各模块设计说明

1. 主函数模块及模块调用关系

模块调用关系：

1. 主函数模块调用数据模块生成数据
2. 主函数模块调用主图形界面模块显示界面
3. 在主界面中调用策略模块生成路径，在主界面中实时模拟旅客位置
4. **if** \_\_name\_\_ == "\_\_main\_\_":
5. db\_file = os.path.join(os.path.dirname(\_\_file\_\_), 'travel\_query.db')
6. # 打开数据库
7. conn = sqlite3.connect(db\_file)
8. cursor = conn.cursor()
9. Data.load\_data(cursor)  # 从数据库加载数据
11. QtCore.QCoreApplication.setAttribute(QtCore.Qt.AA\_EnableHighDpiScaling)
12. # 适应不同分辨率
13. app = QApplication(sys.argv)
15. win = MainWindow(app)   # 新建窗口
16. win.show()  # 显示窗口
18. sys.exit(app.exec\_())
19. 数据模块
    1. 读取所有城市的信息：城市名、经纬度
    2. 读取班次信息：包括飞机、火车、汽车
    3. 处理所有数据，使得他们成为更容易使用的数据结构（如构成字典或列表型的数据结构）
20. city\_dangers = {} # 城市风险值
21. trans\_dangers = {} # 交通工具风险值
22. time\_table\_values = []  # 所有班次时间表
23. map\_values = []     # 所有城市的经纬度信息
24. map\_geo = {}        # 所有城市的经纬度，字典性的结构
25. all\_place = []      # 所有航班里的城市
27. **def** load\_data(cursor)  # 从数据库加载数据
28. 策略模块
    1. 实现计算旅行路程的功能
    2. 能够处理不同的策略和不同选择下的计算，返回计算出来的最优路径

算法说明：

1. 最小风险策略：无时间限制，风险最少即可

策略：使用A\* 算法，可以快速计算出最小路径，它相比dijkstra算法，多了启发式的预估移动耗费（从节点到终点的预估）。使用路径评分 F = G + H，G为从起点到节点的移动耗费，H为节点到终点的预估耗费。A\* 算法使用这种方式，有更好的先见之明，所以能够更快的寻找最短路径。相比BFS来说，它有更好的性能，因为它不需要遍历所有可能路径。

每调用一次A\* 算法，弹出从开启列表中加权危险值最小的节点（城市），然后计算从这个节点能够到达的相邻节点的加权危险值，在把他们加入到开启列表中。当弹出的节点为目的地时，即完成计算旅游路径。

1. 限时最少风险策略：在规定的时间内风险最少

策略：同最小风险策略，使用A\* 算法，但是我们在算法的过程中加入了限制条件，即在每次计算时，需要判断达到相邻节点时是否会超出限时的时间，若超出，则不把它加入到开启列表中。

具体A\*算法流程：

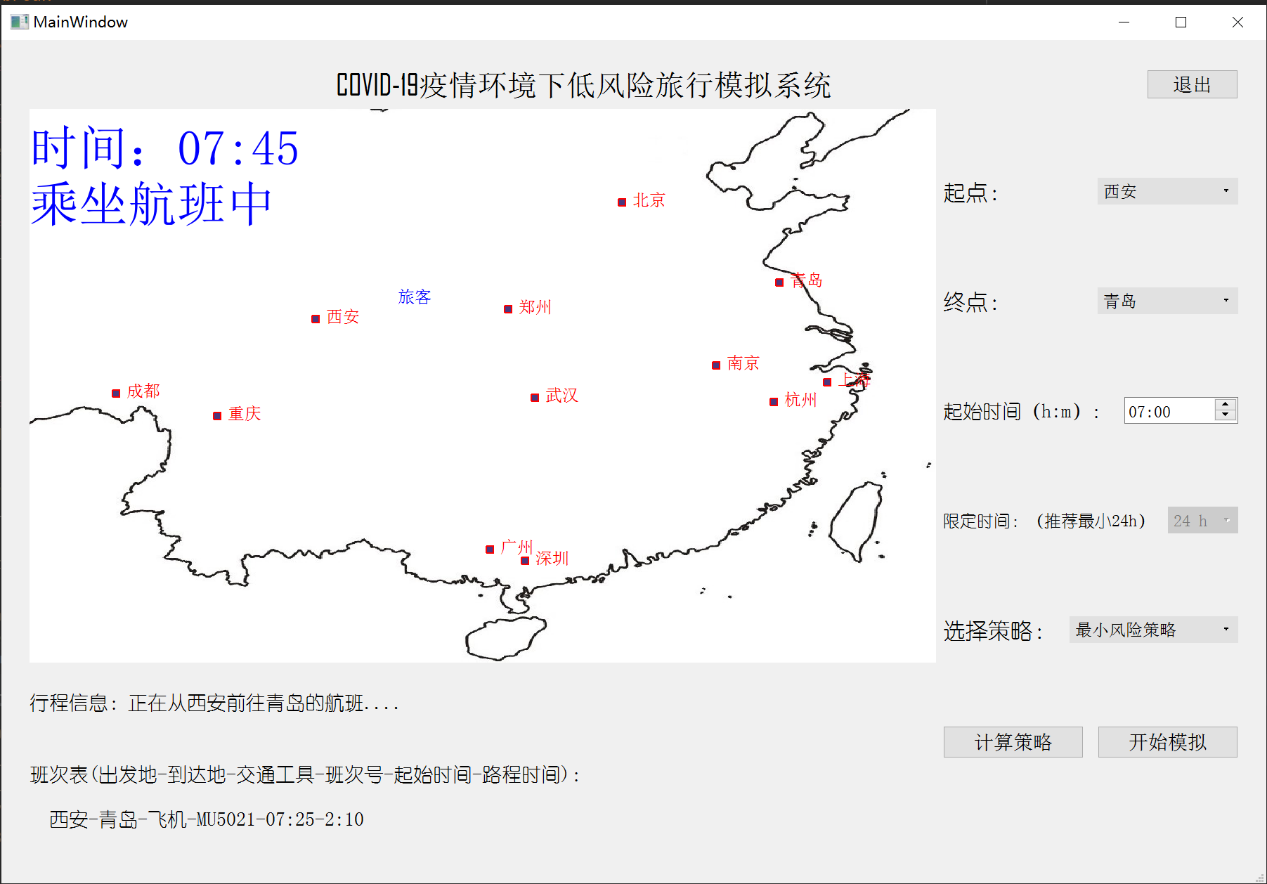
1. 把起点添加到开启列表
2. 重复以下工作：
   1. 弹出开启列表F值最低的节点，变为当前节点
   2. 如果当前节点在关闭列表，进入下一轮重复
   3. 如果当前节点为终点，返回最终计算的路线
   4. 把当前节点放入关闭列表
   5. 对可以到达的所有相邻节点：

