toc513539829)

# 概述

## 项目简介

本项目是一个大型建筑电梯模拟器系统，项目基于Java语言编写，使用了JavaFX框架以实现图形化用户界面（GUI）。

## 项目目的

本项目的编写目的在于通过对电梯系统的运行状况的调度实现对一个多线程系统的模拟，进而提升多线程的理解和运用。

## 项目主要运用到的技术

在项目整体的编写过程中，使用了Java语言的多种特性，同时结合了最新的Java标准（编写平台基于Oracle于2018年3月发布的Java 10[1]，要求运行平台至少具有Java 1.8以上的运行环境）。同时，在GUI编写工作进行的过程中采用了JavaFX模块+JavaFX Scene Builder的可视化编程技术，有效地减轻了GUI编制工作的难度，同时优化了项目的效果。

在对电梯系统的模拟中，采用了MVC（Model-View-Controller）模型，将View模型交给JavaFX制作，建筑的模拟则封装在Model中，同时为了准确，将每一个电梯模型封装为一个线程，通过一个独立的Controller线程加以控制，形成运行性能良好、模拟效果优秀的项目模型。

# 项目内容

## 类简介

### cn.leonwong.ElevatorSimulator.Model 包

cn.leonwong.ElevatorSimulator.Model包（下简称Model包）包含了所有建筑的相关信息构成的模型，包括Elevator（电梯）类、Building（建筑）类、Passenger（乘客）类和Message（消息）类

#### Building类

Building类的成员：

公有：  
/// Note for each elevator in this building  
public Vector<Elevator> elevatorList;  
/// Note for persons waiting for elevators in each level  
public Vector< Vector<Passenger> > levelList;  
/// A message center noting for events  
public Vector<Message> messageCenter;

*/\*\*  
 \* Creates a building  
 \** ***@param*** *levs \#levels in ths building  
 \** ***@param*** *elevs \#elevators in this building  
 \** ***@param*** *maxPass max \# of passengers an elevator can contain  
 \*/*public Building(int levs, int elevs, int maxPass)；

*/\*\*  
 \* getter for numbers of elevators  
 \** ***@return*** *numbers of elevators in this building  
 \*/*public int getElevators()；

*/\*\*  
 \* getter for numbers of levels  
 \** ***@return*** *numbers of levels in this building  
 \*/*public int getLevels();

私有成员：  
/// \#Levels of this building  
private int levels;  
/// \#Elevators of this building  
private int elevators;  
/// Note for each elevator in this building  
/// A lock used to help lock levelList  
private ReentrantLock lock;

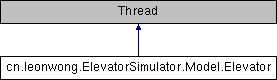
Building类的作用：

Building类模拟了一栋建筑，保存了建筑整体的有关信息（楼层数、电梯数、每一层楼的状况等等），并且利用了Java中Vector容器的线程安全性和ReentrantLock重入锁来保护楼层数据的安全。

#### Elevator类

继承关系：

Elevator类继承了Thread类。



Elevator类的成员：

公有成员：

/// time interval for going to another floor  
public static final int *FLOOR\_INTERVAL* = 500;  
/// time interval for a passenger entering/leaving  
public static final int *PASS\_INTERVAL* = 200;  
  
*/\*\*  
 \* A class contains to directions in order to indicate direction  
 \*/*public static final class Direction;  
*/\*\*  
 \* Start this Thread  
 \*/*@Override  
public void start();

*/\*\*  
 \* Run function  
 \*/*@Override  
public void run();  
@Override  
public String toString();

*/\*\*  
 \* create a new elevator  
 \** ***@param*** *name the in dex of this elevator  
 \** ***@param*** *max the max level of the building  
 \** ***@param*** *maxPass the capacity of this elevator  
 \** ***@param*** *levs a list for passengers in each level  
 \** ***@param*** *mess message center  
 \** ***@param*** *l a reentrantlock to lock level list  
 \*/*public Elevator(int name, int max, int maxPass, Vector< Vector<Passenger> > levs, Vector<Message> mess, ReentrantLock l);

*/\*\*  
 \* decide whether this elevator is full  
 \** ***@return*** *true if this elevator is full, false otherwise  
 \*/*public boolean isFull();

*/\*\*  
 \* decide whether this elevator is empty  
 \** ***@return*** *true if this elevator is empty, false otherwise  
 \*/*public boolean isEmpty();

*/\*\*  
 \* decide whether this elevator is idle  
 \** ***@return*** *true if this elevator is idle, false otherwise  
 \*/*public boolean isIdle();

*/\*\*  
 \* add a destination for this elevator  
 \** ***@param*** *dest the destination level to be added  
 \*/*public void addDestination(int dest);

*/\*\*  
 \* getter for the current level this elevator is at  
 \** ***@return*** *the current level this elevator is at  
 \*/*public int getLevel();

*/\*\*  
 \* getter for the current direction of this elevator  
 \** ***@return*** *the current direction of this elevator  
 \*/*public int getDirection();

*/\*\*  
 \* getter for the biggest level this elevator is heading for  
 \** ***@return*** *the biggest number of level in destination list  
 \*/*public int getMaxDestination();

*/\*\*  
 \* getter for the lowest level this elevator is heading for  
 \** ***@return*** *the smallest number of level in destination list  
 \*/*public int getMinDestination();

*/\*\*  
 \* getter for the numbers of passengers in this elevator  
 \** ***@return*** *the number of passengers in this elevator  
 \*/*public int getPassengers();

*/\*\*  
 \* directly move this elevator to a level  
 \** ***@param*** *l the destination level  
 \*/*public void setLevel(int l);

*/\*\*  
 \* used to stop this thread  
 \*/*public void stopThread();

*/\*\*  
 \* getter for a list of destinations of this elevator in a String  
 \** ***@return*** *the String of list of destinations  
 \*/*public String getDestinations();

*/\*\*  
 \* getter for the destination list's size  
 \** ***@return*** *this size of this destination list  
 \*/*public int getDestinationSize();

私有成员：

/// Note for the direction this elevator is heading  
private int direction;  
/// When changing direction is needed, add another flag  
private int nextDire;  
/// Thread for an elevator  
private Thread t;  
/// Note for the index of this elevator  
private int index;  
/// Note for person waiting for elevators in each level  
private Vector< Vector<Passenger> > levelList;  
/// Note for max level available  
private int maxLevel;  
/// Note for maximum numbers of passengers available in this elevator  
private int maxPassenger;  
/// Note for which level this elevator is at  
private int level;  
/// Note for messageCenter  
private Vector<Message> messageCenter;  
/// Note for each passenger in this elevator  
private Vector<Passenger> passengerList;  
/// Note for each level that this elevator should stop and open  
private TreeSet<Integer> destinations;  
/// Note for the lock used to lock levelList  
private ReentrantLock lock;

*/\*\*  
 \* add one passenger into this elevator  
 \** ***@param*** *lev where the passenger entered the elevator  
 \** ***@param*** *pass an object refering to the passenger  
 \** ***@return*** *true if the passenger entered this elevator successfully, false otherwise  
 \*/*private synchronized boolean passengerEnterElevetor(int lev, Passenger pass)；

*/\*\*  
 \* remove a passenger from this elevator  
 \** ***@param*** *lev the level that the passenger leaves at  
 \** ***@param*** *pass an object refering to the passenger  
 \** ***@return*** *true if the passenger removed successfully, false otherwise  
 \*/*private synchronized boolean passengerLeaveElevator(int lev, Passenger pass)；

*/\*\*  
 \* decide what direction this elevator should go after a level's all work done  
 \*/*private synchronized void decideDirection()；

*/\*\*  
 \* use this to ensure that every passenger's destination is in this destination list  
 \*/*private void decideDestinations()；

/// decide whether this thread has been stopped  
private boolean stop;

Elevator类的作用：

Elevator类采用线程机制，每一个线程模拟一台电梯的运行情况。

#### Message类：

Message类的成员：

公有成员：

/// used to indicate this message is describing a passenger leaving an elevator  
public static final int *passengerLeaveElevator* = 0;  
/// used to indicate this message is describing a passenger entering an elevator  
public static final int *passengerEnterElevator* = 1;  
/// used to indicate this message is describing an elevator has moved to another floor  
public static final int *elevatorChangeFloor* = 2;  
/// used to indicate this message is describing an elevator is idle  
public static final int *elevatorIsIdle* = 3;  
  
/// to denote which kind of message this one is  
public int mode;  
/// to denote the elevator that this message took place in  
public int destElevator;  
/// to denote the level that this message took place at  
public int destLevel;  
/// to denote the passenger related to this message  
public Passenger pass;  
  
*/\*\*  
 \* to build a new messafe  
 \** ***@param*** *modeOfMessage the kind of message  
 \** ***@param*** *elevatorSender the elevator this message took place in  
 \** ***@param*** *levelHappened the level this message took place at  
 \** ***@param*** *passenger the related passenger  
 \*/*public Message(int modeOfMessage, int elevatorSender, int levelHappened, Passenger passenger);

私有成员：

Message类没有私有成员。

Message类的作用：

每个Elevator线程触发一个电梯事件（进入电梯、离开电梯、电梯空闲、电梯移动）的时候都会构造一个新的Message类累计在MessageCenter里面，Controller类通过访问MessageCenter来进行界面的绘制，显示正确的动画。

#### Passenger类

Passenger类的成员：

公有成员：

/// the destination of this passenger  
public int destination;  
*/\*\*  
 \* create a new passenger heading for some floor  
 \** ***@param*** *dest the destination of this passenger  
 \*/*public Passenger(int dest);

私有成员：

Passenger类没有私有成员。

Passenger类的作用：

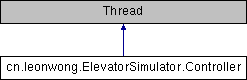
Passenger类模拟了一个前往某一层的乘客，这个乘客出现在某一层的层列表的时候会根据这个乘客的目的地的方向（向上还是向下）触发这一层的方向指示灯，乘客在某个电梯的时候会根据这个乘客的目的地楼层激活电梯内部的目的地列表。

