Programming Distributed Components COMP1690

Final Report

Joe Schofield
000776975

Due Date: 27/03/2017

Word Count: 2,325

GitHub: https://github.com/JoeScho/SafeHome

Contents

1. Introduction

This report discusses the design, evolution, and evaluation of the intruder alert system, ‘Safe Home’. The system is composed of three applications, the web forms application, the SOAP API, and the website.

Section 1 contains all the design documentation of the system, including the database ERD and UML diagrams.

Section 2 contains screenshots of the finished product, demonstrating each feature that has been implemented.

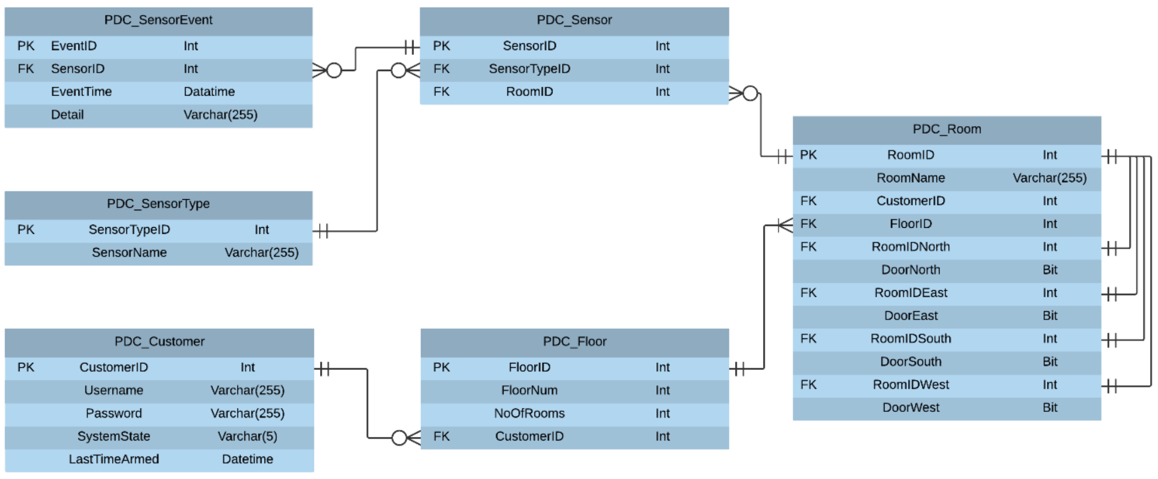
Section 3 is a critical evaluation of the evolution of the applications, including any issues during development, an evaluation of the finished product, and how the implemented system could be improved.

Section 4 is an explanation of how the system checks that layouts are physically feasible (i.e. that no two rooms can reside in the same location).

2. Design Documentation

a. ERD

Figure 1 shows the Entity Relationship Diagram (ERD) of the Safe Home database. It denotes the tables, column names, keys (i.e. Primary, Foreign), data types, and relationships.



PDC_SensorEvent		
PK	EventID	Int
FK	SensorID	Int
	EventTime	Datatime
	Detail	Varchar(255)

PDC_Sensor		
PK	SensorID	Int
FK	SensorTypeID	Int
FK	RoomID	Int

PDC_SensorType		
PK	SensorTypeID	Int
	SensorName	Varchar(255)

PDC_Room		
PK	RoomID	Int
	RoomName	Varchar(255)
FK	CustomerID	Int
FK	FloorID	Int
FK	RoomIDNorth	Int
	DoorNorth	Bit
FK	RoomIDEast	Int
	DoorEast	Bit
FK	RoomIDSouth	Int
	DoorSouth	Bit
FK	RoomIDWest	Int
	DoorWest	Bit

PDC_Customer		
PK	CustomerID	Int
	Username	Varchar(255)
	Password	Varchar(255)
	SystemState	Varchar(5)
	LastTimeArmed	Datetime

PDC_Floor		
PK	FloorID	Int
	FloorNum	Int
	NoOfRooms	Int
FK	CustomerID	Int

Figure 1 – ERD

b. Class Diagram

i. Windows Forms Application

Figure 2 displays the Class diagram for the *Safe Home* Windows forms application (including the Sensor emulator and Floor visualisation pages). The diagram shows the classes, relationships, variables, and methods.

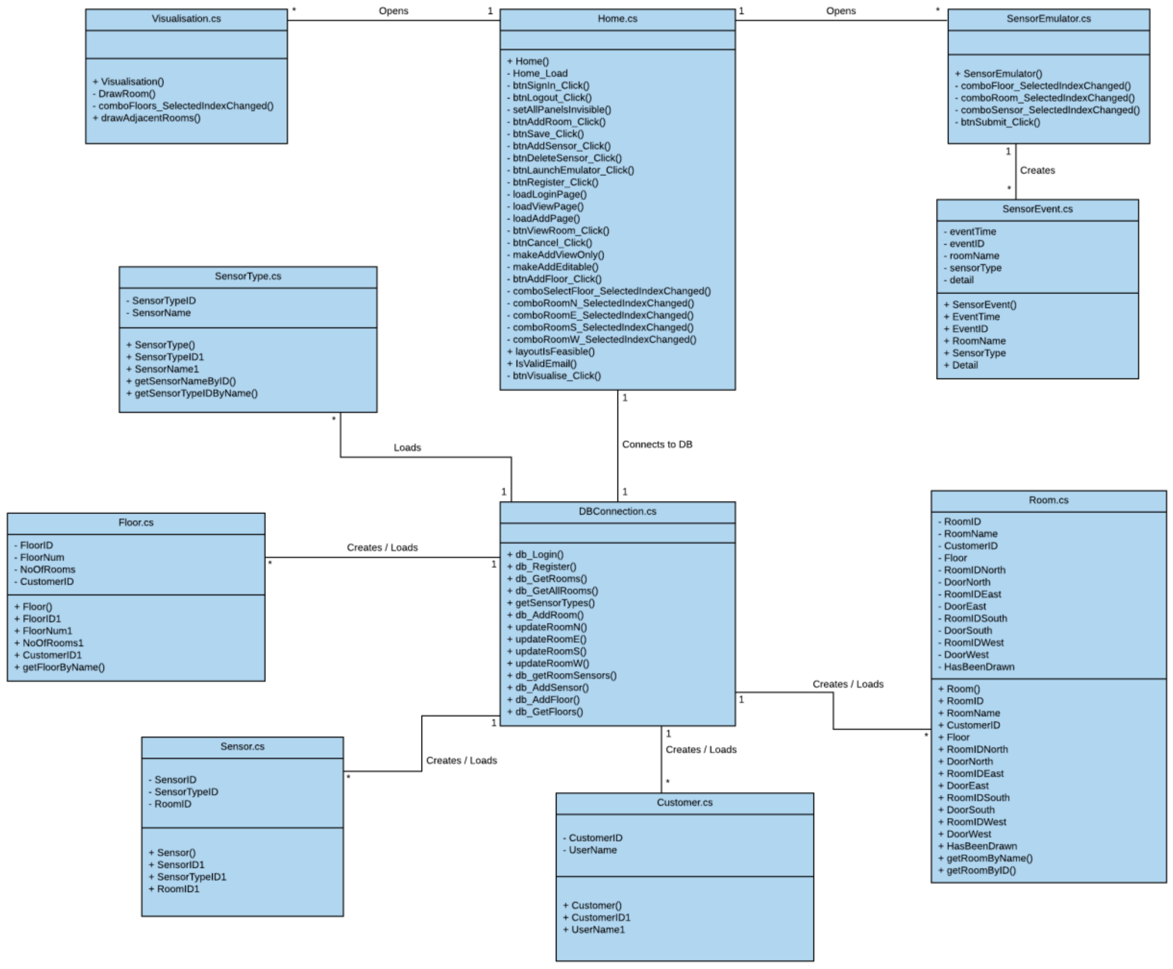


Figure 2 – Windows Forms Class Diagram

ii. SOAP API

Figure 3 displays the classes, relationships, attributes, and methods used within the SOAP API. There are no visual classes here as the API does not have a visual aspect.