如果不在开启列表或F值更低（加权危险值更低），则更新：

* 1. 设置这个相邻节点的父节点为当前节点
  2. 更新这个相邻节点的加权危险值

1. **class** Solution:  # 根据不同的参数、策略返回最有路径
3. **def** \_\_init\_\_(self):
4. self.start\_place = "北京" # 默认参数
5. self.end\_place = "广州"
6. self.start\_time = "7:00"
7. self.limit\_time = -1 # 默认最小风险策略
8. **def** shortestPath(self, start\_place="北京", end\_place="广州",
9. start\_time="9:15", limit\_time=-1)  # 计算最优路径
10. **def** a\_star\_graph\_search(    # A\* 算法
11. start,
12. end,
13. start\_time,
14. limit\_time,
15. goal\_function,
16. successor\_function,
17. heuristic
18. ):
19. visited = set()  # 关闭列表
20. came\_from = dict()  # 标记父节点
21. dangers = {start: 0}  # 加权危险值
22. frontier = PriorityQueue()  # 开启列表（按照加权危险值排序）
23. current\_time = parse\_time(start\_time)
24. frontier.add([start, current\_time], priority=0)  # 出发点入开启列表，优先级最高
25. **while** frontier:
26. node, current\_time = frontier.pop()  # node="北京"（开启列表中优先级最高（加权危险值最低）的节点）
28. **if** node **in** visited:  # 已经在关闭列表中
29. **continue**
30. **if** goal\_function(node, end):  # 到达终点，返回路线
31. **return** reconstruct\_path(came\_from, start, node)
32. visited.add(node)  # 加入关闭列表
33. **for** successor\_node, successor\_info **in** successor\_function(
34. node, current\_time).items():  # 所有能够到达的后继节点
35. update\_t = update\_time(successor\_info[1])  # 更新时间
36. priority\_update = dangers[node] + successor\_info[0] +
37. heuristic(successor\_node, end)
38. # 计算优先级 F = G + H （到后继节点的危险值 + 从后继节点到终点的预估危险值）
40. **if** is\_in\_limit\_time(limit\_time, successor\_info[1]) **or**
41. limit\_time == -1:  # -1代表最小风险策略，否则使用规定时间的策略
42. frontier.add(  # 把后继节点添加到开启列表
43. [successor\_node, update\_t],
44. priority=priority\_update
45. )
46. new\_danger = successor\_info[0] + dangers[node]  # 经过node到达后继节点的加权危险值
47. **if** successor\_node **not** **in** dangers **or** new\_danger < dangers[successor\_node]:  # 经过node到达后继节点危险值更低
48. dangers[successor\_node] = new\_danger
49. time\_table\_index = successor\_info[1]
50. came\_from[successor\_node] = Data.time\_table\_values[
51. time\_table\_index]  # 更新父节点的指向
52. **return** reconstruct\_path(came\_from, start, end)
53. 主界面模块
    1. 模拟真实的地图，按照经纬度比例缩放显示所有城市
    2. 显示所有可以改变的参数，包括：起点、终点、起始时间、限时时间、选择策略
    3. 显示计算出来的旅行路径，并能够模拟真实时间，在城市之间移动
    4. 设置日志功能，能够输出到log日志文件中

图形界面示例如下：



1. **class** MainWindow(QMainWindow):      # 主图形界面
2. **def** \_\_init\_\_(self, qApp=None):  # 初始化
3. self.log\_file = open("log.txt", "w") # 日志文件
4. self.gui = uic.loadUi(ui\_file, self) # 界面设计
5. self.solution = Strategy.Solution() # 策略
6. self.map\_pic = QPixmap("./map.jpg") # 背景图
7. self.current\_x = 0 # 当前位置（经纬坐标）
8. self.current\_y = 0
9. self.start\_time = "" # 起始时间
10. self.path = [] # 计算出来的路径
11. self.set\_traveller = False # 是否模拟行程
12. self.waiting = False # 旅客是否在等待航班
13. self.current\_time = 0 # 当前时间
14. **def** set\_initial(self):  # 设置默认参数
15. **def** kind\_changed(self):     # 选择不同的策略
16. **def** compute\_clicked(self):  # 计算路径
17. **def** simulate\_clicked(self):     # 开始模拟
18. **def** simulation(self):       # 模拟旅客路径
19. **def** paintEvent(self, event):    # 刷新旅客位置、显示旅客状态
20. **def** drawText(self, qp):  # 画图，每次旅客位置更新，或初始化时重新画图
21. **def** quit(self):     # 关闭窗口并退出