### cn.leonwong.ElevatorSimultor中的其他类

#### Controller类：

继承关系：

Controller类继承了Thread类。



Controller 类的成员：

公有成员：

*/\*\*  
 \* an enum for dispathching strategies  
 \*/*public static class DispatchingStrategy;

*/\*\*  
 \* default constructor, set strategy to "Speed First" and thread running  
 \*/*public Controller();

*/\*\*  
 \* start this thread  
 \*/*@Override  
public void start();

*/\*\*  
 \* run this thread  
 \*/*@Override  
public void run();

*/\*\*  
 \* setter for message center  
 \** ***@param*** *mc the message center  
 \*/*public void setMessageCenter(Vector<Message> mc);

*/\*\*  
 \* setter fr view  
 \** ***@param*** *v the view  
 \*/*public void setView(View v);

*/\*\*  
 \* add a new passenger  
 \** ***@param*** *from the starting level of the passenger  
 \** ***@param*** *to the destination of the passenger  
 \*/*public void addPassenger(int from, int to);

*/\*\*  
 \* setter for building  
 \** ***@param*** *b the building  
 \*/*public void setBuilding(Building b);

*/\*\*  
 \* create a new building using the corrected params  
 \** ***@param*** *levs the total levels of this building  
 \** ***@param*** *elevs the numbers of elevators of this building  
 \** ***@param*** *cap the capacity of each elevator  
 \*/*public void createBuilding(int levs, int elevs, int cap);

*/\*\*  
 \* determine whether the building has been created  
 \** ***@return*** *true if the building has been created, false otherwise  
 \*/*public boolean isBuildingCreated();

*/\*\*  
 \* getter for total levels  
 \** ***@return*** *the number of levels  
 \*/*public int getLevels();

*/\*\*  
 \* getter for the numbers of elevators  
 \** ***@return*** *the number of elevators  
 \*/*public int getElevators();

*/\*\*  
 \* setter for strategy  
 \** ***@param*** *str the strategy  
 \*/*public void setStrategy(int str);

*/\*\*  
 \* randomly place elevators  
 \*/*public void randomizeElevators();

*/\*\*  
 \* used to stop this thread and its elevators  
 \*/*public void stopThread();

*/\*\*  
 \* getter for elevator list  
 \** ***@return*** *the elevator list  
 \*/*public Vector<Elevator> getElevatorList();

*/\*\*  
 \* getter for passenger list  
 \** ***@return*** *the passenger list  
 \*/*public Vector< Vector<Passenger> > getLevelList();

私有成员：

/// the current strategy  
private int strategy;  
/// message center  
private Vector<Message> messageCenter;  
/// an object trefering to view  
private View view;  
/// the controller's thread  
private Thread t;  
/// denote the building  
private Building building;

*/\*\*  
 \* arrage for an elevator to some level according to the dispatching strategy  
 \** ***@param*** *from the starting level  
 \** ***@param*** *to the destination level  
 \*/*private void arrangeForElevator(int from, int to);

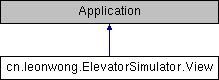
/// denote if this thread has been ended  
private boolean stop;

Controller类的作用：

Controller类是一个随着建筑构造而开始运行的线程，它的主要任务有两个：一是根据Model包累计的Message数据，进行GUI中建筑物的绘制；第二个任务是控制在添加乘客的时候根据此时的策略安排电梯进行接送任务。

#### View类：

View类继承了Application类。



View类的成员：

公有成员

*/\*\*  
 \* stop this thread  
 \*/*public void stopThread();

*/\*\*  
 \* start this application  
 \** ***@param*** *primaryStage the main stage  
 \** ***@throws*** *Exception if the application faces some errors starting itself, there will be some exceptions thrown  
 \*/*@Override  
public void start(Stage primaryStage) throws Exception;

*/\*\*  
 \* nothing to comment here...  
 \** ***@param*** *args command line arguments which are not used  
 \*/*public static void main(String[] args);

*/\*\*  
 \* re-draw an elevator at some level, should show or un-show them according to the page  
 \** ***@param*** *index the index of the elevator  
 \** ***@param*** *lev the new level the elevator is at  
 \*/*public void moveElevator(int index, int lev);

私有成员：

/// controller Object  
private Controller c;  
/// contains buttons for elevators  
private ArrayList<Button> GUIElevatorButtons;  
/// contains labels for elevators  
private ArrayList<Label> GUIElevatorLabels;  
/// contains buttons for levels  
private ArrayList<Button> GUILevelButtons;  
/// if there are two pages, indicate which one is being shown  
private Boolean isFirstPageShown;

*/\*\*  
 \* stop all threads on exit  
 \*/*private class ExitHandler implements EventHandler<WindowEvent>;

*/\*\*  
 \* show information of an elevator when clicking the elevator button  
 \*/*private class ElevatorButtonOnClickHandler implements EventHandler<javafx.event.ActionEvent>;

*/\*\*  
 \* show information when clicking the level button  
 \*/*private class LevelButtonOnClickHandler implements EventHandler<javafx.event.ActionEvent>;

*/\*\*  
 \* change the page showing  
 \*/*private class PageButtonOnClickHandler implements EventHandler<javafx.event.ActionEvent>;

*/\*\*  
 \* initialize elevators on the canvas  
 \** ***@param*** *elevs the number of elevators  
 \*/*private void initGUIElevators(int elevs);

*/\*\*  
 \* initialize levels on the canvas, should create page buttons according to the number of levels  
 \** ***@param*** *levs the number of levels  
 \*/*private void initGUILevels(int levs);

*/\*\*  
 \* show an elevator  
 \** ***@param*** *e the elevator label  
 \*/*private void showElevator(Label e);

*/\*\*  
 \* un-show an elevator  
 \** ***@param*** *e the label of the elevator  
 \*/*private void unshowElevator(Label e);

*/\*\*  
 \* set the page showing  
 \** ***@param*** *isFirstPageShownIndicator true if the first page is being shown, false otherwise  
 \*/*private void setIsFirstPageShown(boolean isFirstPageShownIndicator);

/// to write down the number of levels for the user  
@FXML  
private TextField numberOfLevelsText;  
  
/// to write down the number of elevators for the user  
@FXML  
private TextField numberOfElevatorsText;  
  
/// the button of creating a new building  
@FXML  
private Button createBuildingButton;  
  
/// to write down the capacity of the elevators for the user  
@FXML  
private TextField capacityText;  
  
/// when adding a new passenger, choose its starting level here  
@FXML  
private ChoiceBox<Integer> startingLevelChoiceBox;  
  
/// when adding a new passenger, choose its destination here  
@FXML  
private ChoiceBox<Integer> destLevelChoiceBox;  
  
/// the button to add a new passegner  
@FXML  
private Button addPassengerButton;  
  
/// list the three strategies and make the user choose by this ListView  
@FXML  
private ListView<String> strategyListView;  
  
/// show the help information of strategies button  
@FXML  
private Button strategyHelpButton;  
  
/// the button to add random passengers  
@FXML  
private Button testPassengerListButton;  
  
/// the canvas to draw the whole building  
@FXML  
private AnchorPane buildingCanvas;  
  
*/\*\*  
 \* when click on the create building button, create the building but firstly check the parameters  
 \*/*@FXML  
private void onClickCreateBuildingButton();

*/\*\*  
 \* handle the event that the user requires to add a new passenger  
 \*/*@FXML  
private void onClickAddPassengerButton();

*/\*\*  
 \* show the help info about strategies  
 \*/*@FXML  
private void onClickStrategyHelpButton();

*/\*\*  
 \* add random passengers  
 \*/*@FXML  
private void onClickTestPassengerListButton();

*/\*\*  
 \* change the strategy  
 \*/*@FXML  
private void onChangingStrategyList();

View类的作用：

View类是程序的入口，内部定义了大量的用户交互操作，主要负责和用户交互以及程序的初始化。

View类内部定义了大量的EventHandler接口的派生类，均为私有，它们用于处理各类事件。

## 项目运行机制简介

项目采用JavaFX包实现GUI。在程序载入后，提示用户新建一个建筑物。新建建筑物后，为每一个电梯创建一个线程，然后为Controller创建一个线程，并时刻允许用户增加一个乘客或批量随机增加乘客。在项目运行停止前，会根据情况安排电梯运行，并通过GUI和控制台告知用户运行状态。