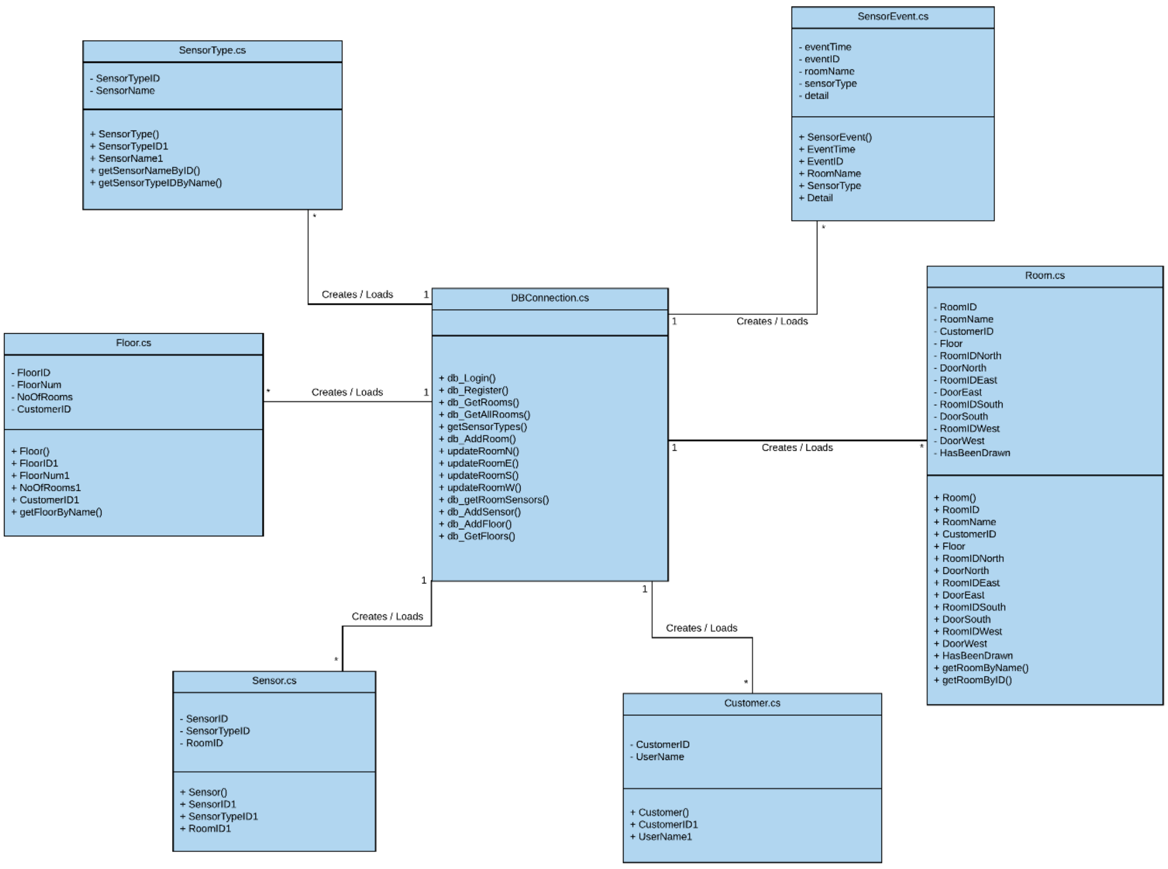


Figure 3 – SOAP API Class Diagram

iii. Web Forms Application

Figure 4 displays the only two classes used by the Web Forms application. There are no non-visual classes here as the application holds a web reference to the SOAP API in which instances of the classes can be created.

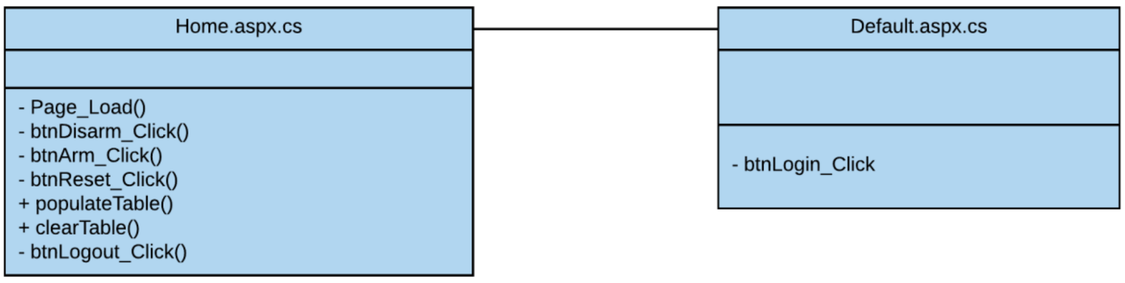


Figure 4 – Web Forms Class Diagram

c. Use Case

Figure 5 demonstrates all actors that will use the system, what they will use the system for, and further steps involved in completing these use cases (where appropriate).

