Design

1 Common data structures

The data structures described here will be common to both the desktop and smartphone applications.

1.1 Graph representation

Lines 238 & 18, DesktopMapper/test/NodeClass.cs

Lines 31 & 56, SmartphoneMapper/SmartphoneMapper/NodeClass.cs

The basis of the path generation in the smartphone navigation system is **graph traversal**, by treating rooms/corridors as nodes and immediate connections between rooms/corridors as edges (weighted, undirected) of a graph. As such, one critical data structure required is a representation of the graph (vertices/edges). This will be implemented in the form of an **adjacency list**, wherein each node (element of the list) contains a list of tuples of the form (*x*, *y*) where *x* represents the connected node and *y* describes the weight of the respective edge between the two nodes (i.e. the time taken to travel between rooms). This is in contrast to an adjacency matrix, that is a two dimensional array where each index represents source/destination node and the respective element represents edge weight; although more time efficient, an adjacency matrix is highly memory-inefficient for sparse graphs such as those encountered in the context of buildings, where rooms are only connected to a few other rooms.

Each room (‘node’) also has several individual properties that need to be represented. These include:

* Name, description and associations/facilities (as shown on the ‘popup’ display in the smartphone UI section) (conditional on whether the node is going to be displayed, see below);
* Location of the room with respect to the background floor plan image (represented as a percentage offset from the bottom-left corner of the image);
* Floor on which the room is situated;
* Whether the room is to be visible (some nodes can be used in the path generation process even if they are not distinct areas, in which case they should be considered for graph traversal but not for display; observe the ‘route’ display in the smartphone UI section).

Collating the above, this will be implemented by considering a ‘*Node*’ class. The properties of this class are;

* OffsetPosition (an **integer tuple** of form (*x*,*y*) where *x* represents the lateral offset from the left of the floor-plan image as a percentage of the total width of the image, and *y* represents the same for vertical offset from the bottom as a percentage of the image height);
* Floor (**integer**, the key of the current floor; see below regarding floor representation);
* Visible (**boolean** variable corresponding to whether the node is to be treated as a visible node (such as a room/corridor));
* Description (a **string tuple** of form (*a,b,c*) where *a* represents the name of the room, *b* represents the description of the room, and *c* represents the associations of the room (e.g. teachers/facilities available));
* AdjacencyList (a **list** of **integer/floating point tuples** of form (*x,y*) where *x* represents the node ID (index of the node in a list of nodes, see below) of a connected node and *y* represents the weight of the edge between the two nodes).

A list of ‘nodes’ can therefore be constructed by having each element of the list be a new instance of the class described above, with properties assigned at runtime.

1.2 Miscellaneous

Lines 291, DesktopMapper/test/NodeClass.cs

Line 267, SmartphoneMapper/SmartphoneMapper/NodeClass.cs

Other generic data required to represent an entire building includes:

* Background images of individual floor plans;
* Images illustrating each room;
* A list of floors accessible in the building.

The last of these, the list of floors, will be mutable during layout creation on the desktop tool and therefore require more thought; see section 2.3.4 for dealing with floor mutability. Broadly, the list will contain strings as values, with the strings representing the filename of the background image of that floor; when the floor background image needs to be loaded, this filename is used in reference to which floor image to load.

1.3 Storage

Lines 48, 53 & 24, DesktopMapper/test/InputOutput.cs

Line 103, SmartphoneMapper/SmartphoneMapper/InputOutput.cs

Collating all of the above, the storage of data structures used will be implemented as follows (illustrated in a directory-based filesystem, given that this will be the nature of the smartphone application storage as well as the temporary storage of the desktop application, i.e. in its folder under AppData/Local/Temp, before archiving);

* For the graph, a ‘node’ list as described above serialised as JSON and saved as a file in the building’s master directory.
* For miscellaneous data, such as the layout name and floor list, an anonymous object is created where each parameter is the miscellaneous datum to be saved (name, floor list etc). This object is then serialized into JSON and saved as a file in the building’s master directory.
* For floor-plan background images, a sub-directory containing floor-plan background images in the PNG format with filename (ignoring file extension) matching the integer of the floor it illustrates (i.e. index in the floor list).
* For room-specific images, a sub-directory containing images illustrating each room in the PNG format with filename (ignoring file extension) matching the index of the node it represents in the adjacency list.