# 项目操作示例

## 3.1 新建建筑

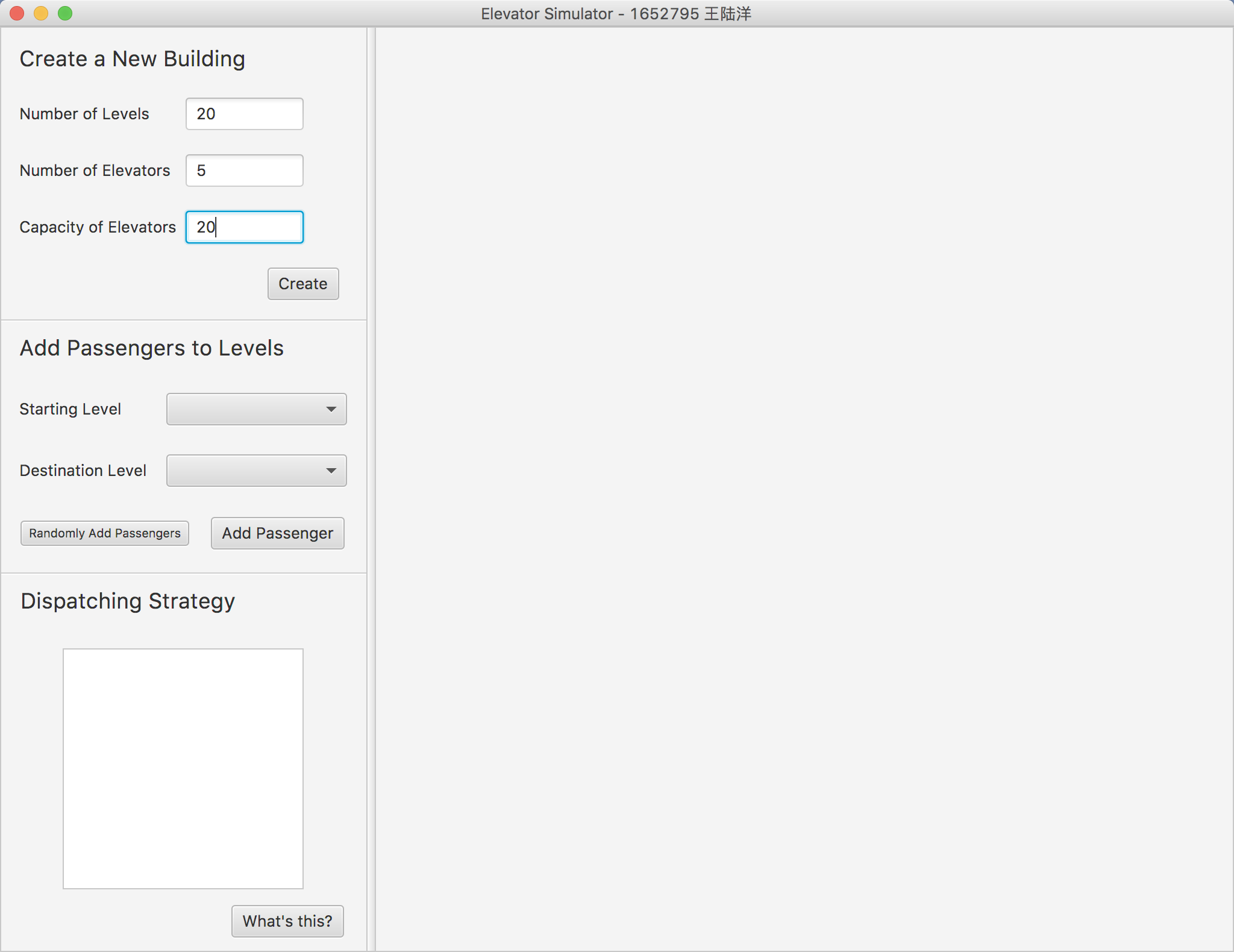
在程序运行后，填写楼层数、电梯数、电梯称重量（以人数计），然后点击“Create Building”按钮创建一个新的建筑。

流程图：

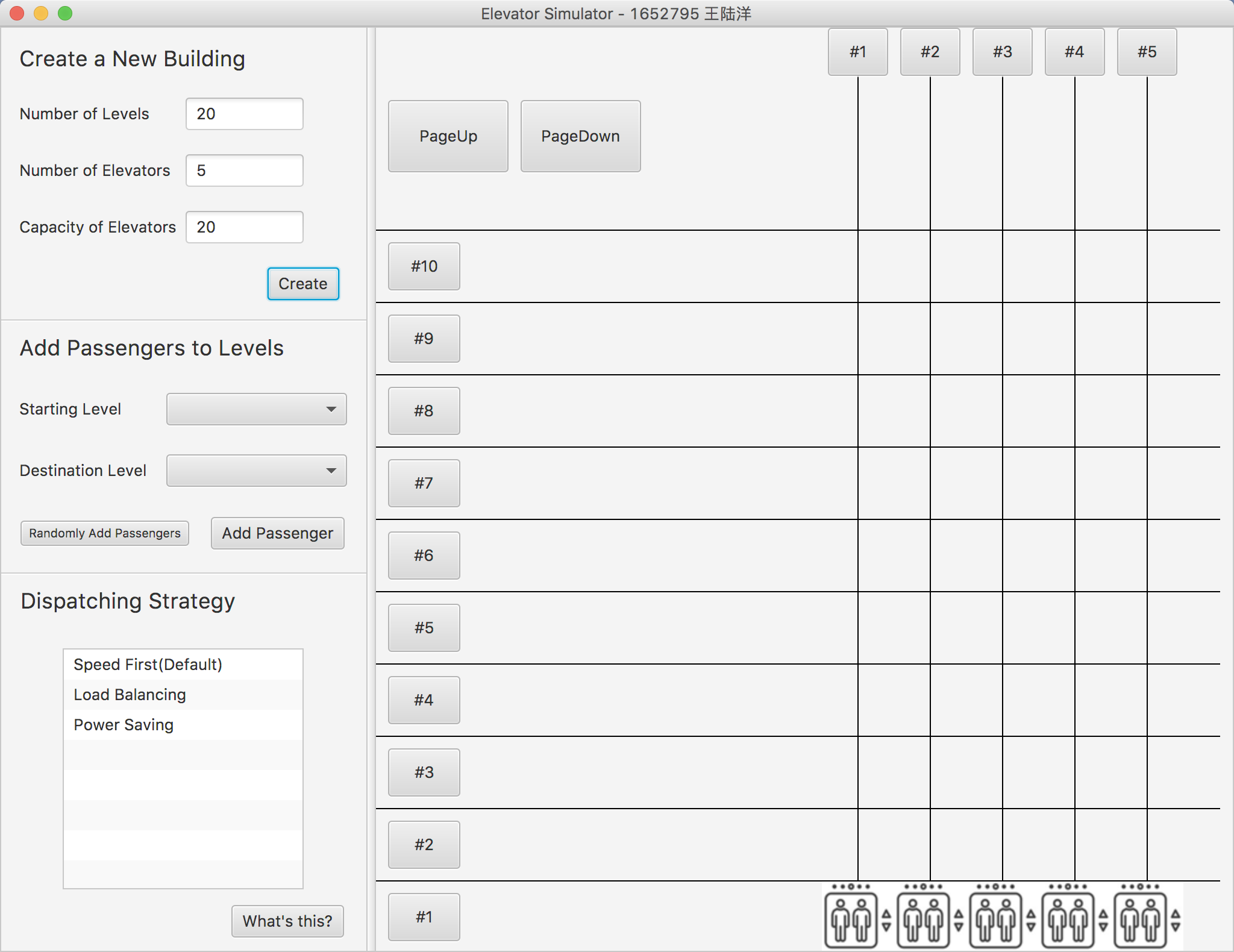


实例：创建一个拥有20层楼、5台电梯、每台电梯承重量20人的建筑。

1. 运行项目。
2. 在相应文本框中输入数据：



1. 点击“Create Building”按钮，程序将自动创建建筑并初始化GUI。



## 增加一个乘客

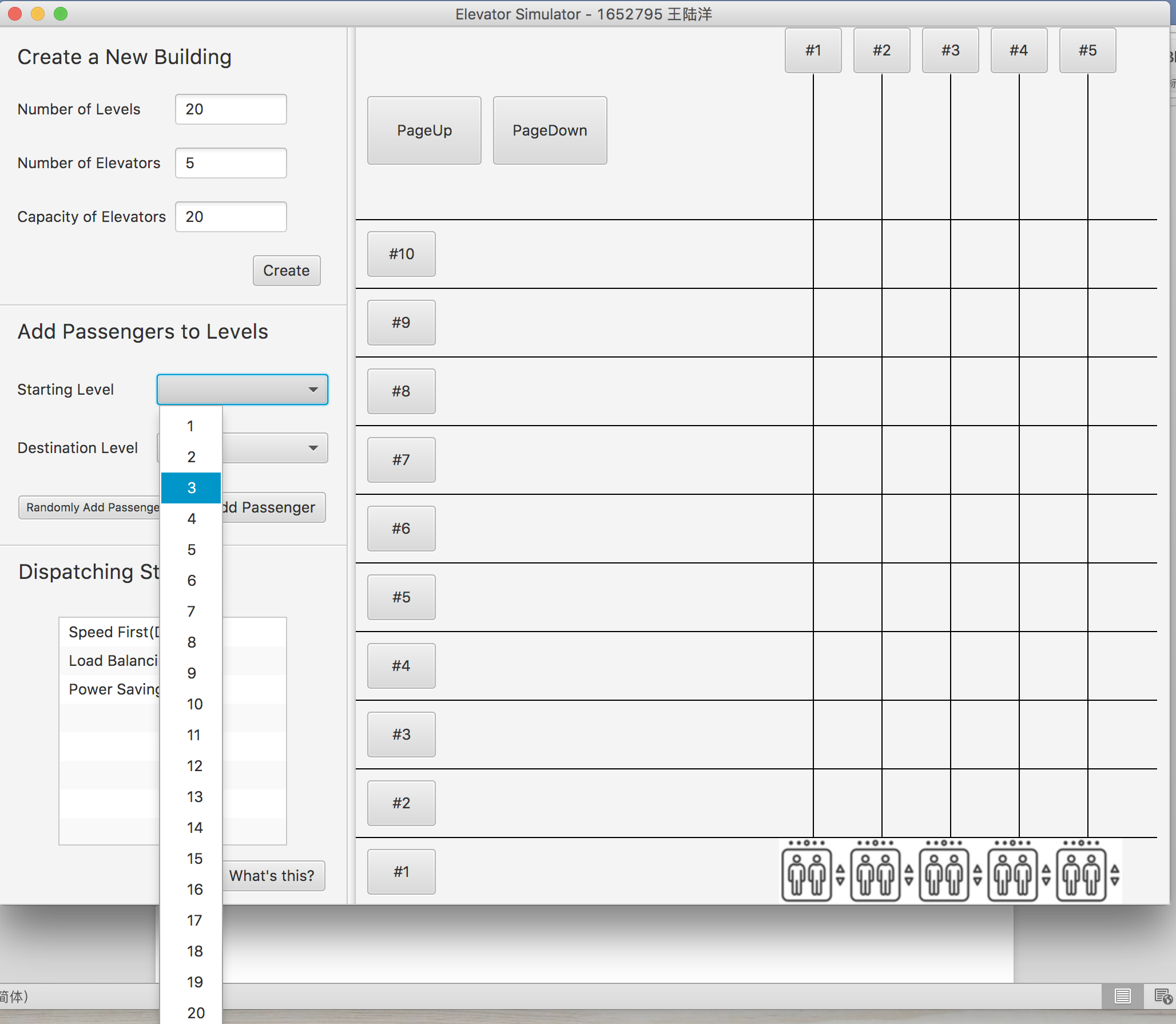
创建建筑完毕后，选择乘客的出发楼层和到达楼层，然后点击“Add Passenger”按钮，即可添加一个乘客。