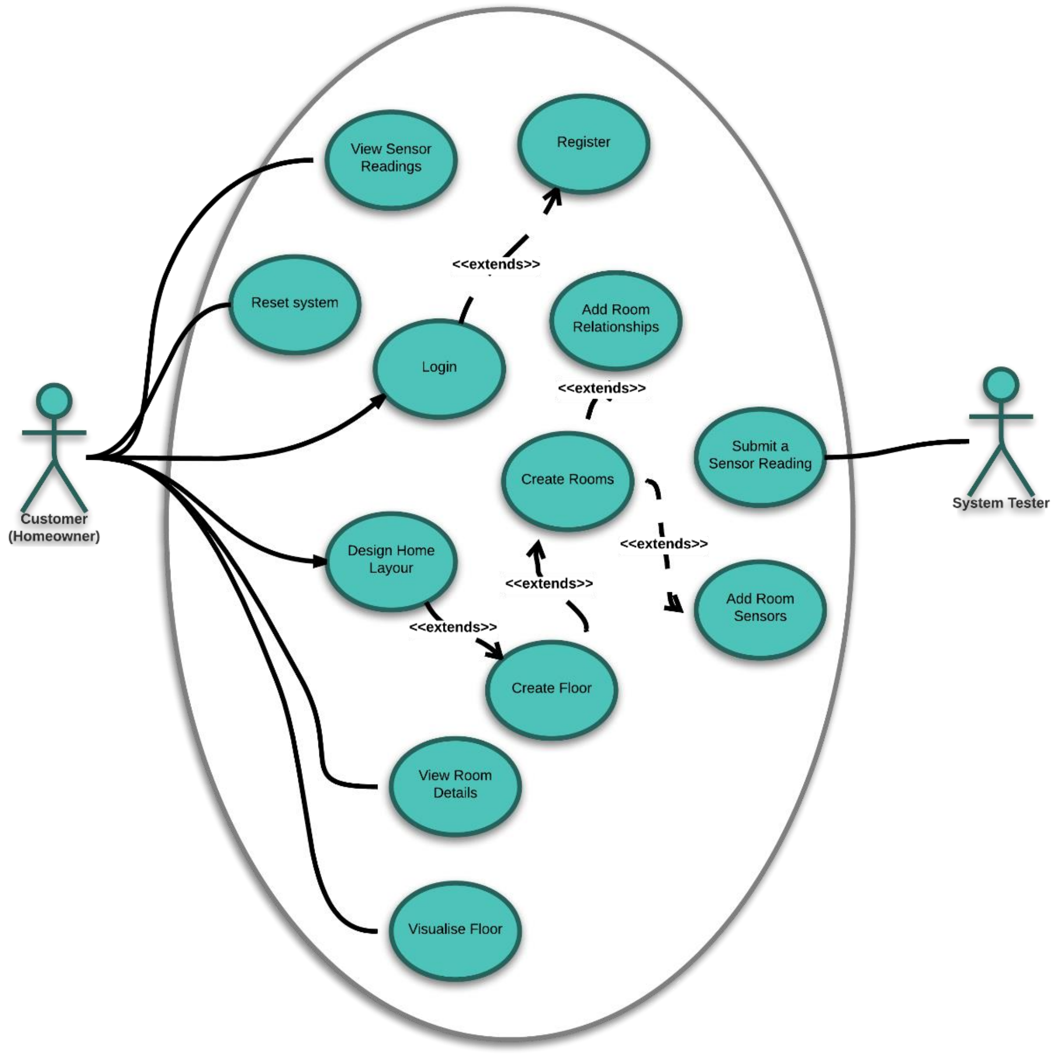


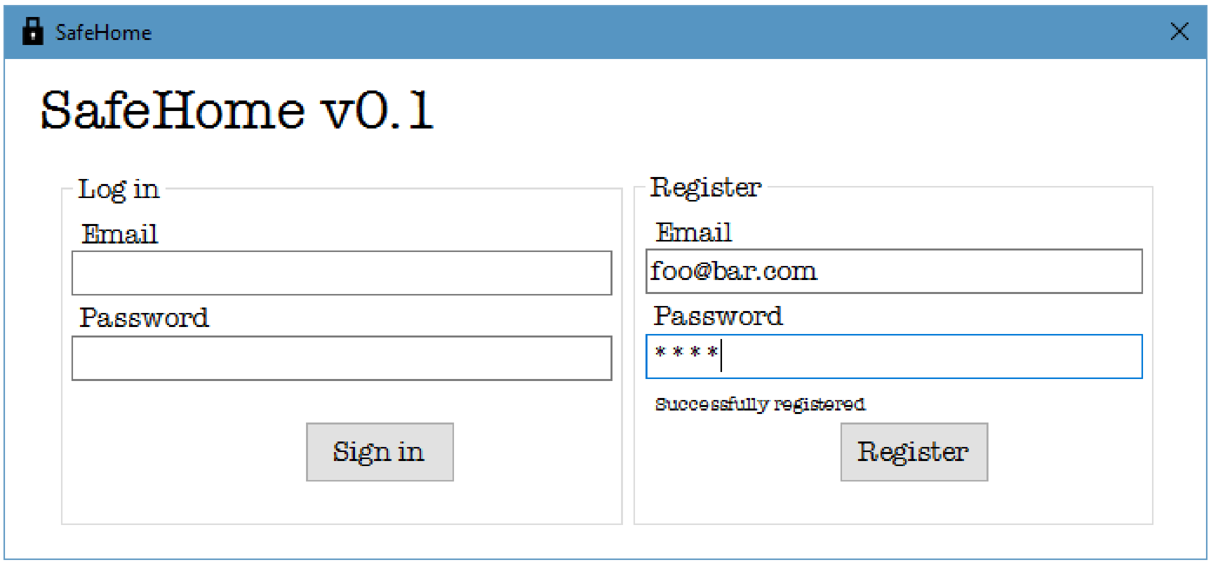
Figure 5 – Use Case

3. Screenshots of Features

a. Windows Forms Application

i. Register / Login Page

Figure 6 shows the SafeHome register and login page. The user simply enters their email address and desired password to register (a message is displayed upon successful registration). Subsequently entering these details into the login form allows the user to sign into their newly created account.



Email: foo@bar.com
Password: ****
Successfully registered

Figure 6 - Register / Login Page

ii. Home Page

Figure 7 shows what the user is displayed upon first login. The ‘Rooms’ ListBox is empty, and the ‘Floor No.’ cannot be selected as they have not been created yet. Likewise, the user cannot view a room, nor add one until a floor is created. The ‘Visualise’ and ‘Launch Emulator’ buttons do function but will have no data to work with. The only function the user can access at this time is the ‘Add a floor’ section. Once a floor has been added, the user can add rooms to that floor.

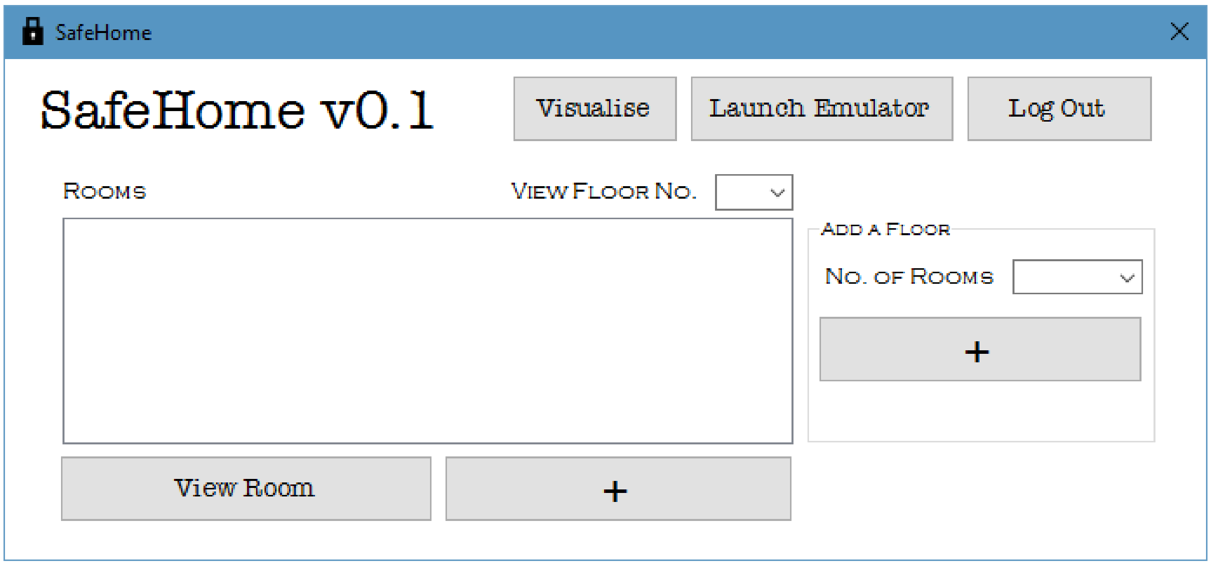
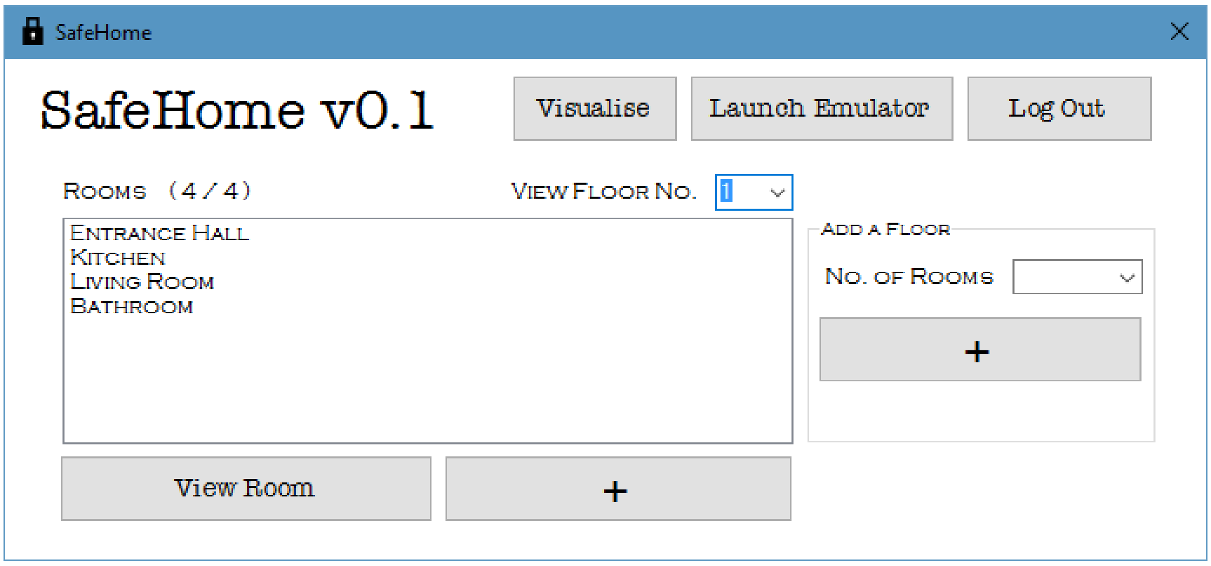


Figure 7 - Empty Home Page

Figure 8 is an example of what the user’s home page could look like after several rooms and floors have been created. The user can select a room from the ListBox to view its details.

