**System Maintenance Document**

**System Overview**

This is the system maintenance document for the timetabler program. The timetabler program is a system which is designed to be used by a school to allocate staff and classrooms into pre-timetabled lessons, while trying to reduce the distance travelled by staff members. The system store data about the school: subjects, staff, sets, year groups, classes, lessons and layout, in a MariaDB relational database. Each to the described types of data is stored in their own table, and liked using primary and foreign keys. A copy of the initialization SQL script will be in this document in order to better show the layout of the database.

In order for a new entry to be added to any of the tables, first the data is extracted from a HTML5 Bootstrap modal using Javascript, where initial data type checks are carried out. This is then passed back through to Java using JavaFX’s JSObject, since the user interface is a HTML5 web page running in a JavaFX WebView. Once Java has the data, it is converted into the correct Java object, before being passed to the corresponding data access object, held by a data access object manager, where it is passed into a Java DataBase Connectivity (JDBC) PreparedStatement, which is sent to the MariaDB server. The server then returns the generated unique ID, which is passed back up through to Javascript, where it adds the data with the ID to the correct HTML table for the user to see.

Once the user specifies to location of the highest level map, and the lower maps, the system parses the highest map into a SchoolMap object, which wraps a 2D array of CellTypes. All maps are in the CSV format, where an empty cell is a Wall, a ‘0’ is a path, a building is just a name and a classroom is in the form C-*name*-*subject* where name is the name of the room and subject is the name of the subject taught in the room. The lower maps are loaded in after the highest map loads, and then each building in the collections of maps is initialised using the loaded maps. This is where the distances are calculated, by having a Walker object,walk recursively across the map to find the shortest distance between two rooms. Once is finds a path, it stores it length, then winds up the stack to find the next available path, and test if it is the shortest. The walker cannot detect beforehand if it will enter an infinite loop, so the user will be told to avoid possible loops in a map file. The shortest distance is then stored in both classroom objects, to be stored in the database after all the parsing and walking is done.

Due to the use of abstractions and interfaces, the majority of the system does not know it is using MariaDB as its data store. This allows for another developer to change the data store with no impact to system critical code. To enable the system to use a new data store, one must implement all interfaces in the me.timetabler.data.daos package, add a new case to the DaoManager’s getManager method’s switch statement and change the entry ‘*type*’ in the ‘*data\_type*’ map in the configuration file to your new data store. The rest of the system will continue to work as normal.

Algorithms

Map Walker

private void move(Coordinate start, Coordinate dest, Coordinate last, int distance) {

distance++;

CellType destination = schoolMap.getCell(dest);

CellType north, south, east, west;

north = schoolMap.getCell(start.x, start.y - 1);

south = schoolMap.getCell(start.x, start.y + 1);

east = schoolMap.getCell(start.x + 1, start.y);

west = schoolMap.getCell(start.x - 1, start.y);

if (north.equals(destination) || south.equals(destination) || east.equals(destination) || west.equals(destination)) {

if (finalDistance > distance) {

finalDistance = distance;

}

return;

}

if (!last.equals(new Coordinate(start.x, start.y - 1)) && north.isTraversable()) {

move(new Coordinate(start.x, start.y - 1), dest, start, distance);

}

if (!last.equals(new Coordinate(start.x, start.y + 1)) && south.isTraversable()) {

move(new Coordinate(start.x, start.y + 1), dest, start, distance);

}

if (!last.equals(new Coordinate(start.x + 1, start.y)) && east.isTraversable()) {

move(new Coordinate(start.x + 1, start.y), dest, start, distance);

}

if (!last.equals(new Coordinate(start.x - 1, start.y)) && west.isTraversable()) {

move(new Coordinate(start.x - 1, start.y), dest, start, distance);

}

}

**User Interface**