流程图：

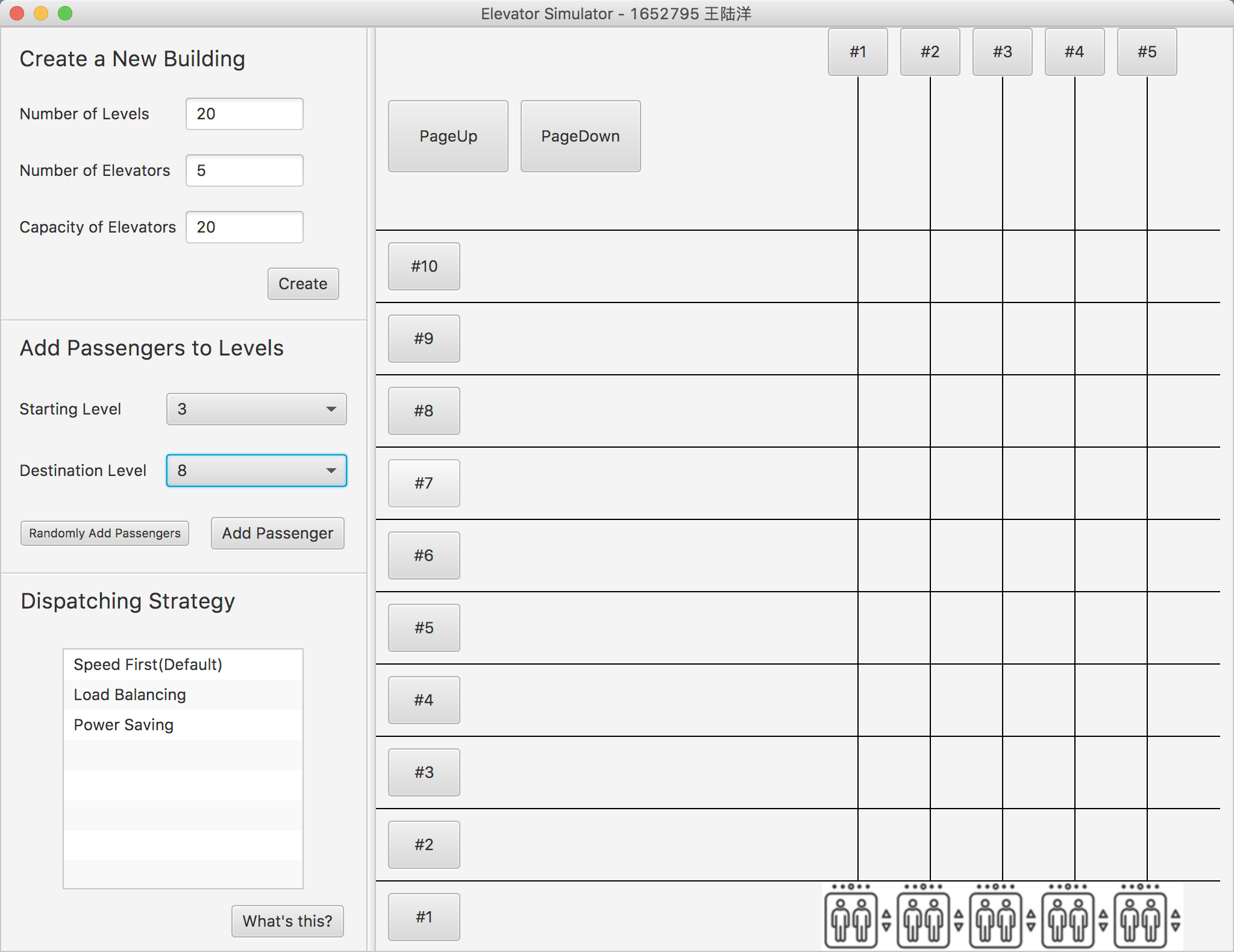


实例：在前述建筑的基础上，增加一个从3楼到8楼的乘客。

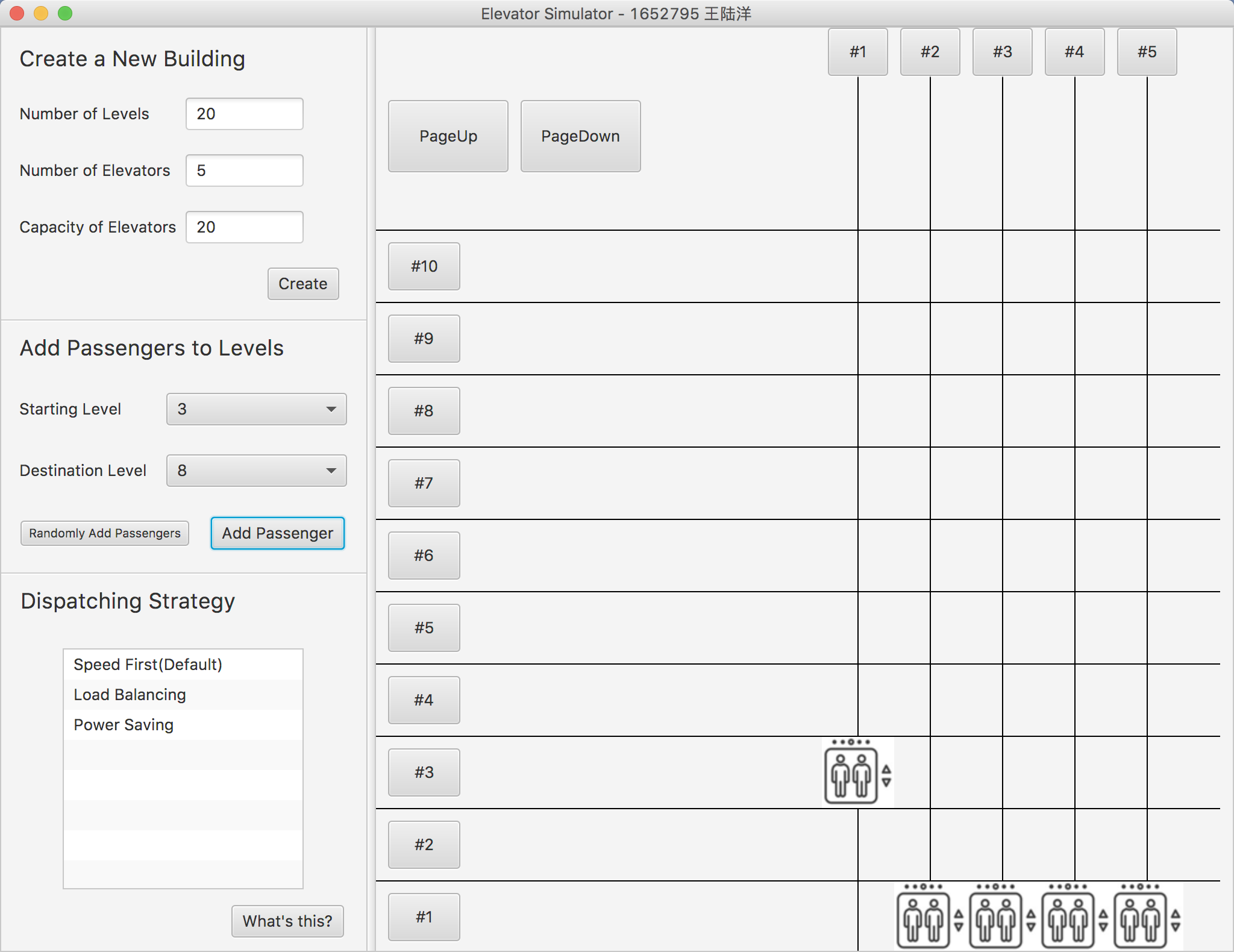
1. 在“Starting Level”中选择“3”。



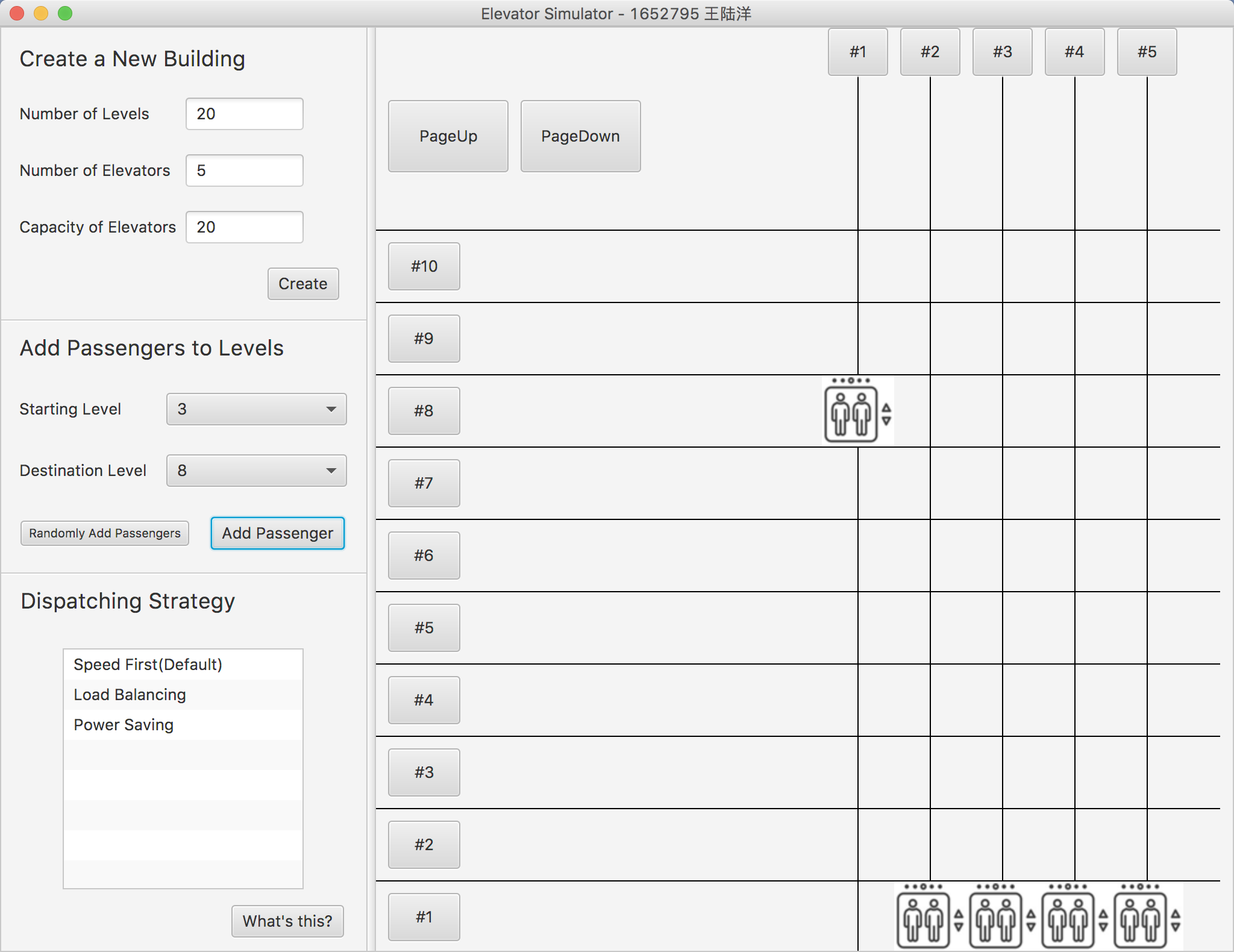
1. 在“Destination Level”中选择“8”，然后点击“Add Passenger”。