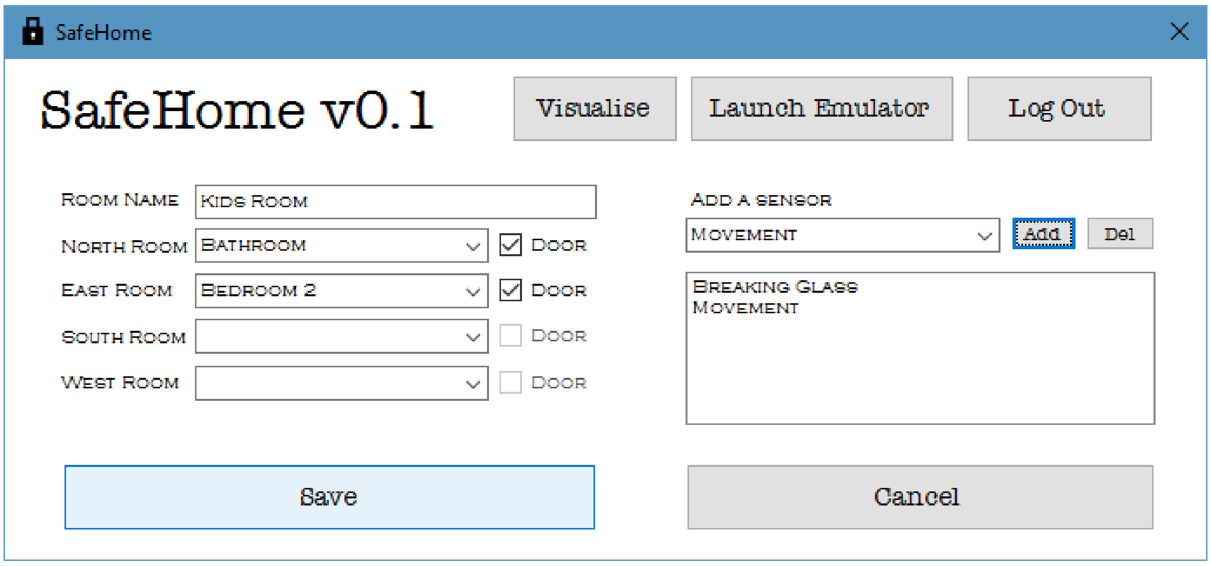


Entrance Hall
Kitchen
Living Room
Bathroom

Figure 8 - Example Home Page

iii. Add Room Page

Figure 9 is an example of how the user can add a room. They can enter a room name, any adjacent rooms, doorways, and sensors. The system ensures that the layout is physically possible so that visualisations can be produced correctly.



Room Name	Kids Room	
North Room	Bathroom	✓ Door
East Room	Bedroom 2	✓ Door
South Room		Door
West Room		Door

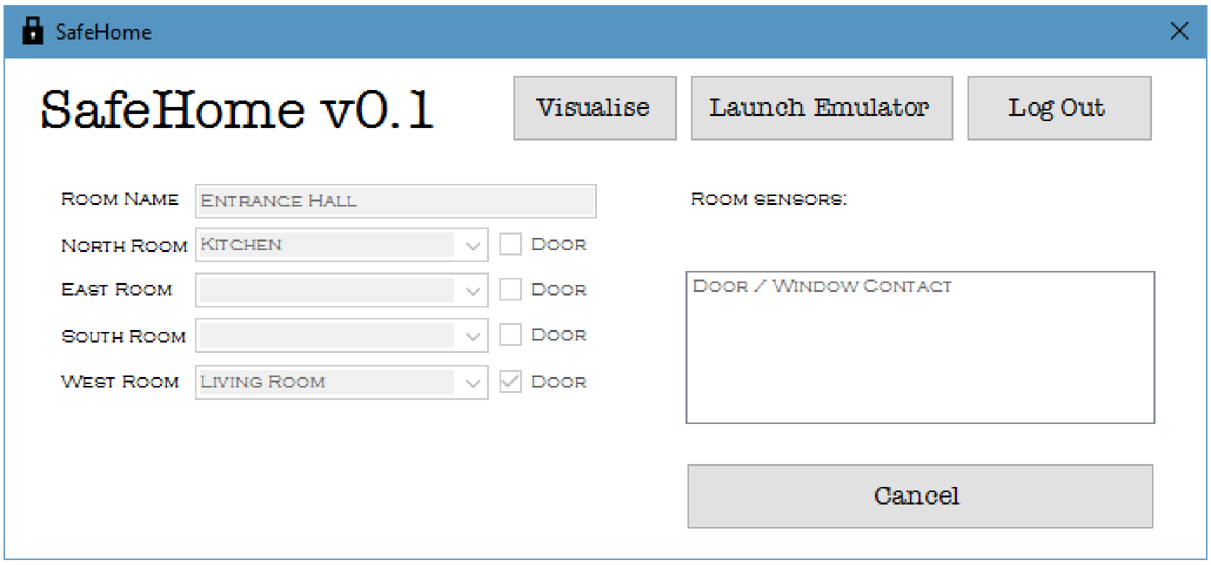
Add a sensor: Movement

Breaking Glass
Movement

Figure 9 - Adding a Room

iv. View Room Page

Figure 10 shows the 'View Room' page. Once the user has created a room they may view the room's details. They are shown the room's name, adjacent rooms, and sensors.



Room Name	Entrance Hall	
North Room	Kitchen	Door
East Room		Door
South Room		Door
West Room	Living Room	✓ Door

Room sensors:

Door / Window Contact

Figure 10 - View Room

v. Visualisation

Figure 11 shows the visualisation form within the application. Once the layout has been created the user can select to visualise the floor, this is useful to ensure the house has been created properly. The visualisation tool displays the layout of a floor, the room names, any sensors which are within the room and if the room has a doorway to other rooms.

