

# Academic Building Electronic Map (ABEM)

Zhao, xianrong (xzhao@ust.hk)

## 1. Background

Academic Building Electronic Map (ABEM) is a program I developed as my independent project, for the MSc (IT) at HKUST, which was under supervision of Professor David Rossiter.

ABEM is a web-based system for people to get more information about the academic building. Compared with the old Academic building map, I make some improvement on it and help people know more about HKUST. Via this map you can:

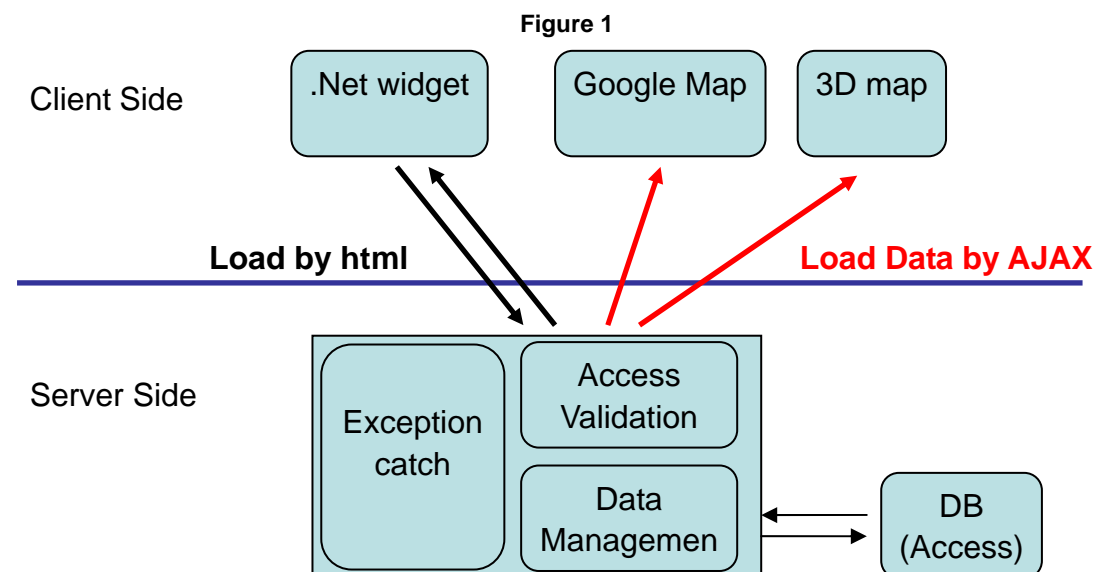
- Have better user experience
- Search the room and the nearest lift
- Look through the academic building
- Get more information of the Lift, office and theater

## 2. Structure

### 2.1 Overview

ABEM is a web-based system. Client side consists of .net widget, Google map and canvas. Especially Google map and Canvas are major of the user page. In order to provide better user experience, the user page load data from the server side by AJAX via JSON format.


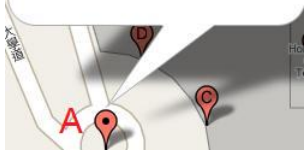


On the other hand, Server side is based on .net 3.0, which controls the data management, access validation and Exception catch by C#. I use Access as my database. Details are shown below (Figure 1):



## 2.1 Google Map

### 2.1.1 Component introduction

Google map plays an important role in the system. It shows us the outline of the academic building. It mainly has four components, there are Marker, information window, path and overlay area.

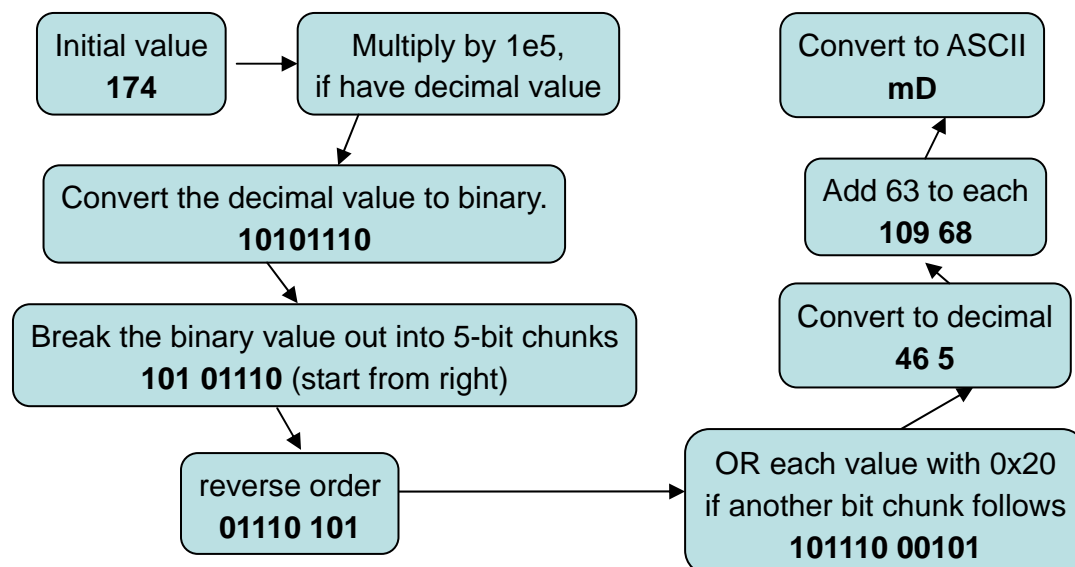
Name	Class/Function	Image	Use
A. Marker	GMarkers		Location of the lift, theater, office and so on.
B. Information window	GInfoWindow		Show information of lift, theater, office and so on.
C. Polyline	GPolyline		A series of straight segments on the map. (path)
D. Overlay area	GPolygons/ GGroundOverlay		Make the map look more realistic

### 2.1.2 Improvement for path

The polyline object within a Google map denotes a line as a series of points, making it easy to use but not necessarily compact. Long and complicated lines require a fair amount of memory, and often may take longer to draw.