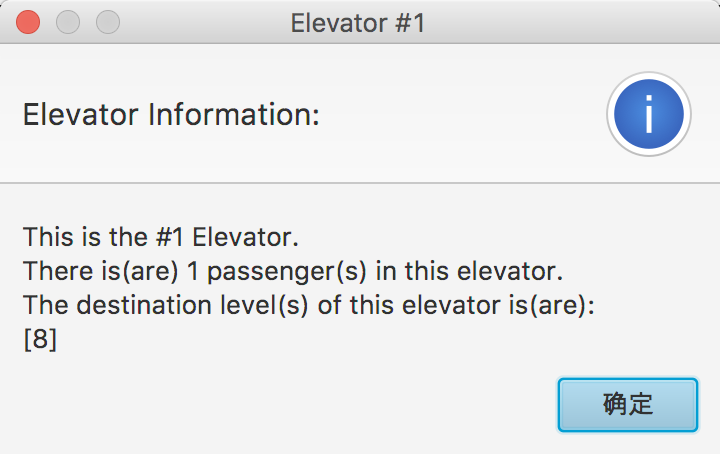
1. 可以看到电梯#1已经出发去接该乘客。



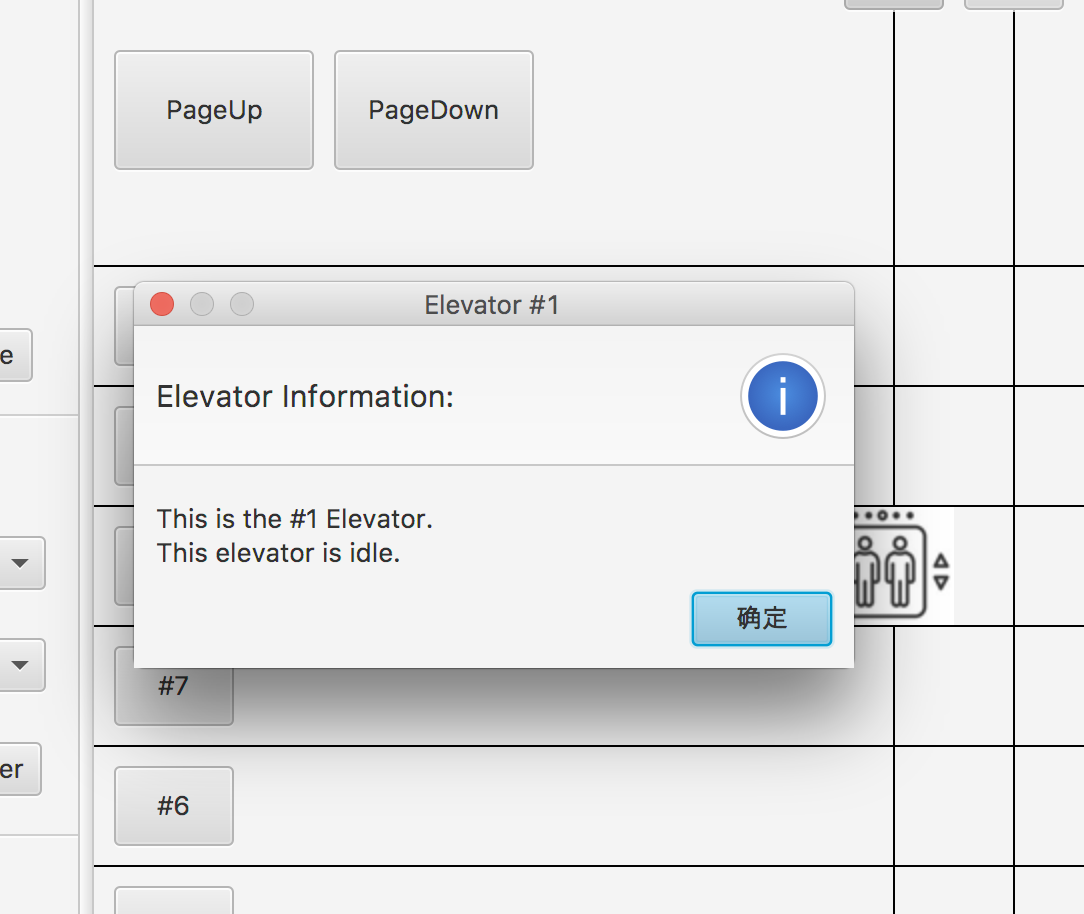
1. 当#1电梯抵达8楼时，可以看到#1电梯停止运行。



1. 在电梯#1运行的过程中，点击电梯#1的按钮，可以查看到电梯#1此时的信息，包括乘客数量和目标楼层。



1. 在电梯#1执行任务完毕后，点击电梯#1的按钮，可以查看到#1已经处于空闲状态。



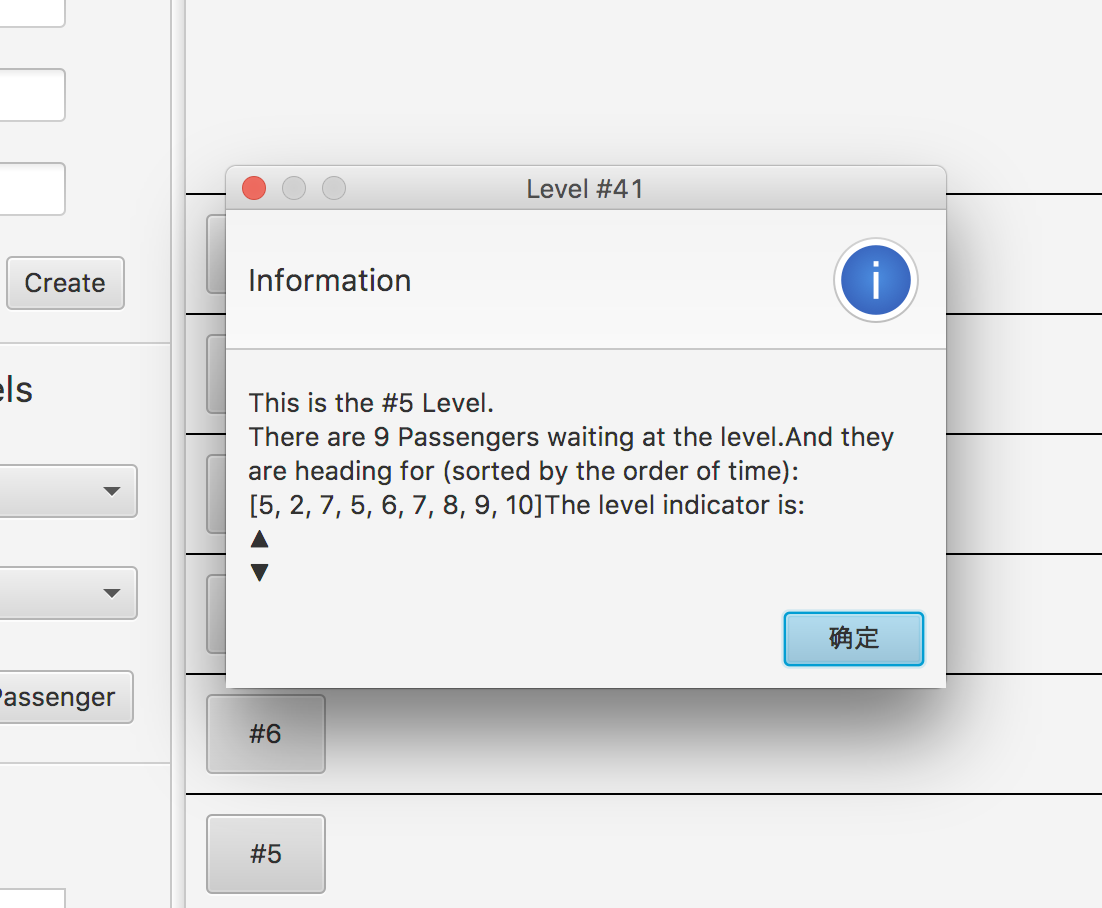
## 随机批量增加乘客

创建好建筑后，点击“Randomly Add Passengers”按钮，可以批量添加乘客。