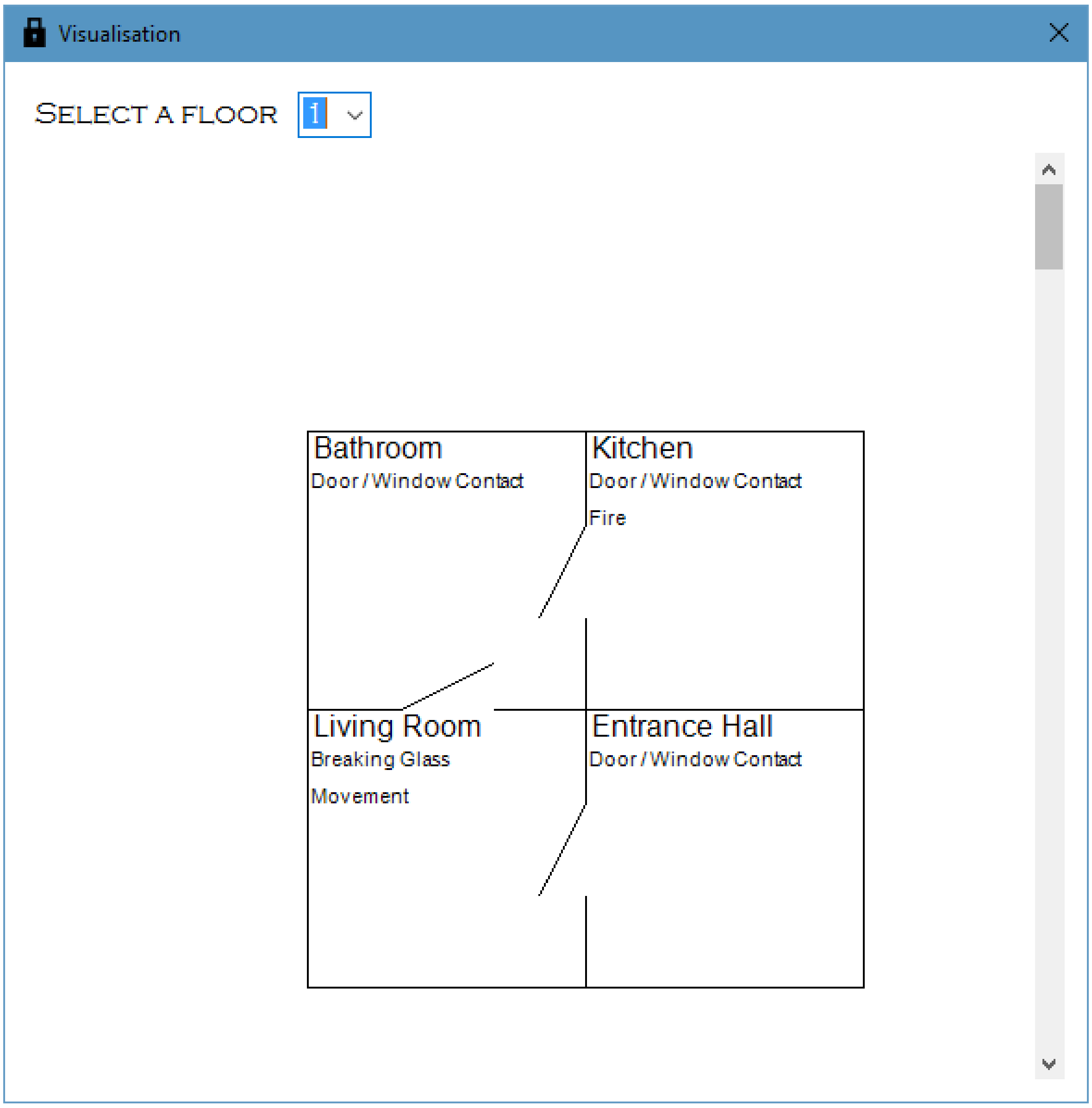
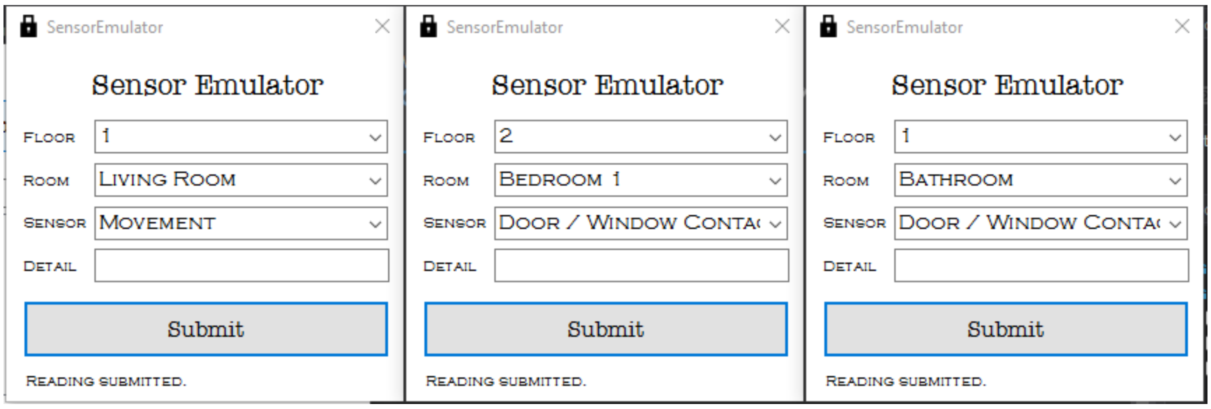


Figure 11 - Visualisation

vi. Sensor Emulator

Figure 12 displays an example of a user submitting a data reading. The user must first select a floor, then which room on the floor, then the sensor within the room, and finally add any details which are appropriate (such as the temperature for a fire sensor). This then queries the API and, upon a successful request, displays a success message. After a reading has been submitted the system will then be in 'Alert' mode, until the user resets the system. It is possible to have multiple instances of the sensor emulator running at any one time.


SensorEmulator

Sensor Emulator

Floor	1
Room	Living Room
Sensor	Movement
Detail	

Submit

Reading submitted.

SensorEmulator

Sensor Emulator

Floor	2
Room	Bedroom 1
Sensor	Door / Window Conta
Detail	

Submit

Reading submitted.

SensorEmulator

Sensor Emulator

Floor	1
Room	Bathroom
Sensor	Door / Window Conta
Detail	

Submit

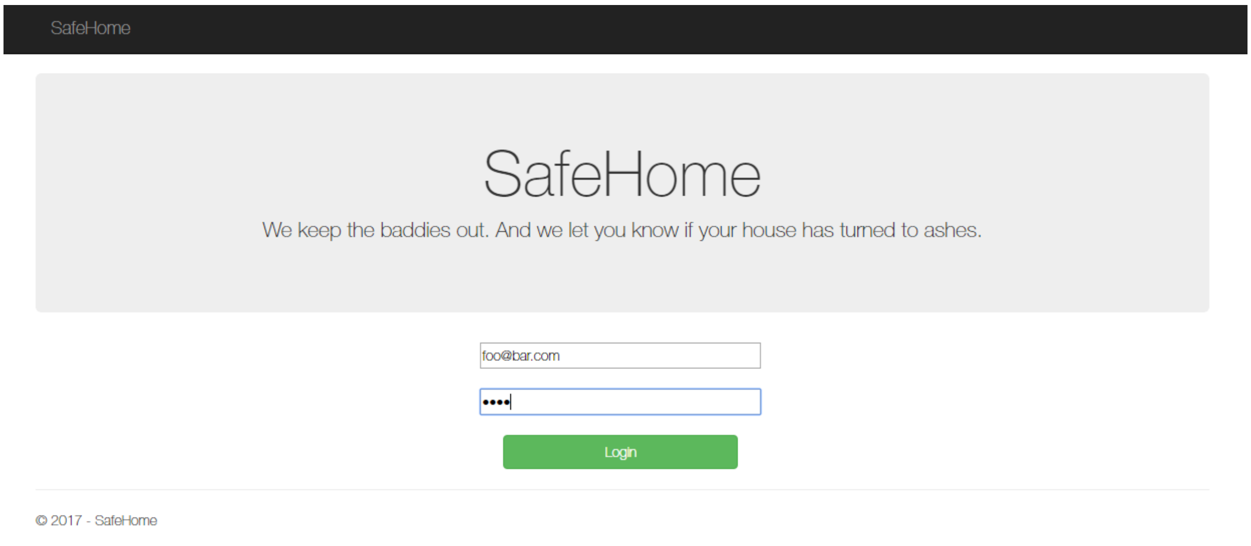
Reading submitted.

Figure 12 - Sensor Emulators

b. Web Forms Application

i. Login Page

The website login page has a very simple user interface, simply asking for the user's email address and password. Once the user has logged in with the correct details they are automatically transferred to the home page.


SafeHome

SafeHome

We keep the baddies out. And we let you know if your house has turned to ashes.

foo@bar.com

••••

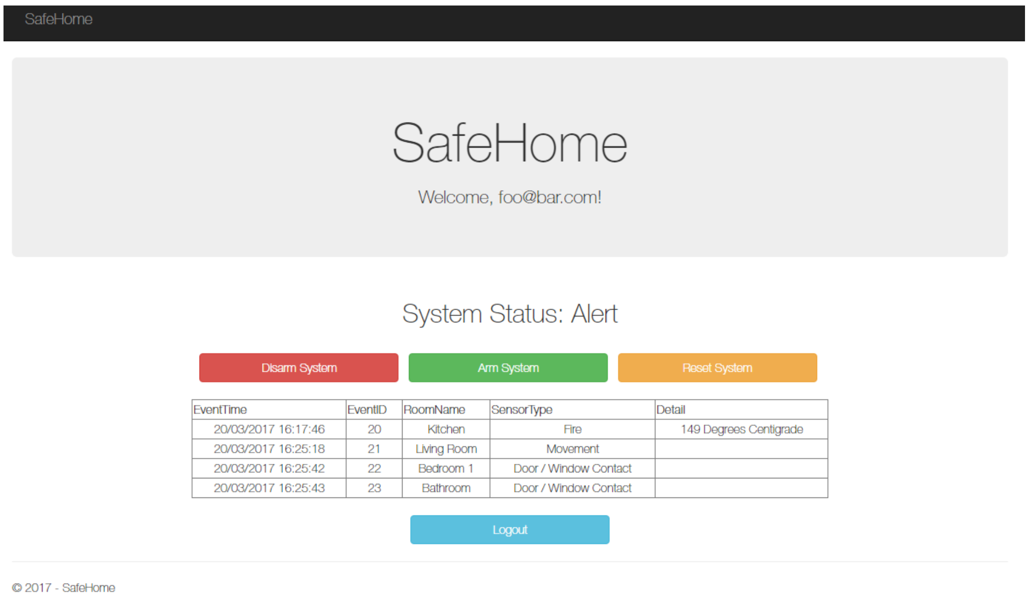
Login

© 2017 - SafeHome

Figure 13 - Login Page

ii. Home Page

Figure 14 shows the home page of the website. The top of the page shows a large jumbotron div containing the SafeHome name and a welcome message for the user. Beneath this the system status is shown. This is currently ‘Alert’ due to the sensor readings previously submitted. There are 3 buttons below the status, ‘Disarm System’, ‘Arm System’, and ‘Reset System’, which query the SOAP API. Disarm system disables the ability to submit a sensor reading from the emulator, Arm system sets the system to armed mode, in which readings can be submitted, and finally reset does a combination of disarm and arm to clear out all events displayed below.



EventTime	EventID	RoomName	SensorType	Detail
20/03/2017 16:17:46	20	Kitchen	Fire	149 Degrees Centigrade
20/03/2017 16:25:18	21	Living Room	Movement	
20/03/2017 16:25:42	22	Bedroom 1	Door / Window Contact	
20/03/2017 16:25:43	23	Bathroom	Door / Window Contact	

Figure 14 - Home Page on Alert

If the system has been reset, the user is no longer displayed any events (although these are still stored in the database for use as evidence if required), as demonstrated in Figure 15.

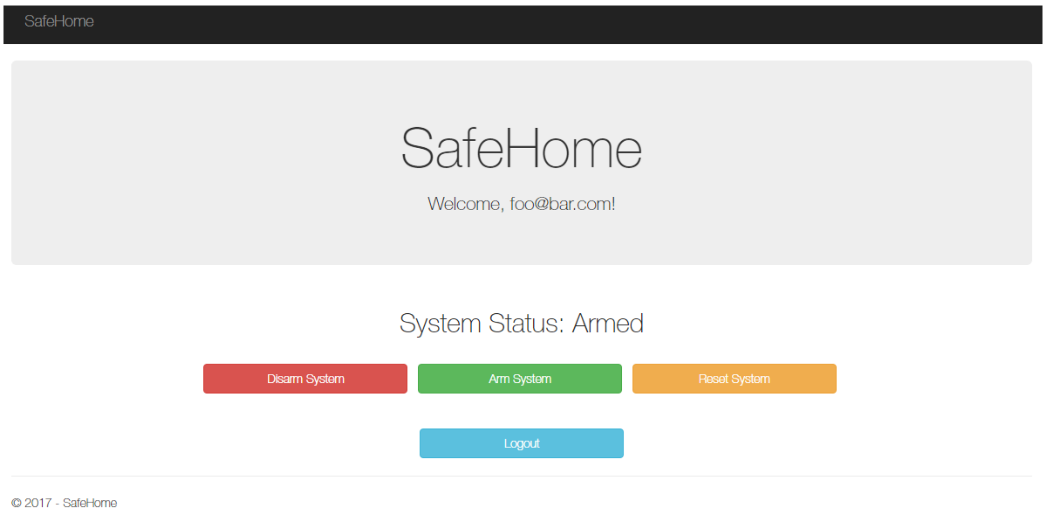


Figure 15 - Armed System

4. Evaluation

The applications created for this project meet all criteria from the coursework specification, handle all types of input by the user without crashing, and contain clear, well commented code. However, the implementation could be improved to make the software more efficient, secure, and extensible.

At present, the SOAP API does not maintain state across multiple calls, therefore it does not consistently check that the user is logged in. In future developments, I would like to alter this to maintain state and use API keys for security.

Another element I would like to implement to improve the system is to use the Polymorphism pattern. To do this I would create a class for each type of sensor and have this inherit from the base Sensor class (which I would convert to an interface). This would allow for more specific features from each sensor (i.e. if a fire sensor is selected, a temperature reading could be given, and if the earthquake sensor is selected, a Richter scale reading could be given). To further improve this, I could implement the factory pattern, as this would ensure flexibility for introducing new types of sensors.

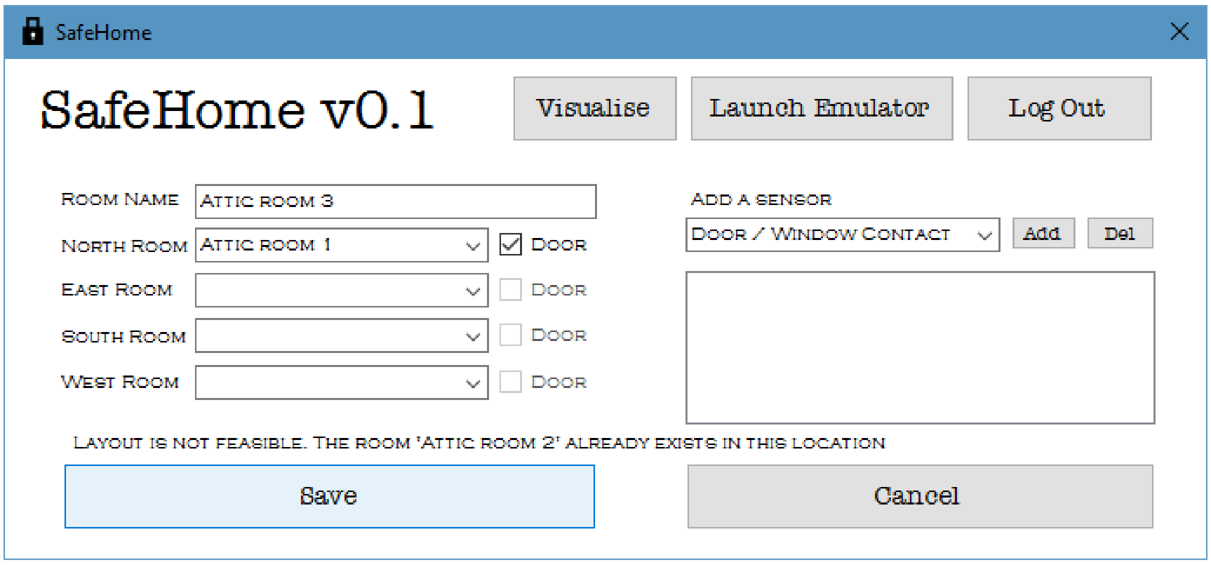
Furthermore, there are other design patterns which could be implemented, or improved, within the system. While most of the classes and methods are cohesive, some are quite tightly coupled. This produces inflexibility in the system for future developments. I would also like to have included the Controller pattern as, at present, the UI code behind class (Home.cs) contains a large proportion of the application logic. To incorporate the Controller pattern, I would create a class which is called by the UI class and which delegates the responsibility to specific classes.

Finally, I would like to improve how the application connects to, and uses, the database. At present the connection string is a hard-coded value which is stored in the “Settings.Settings” file. I would like to adapt the solution to read from a config.xml file, as this would create a simple process for changing any details. A further change which could be implemented to improve the database connection is the use of an Entity Framework Database Model, as this generates the SQL and automatically sanitises all queries, so that the database is protected from SQL injection, a common form of hacking. Furthermore, it makes the code more readable, so that it is obvious to other developers what the code does.