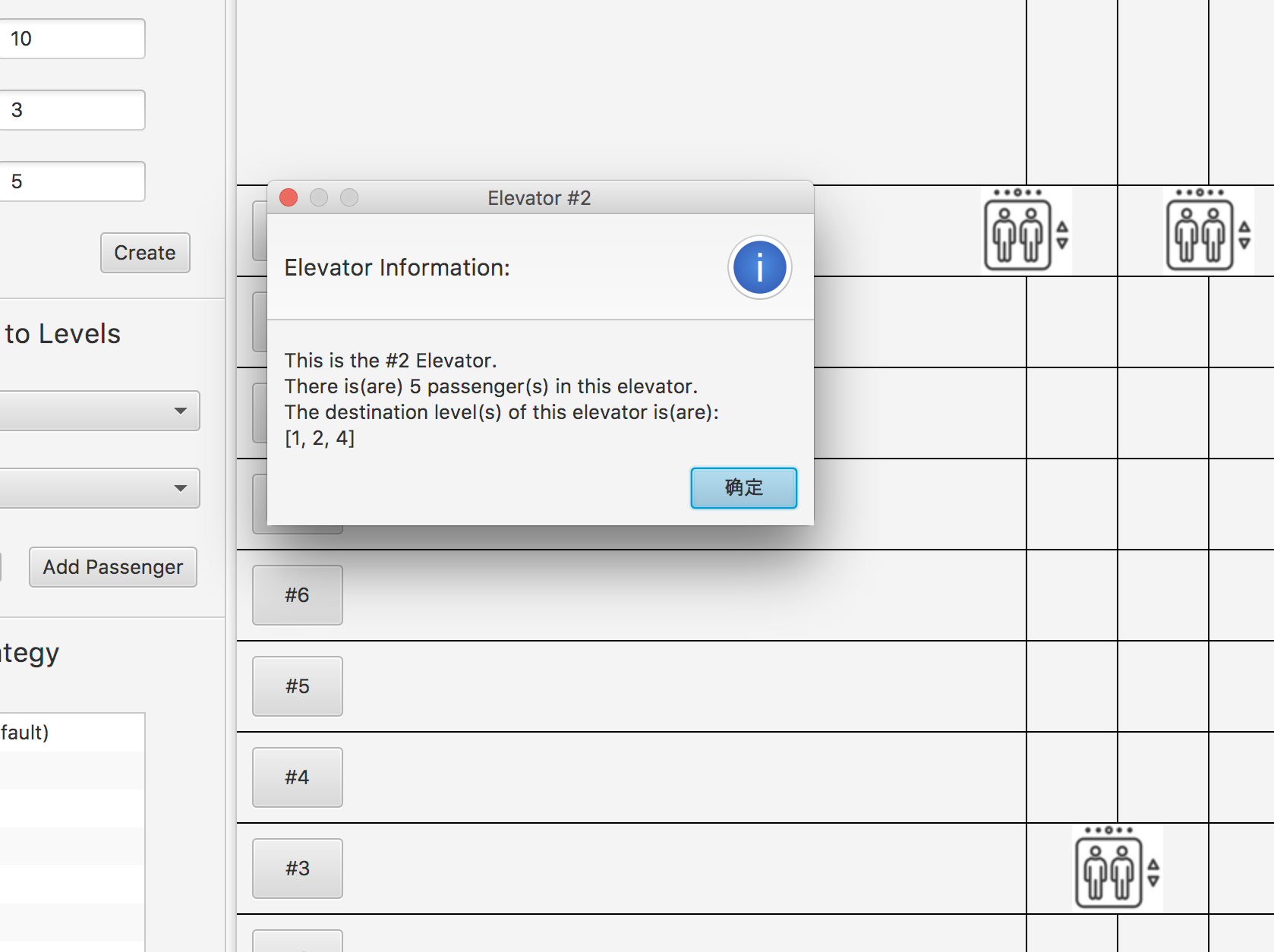
由于随机生成的乘客不需要用户指定参数，流程图较简单，略。

实例：在一个10层楼、3部电梯、电梯载重5人的建筑中随机增加乘客，然后观察效果。

1. 根据前述流程创建10/3/5的建筑。
2. 点击“Randomly Add Passenger”按钮。
3. 此时点击#5楼层的按钮，可以看到这一楼层当前的乘客目的地和指示灯信息：



1. 此时点击#2电梯的按钮，可以查看#2电梯的状态：



## 修改分配策略。

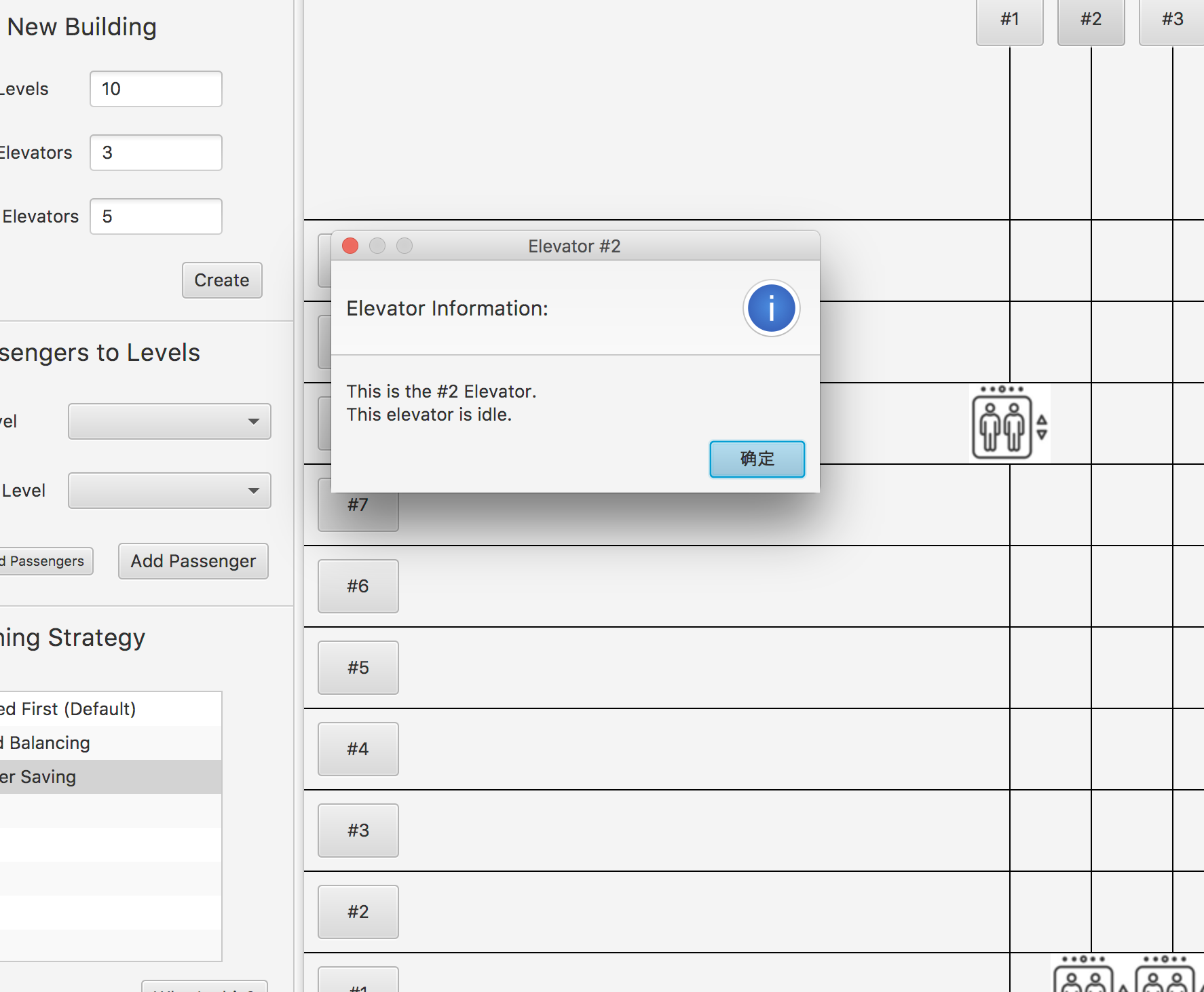
在电梯运行的任意时刻，用户都可以通过点击“Dispatching Strategy”列表中不同的分配策略来修改电梯分配任务的策略，已经分配的任务不受影响。