5. Algorithms Explanation

The SafeHome application contains a feature to ensure that floor layouts are physically possible. This ensures that rooms do not overlap when visualising the house. Figure 16 demonstrates an example error message the user might encounter. In this example, there are three rooms; ‘Attic Room 1’, ‘Attic Room 2’, and ‘Attic Room 3’. The user has created ‘Attic Room 1’ and ‘Attic Room 2’, setting ‘Attic Room 1’ north of ‘Attic Room 2’. They are now attempting to create ‘Attic Room 3’ in the same location as ‘Attic Room 2’. When they click on ‘Save’ they are displayed the following error message:

“Layout is not feasible. The room ‘Attic room 2’ already exists in this location”



ROOM NAME: ATTIC ROOM 3
NORTH ROOM: ATTIC ROOM 1 [✓ DOOR]
EAST ROOM: DOOR
SOUTH ROOM: DOOR
WEST ROOM: DOOR
ADD A SENSOR: DOOR / WINDOW CONTACT

LAYOUT IS NOT FEASIBLE. THE ROOM 'ATTIC ROOM 2' ALREADY EXISTS IN THIS LOCATION

Figure 16 - Layout error

The algorithm works by checking each room currently on that floor (in this case ‘Attic Room 1’ and ‘Attic Room 2’) to see if any adjacent rooms selected (‘Attic Room 1’) are already adjacent in the same position (North) to another room. In the case of the previously stated example, the algorithm discovers that ‘Attic Room 2’ has already stated that ‘Attic Room 1’ is directly North of it, therefore ‘Attic Room 3’ cannot be in this location.

a. Pseudo-code

Below is the pseudo-code created to design the algorithm to check that the layout of the floor is feasible.

```
checkLayoutIsPhysicallyPossible(newRoom, listOfCurrentRoomsOnFloor) {
  if (newRoom.hasANorthRoom) {
    if (listOfCurrentRoomsOnFloor.RoomHasSameNorthRoom()) {
      return false
    }
  }
  if (newRoom.hasAnEastRoom) {
    if (listOfCurrentRoomsOnFloor.RoomHasSameEastRoom()) {
      return false
    }
  }
  if (newRoom.hasASouthRoom) {
    if (listOfCurrentRoomsOnFloor.RoomHasSameSouthRoom()) {
      return false
    }
  }
  if (newRoom.hasAWestRoom) {
    if (listOfCurrentRoomsOnFloor.RoomHasSameWestRoom()) {
      return false
    }
  }
  // If it reaches here then layout is possible
  return true
}
```

b. Example Using Diagrams

This section contains an example and diagrams to explain the process of the algorithm. The example will follow the algorithm as:

a) the user successfully creates a room
b) the user fails to create a room due to an invalid location.

Below is a table of data regarding one floor of a house.

Room Name	RoomNorth	RoomEast	RoomSouth	RoomWest
1	-	-	2	-
2	1	3	-	-
3	-	4	-	2
4	5	-	-	3
5	-	-	4	-

This creates a floor as follows:

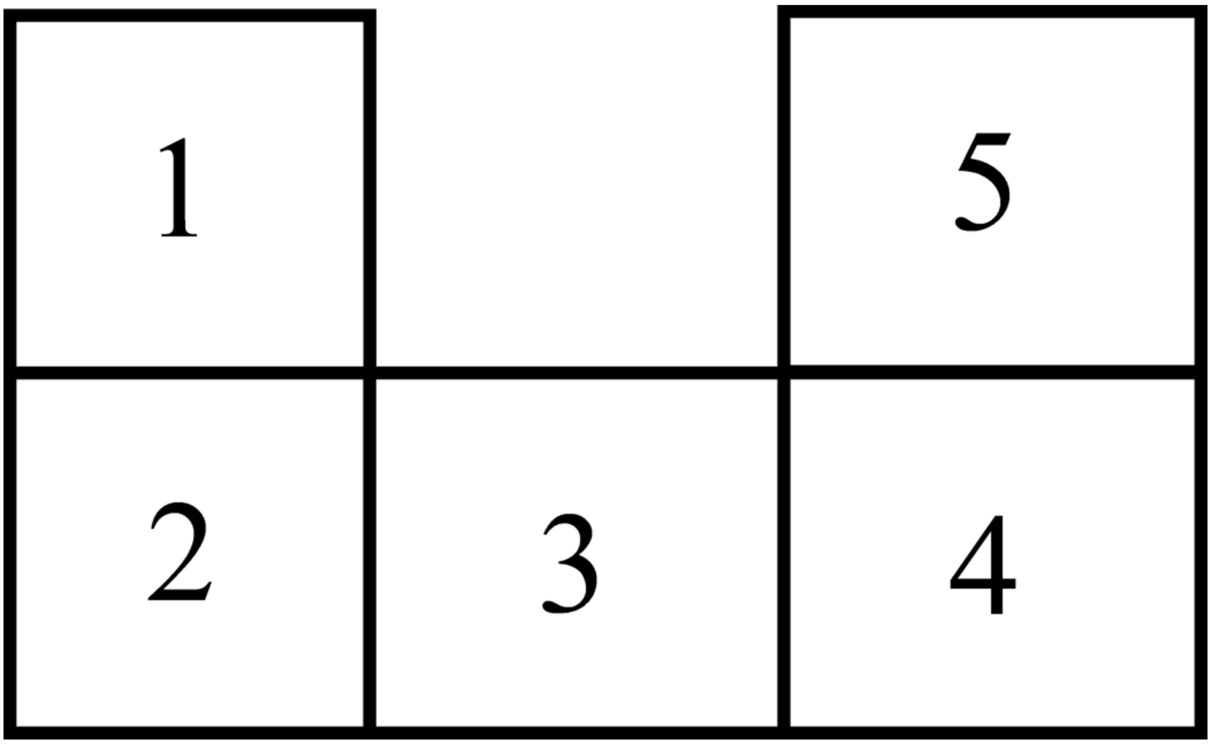


Figure 17 - Example room layout

In example a), the user successfully adds a room to the layout by inserting a room (6) between rooms 1,3 and 5, like so:

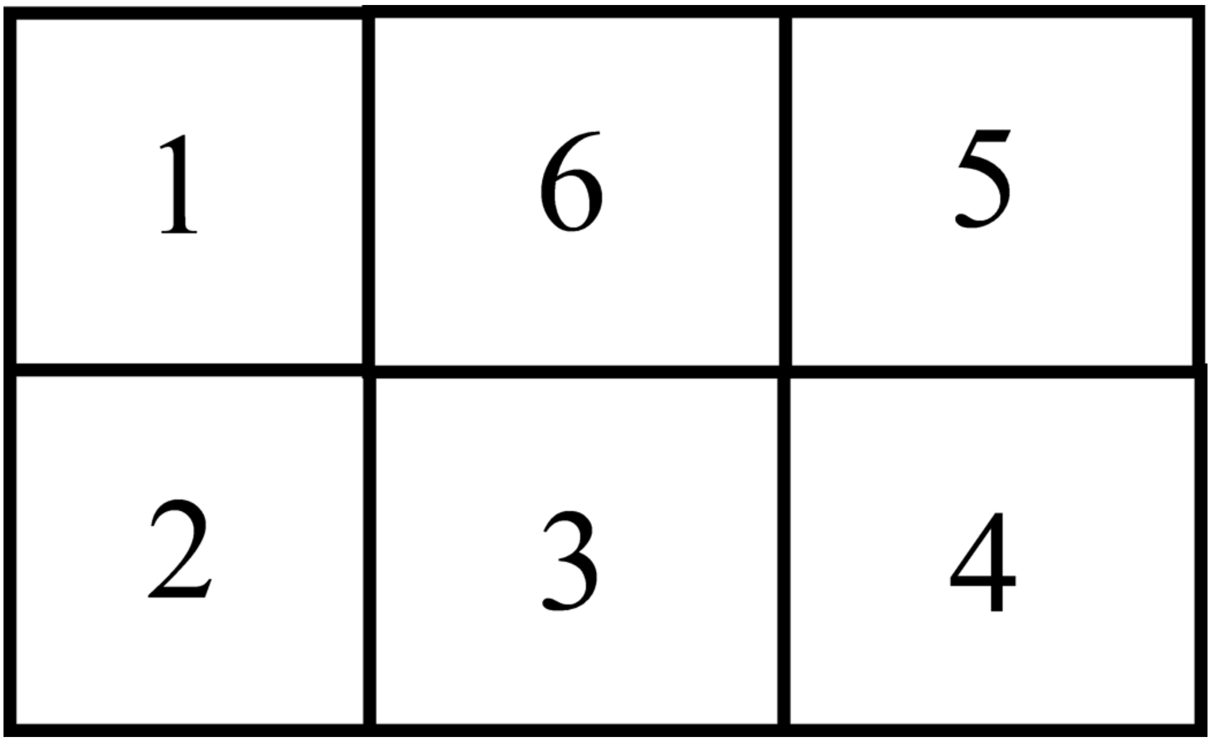


Figure 18 - Successful Room Addition

In this case the algorithm would check:

- Does the room have a room to the North of it?
    - No
- Does the room have a room to the East of it?
    - Yes. Does any other room have ‘5’ to the East of it?
        - No – therefore this is successful
- Does the room have a room to the South of it?
    - Yes. Does any other room have ‘3’ to the South of it?
        - No – therefore this is successful
- Does the room have a room to the West of it?
    - Yes. Does any other room have ‘1’ to the West of it?
        - No – therefore this is successful
- It will then reach the ‘return true’ statement at the end of the method.

This will successfully create the room.

In example b), the user attempts to add a room to the layout by inserting a room in the same location as room 5, like so:

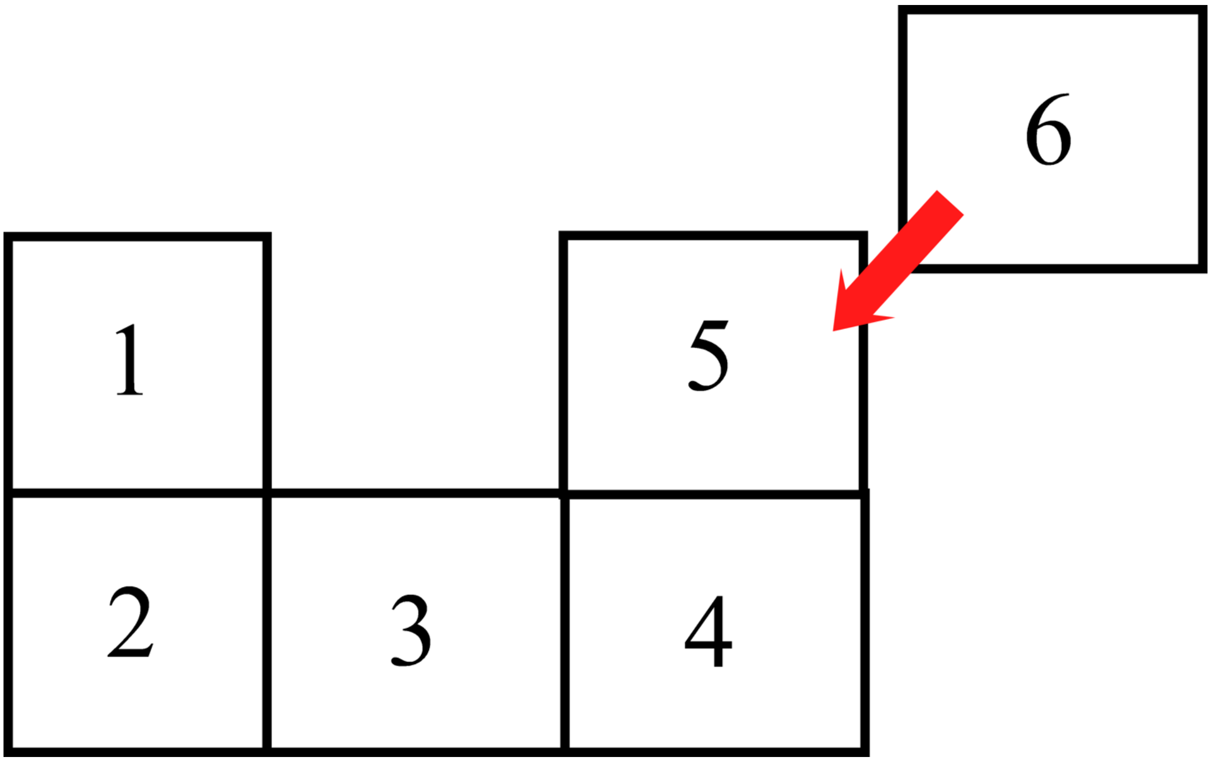


Figure 19 - Impossible Floor Layout

In this case the algorithm would check:

- Does the room have a room to the North of it?
    - No
- Does the room have a room to the East of it?
    - No
- Does the room have a room to the South of it?
    - Yes. Does any other room have '4' to the South of it?
        - Yes. It will return 'false'

The room will not be created. It will display the name of the room that is in that location to the user.

c. Improving the algorithm

This algorithm does do the job of ensuring that impossible layouts cannot be created. However, there are some cases in which impossible layouts are created.
When a user creates a relationship between two rooms, it updates the target room to add the new adjacent room. However, if the user specifies one relationship, but not another, and then creates another room and specifies the other relationship, but not the first, then the algorithm will not discover the impossible layout, and the two added rooms will co-exist in the same location. See figure 20.

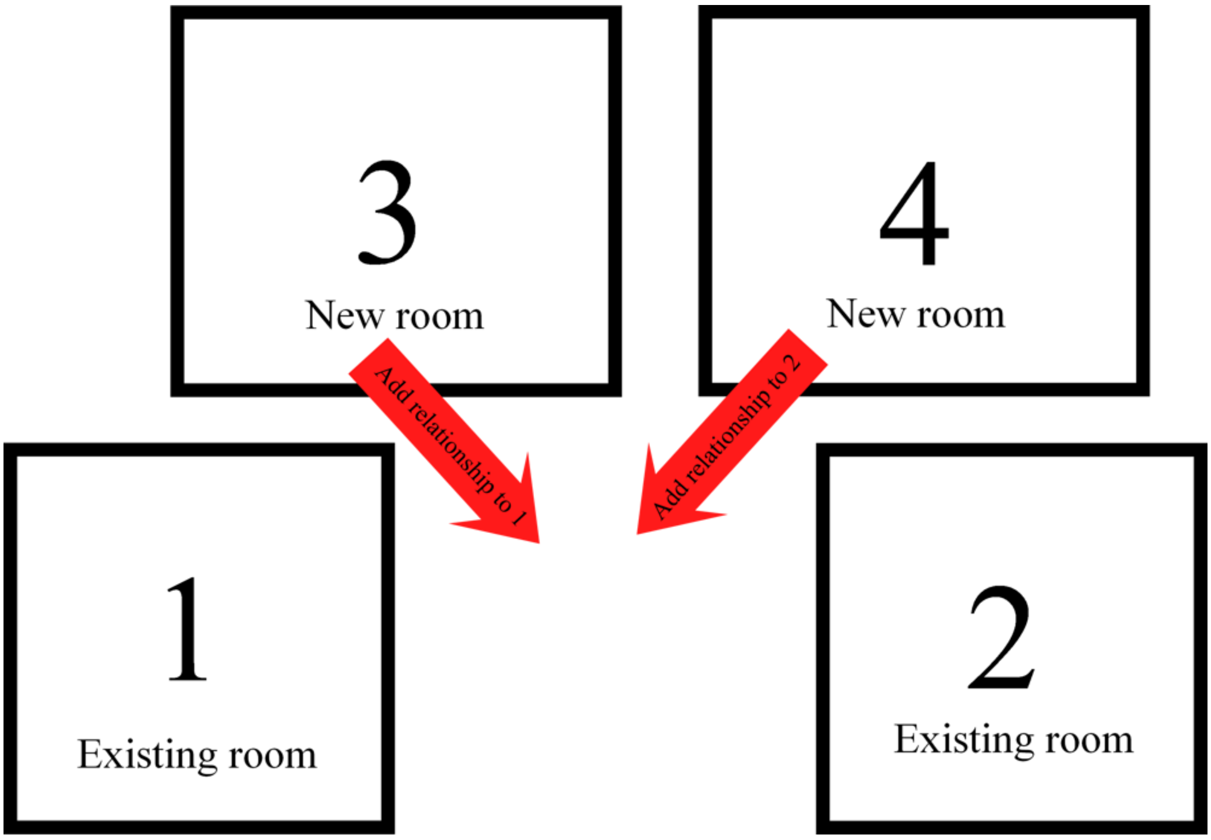


Figure 20 - Algorithm Flaw

To fix this flaw I would not need to change the algorithm, but the code to add a room. It would need to be aware that rooms 1 and 2 have a gap the size of one room between them, so that both relationships can be added automatically.