实例：在3.3的条件下，修改策略为省电模式“Power Saving”。

1. 设置分配策略为“Power Saving”。

2. 按照3.3节中的操作创建建筑并随机批量添加乘客。

3. 可以观察到系统选择只允许#1电梯工作，此时查看#2电梯状态，显示为空闲。



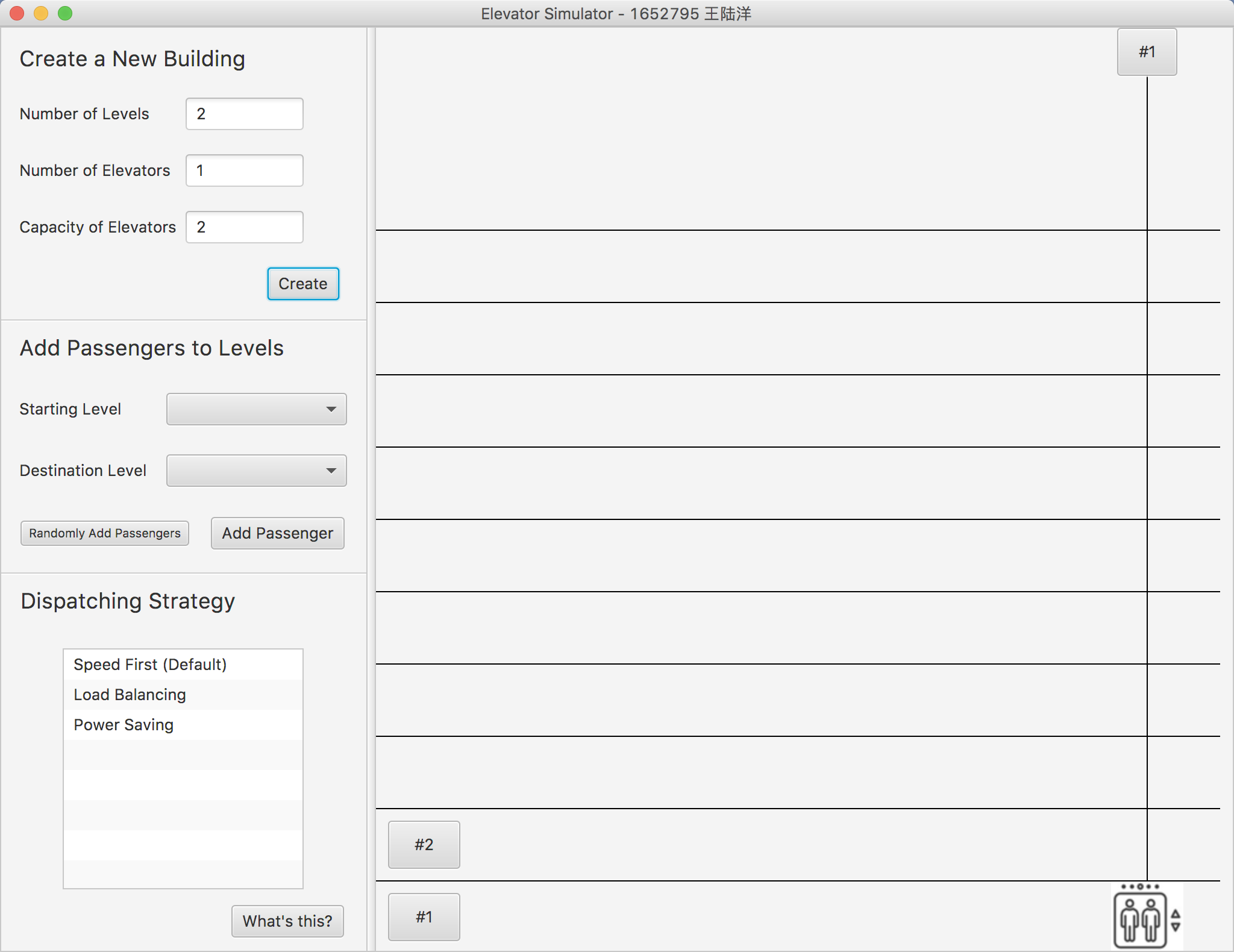
# 4. 程序测试

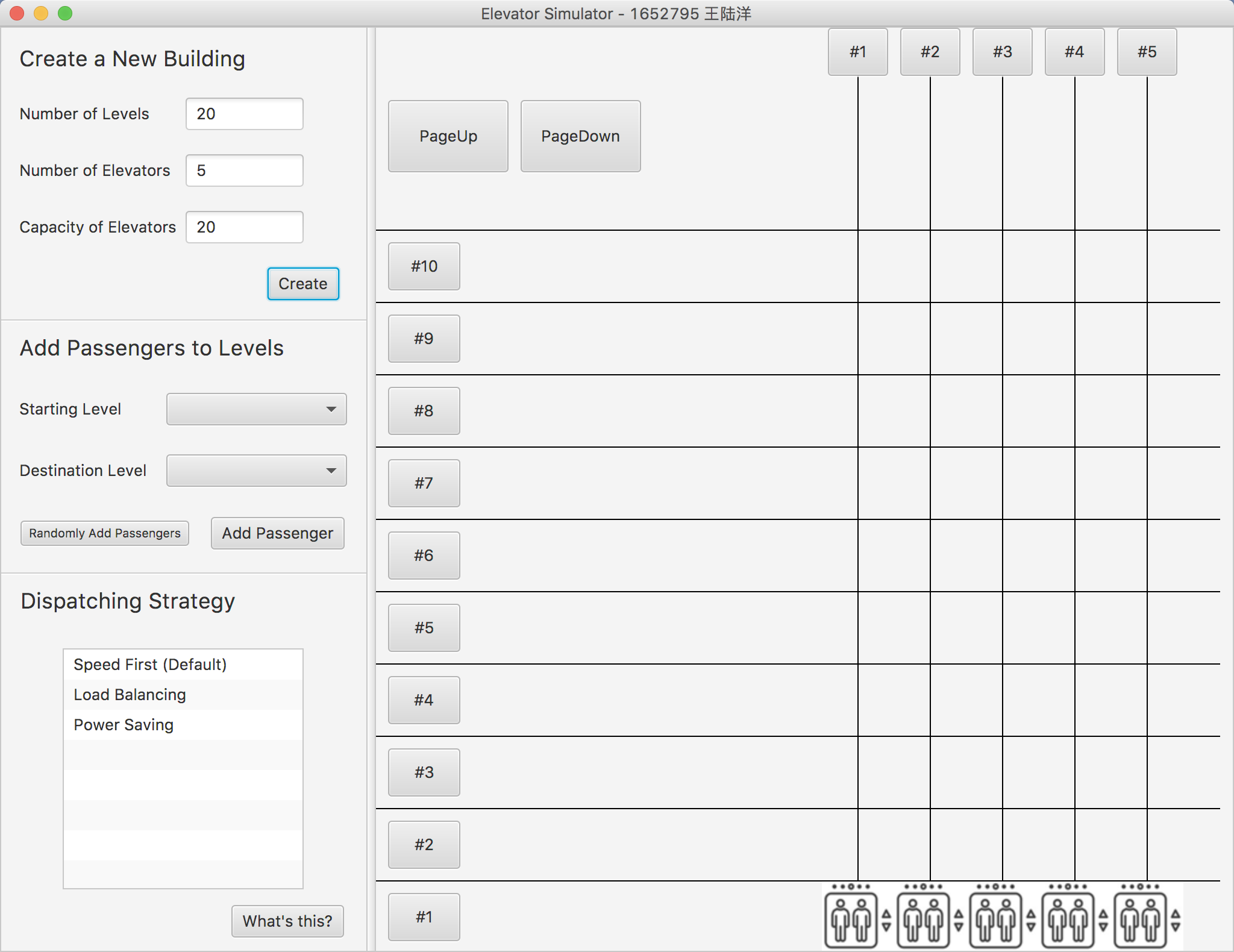
## 4.1 边界测试

### 4.1.1 建筑参数在边界上。

实例：由于本程序限制楼层数范围为2~20（含，下同），电梯数量为1~5，电梯承重量为1~20。故创建一个参数为2/1/1和参数为20/5/20的建筑。

程序运行结果：创建均成功。



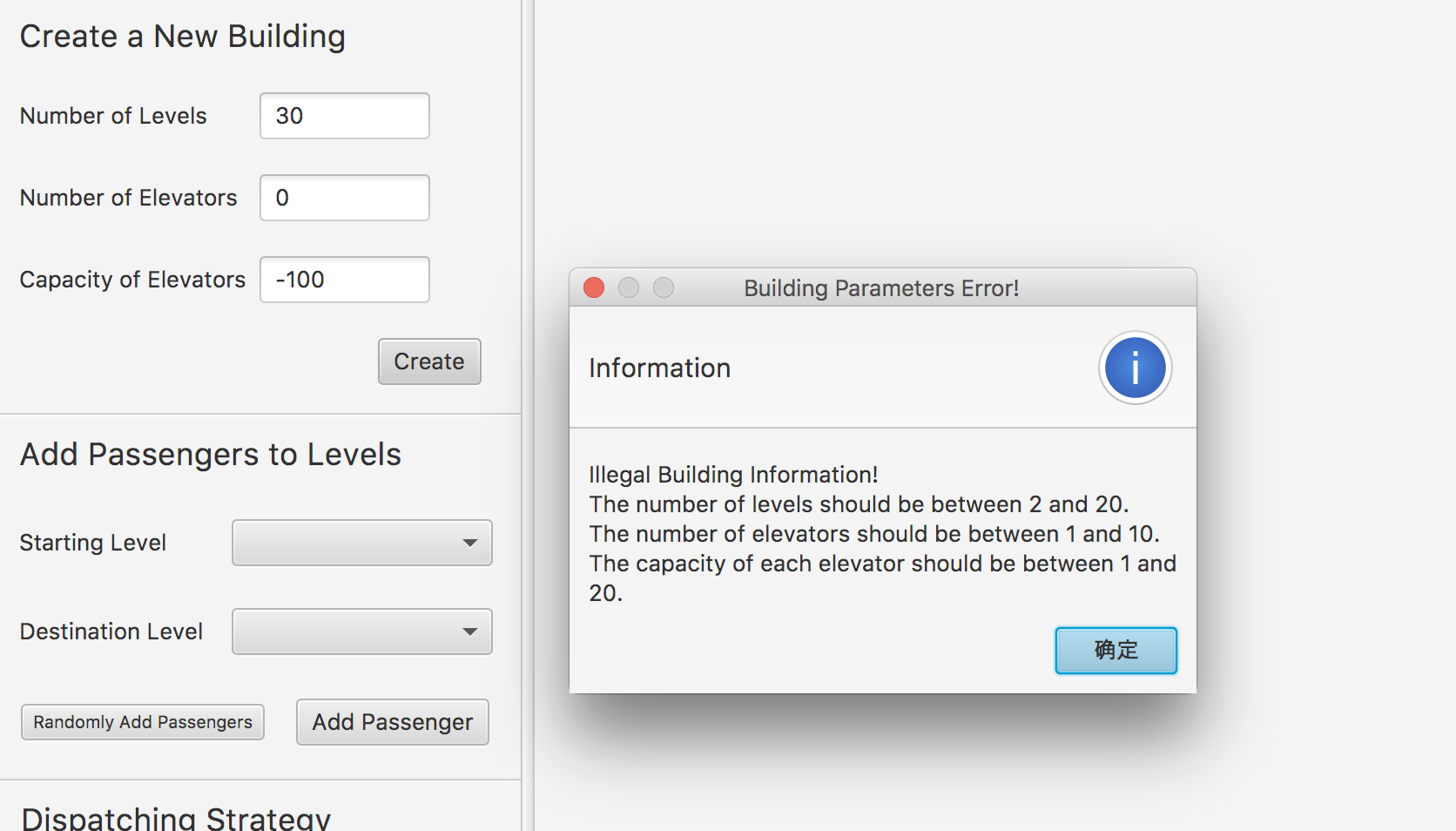


## 出错测试

### 建筑物参数错误

实例：创建一个参数为30/0/-100的建筑。

程序运行结果：提示用户检查数据，并拒绝创建电梯。

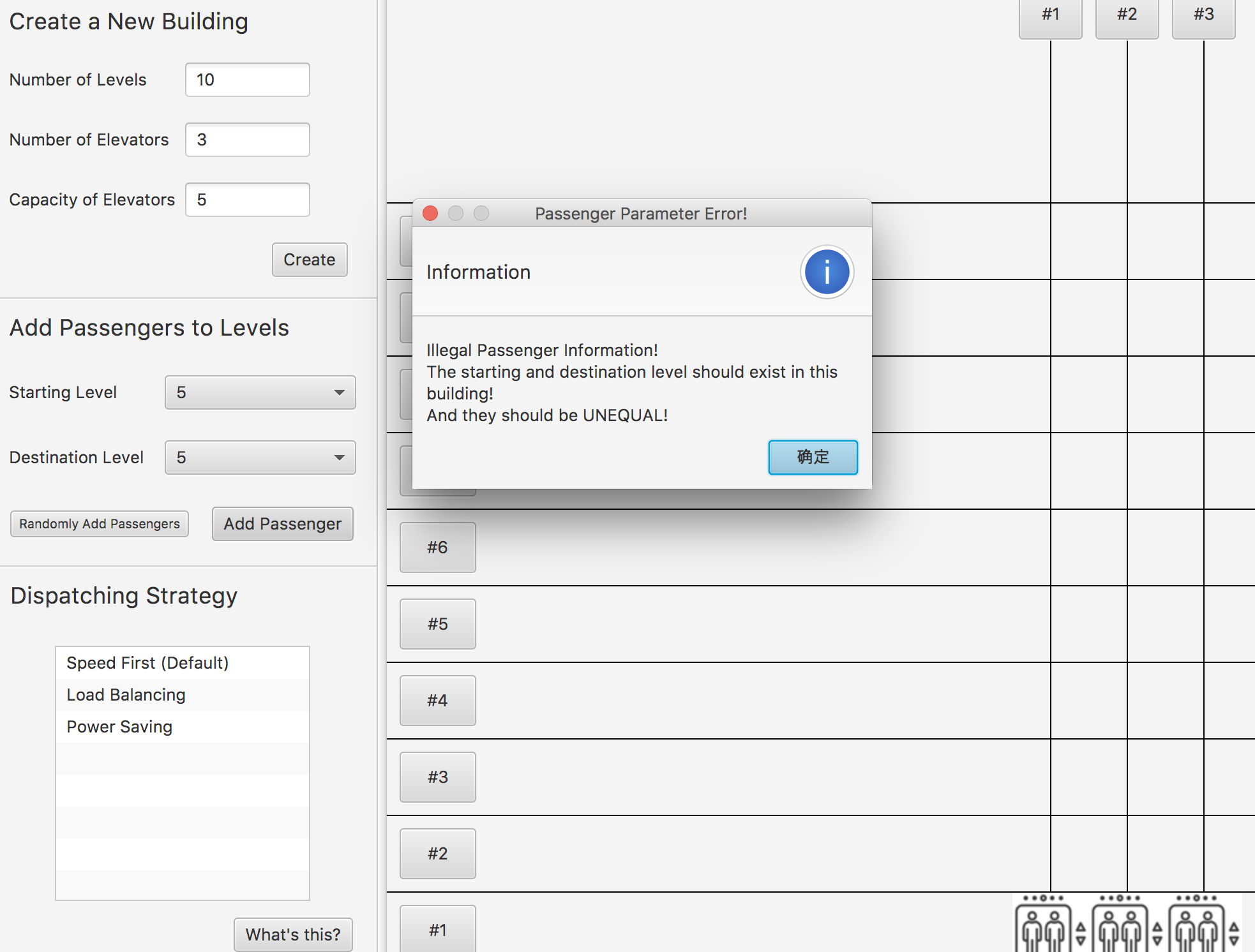


### 乘客参数错误

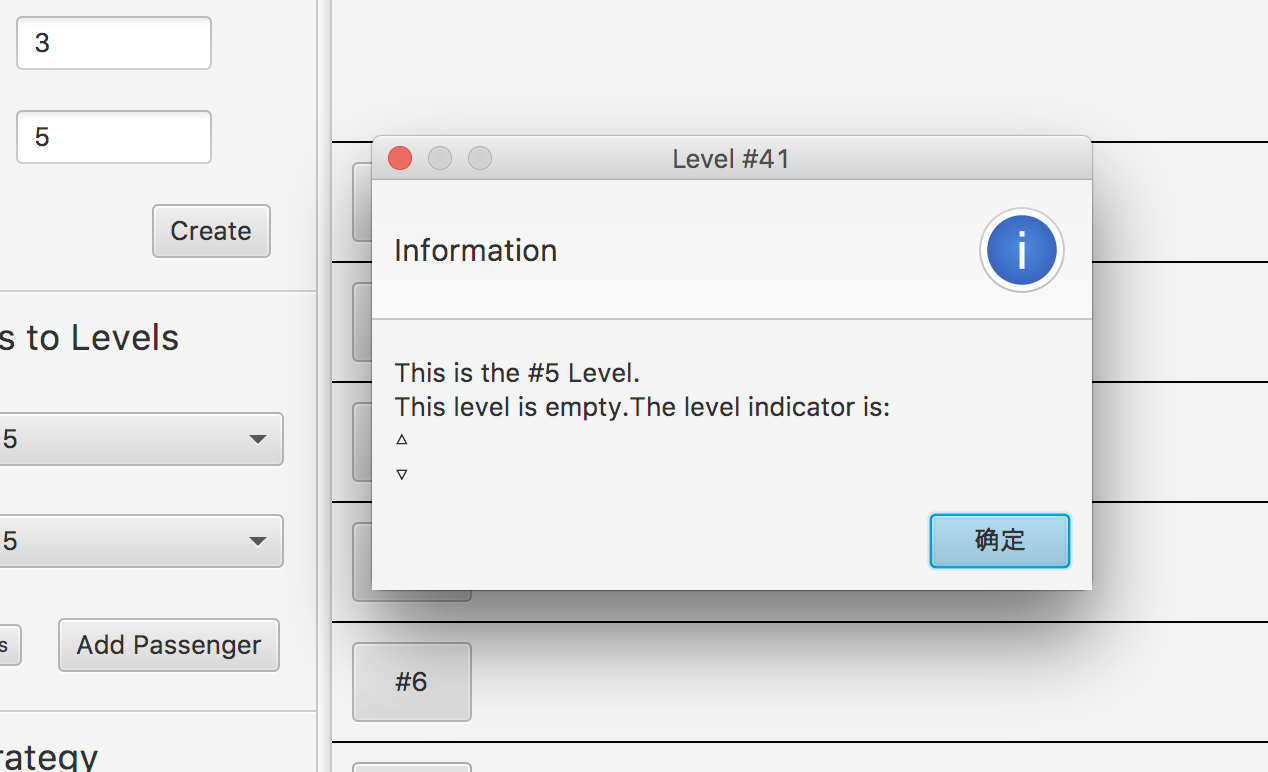
由于乘客的选择框不显示非法楼层数，所以本测试主要测试目的是验证乘客不能从某一层出发到同一层的情况。

实例：按照3.3节的参数创建建筑，然后添加一个从5层到5层去的乘客。

程序运行情况：程序要求用户检查输入，并且拒绝添加乘客。



此时可以检查#5楼层的状态，显示为没有乘客。



# 项目心得

本项目是我学习使用多线程技术的一个实践。在项目开发的过程中，采用了Java的线程实现，也学习到了GUI的制作和JavaFX、Scene Builder的使用。感谢张老师的悉心指导。

## 5.1 线程的运用

在这次项目的开发中，我体会到了多线程的特性，我在开发过程中遇到了一个重大bug——乘客列表的同时访问。

问题是这样的：我创建了一个参数为5/2/3的简单建筑，在#1楼层添加了去2、3、4层的三个乘客，两台电梯在这一层开始读取乘客列表，然后每个乘客都“进入”了#1和#2电梯，然后到达目标楼层后离开电梯，此时，每个乘客都“被分身”了。

后来发现我起初用于保存楼层乘客列表的ArrayList<Passenger>是线程不安全的，也就是没有进行同步处理，不具有synchronized关键字，而类似的容器Vector是线程安全的，遂将ArrayList更换为Vector。

然后遇到了第二个问题：并发读写抛出ConcurrentModificationException。查阅资料后发现这是由于并发读写，尤其是乘客进入电梯和离开楼层导致的同时读写楼层乘客列表导致的。于是我采用了ReentrantLock重入锁，在电梯进行乘客进入的时候对楼层乘客列表上锁，解决了乘客“被分身”的问题。

## GUI的编写

在GUI的编写过程中，我选择了采用之前没有使用过的JavaFX包，一个原因是我之前采用过另一个主流的GUI包swing，这次想学习新的知识，另一个原因是Scene Builder真的很好用。

这次我主要运用了Label、Button、AnchorPane、ListView、ChoiceBox等控件，掌握了它们的使用。

当然我的项目GUI还有不足的地方，一个是没有动画，电梯的移动看起来很生硬。

## MVC模式的运用

在这次项目的开发中，我选择了MVC模式，即Model-View-Controller模式，将建筑模型封装在Model包内部，对外只共享MessageCenter和开放几个必要的数据getter和setter，算是对Java和面向对象的一次致敬。

## 项目的不足和改进空间

第一点是没有动画。我在GUI绘制的过程中没有加入动画，而是直接对坐标进行操作，加上界面没有美化，看起来很直男。

第二点是开发后期对前期Model的内容引用太多，导致耦合程度增高。由于在开发过程中我经历了编译前加载测试样例-控制台输入输出-GUI输入控制台输出-GUI输入输出的阶段，在后期进行改进的时候不得已把一部分Model的内容直接改成了public，算是我再考虑欠佳，增高了耦合程度。

以上就是本次项目的文档，感谢您的悉心阅读！请您不吝赐教！