As result, I use encoded polylines, which specify a series of points using a compressed format of ASCII characters. It makes our path draw much more efficiently. The algorithm is shown below:

Figure 2



By the help of path code algorithm, I build a path code generator for create path for lift. Details will be introduced in User guide section.

## 2.2 3D Map

### 2.2.1 Introduction

3D Map is a pseudo-3D scene that helps us look through the academic building. It base on canvas which is element of HTML5.

In the coming paragraph, I'll deconstruct the 3D map and go through the details of how to create pseudo-3D map. I say pseudo-3D because what we're essentially creating is a 2D map that we can make appear 3D by as long as we restrict how the player is able to view the world.

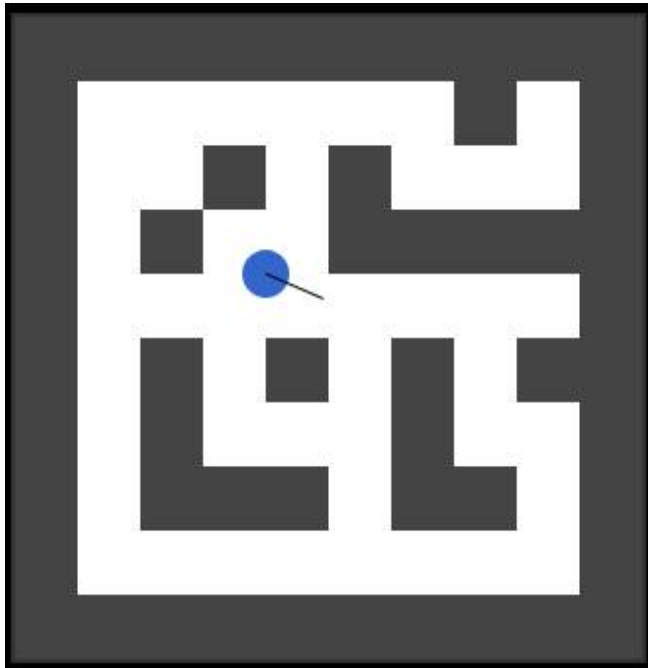
### 2.2.2 The map and Control

The first thing we need is a map format. One easy way to store this data is in an array of arrays. (Basically, 1 for wall and 0 for road) The wall type will be used later to determine which texture to render. (Figure 3)

Meanwhile, we need to bind the arrow keys to control the movement and detect the collision of wall (`map[newx][newy]==1`).

```
switch (window.event){  
    case 65:case 37: break; // left  
    case 87: case 38: break; // up  
    case 68: case 39: break; // right  
    case 83: case 40: break;// down  
}
```

Figure 3



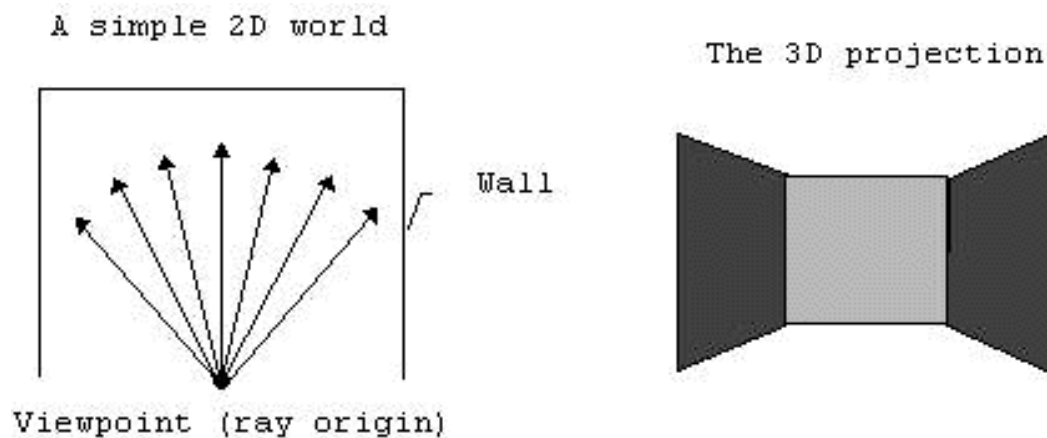
### 2.2.3 The Ray-Casting Algorithm

Now we can begin moving to 3D. Ray-Casting algorithm is the key part of it. In order to make the calculation simple, I set some default value in my system.

- field of view is 60 degrees
- the play is half of the wall height
- walls are always at 90 angles with the floor

To understand Ray-Casting, imagine rays being shot or "cast" out from the viewer in all directions within their field of view. When the ray hits a block (by intersecting one of its walls), we know which block/wall on the map should be displayed in that direction. For example, ray-casting transforms something like A into B in Figure 4 below.

Figure 4



The tricky part here is how to find the corner of the wall and we need to use this precious data to build the 3D map.

First of all, to find out all such corner, we can simply trace several rays starting from left to right. This can be done in a loop.

Then, we begin to detect each rays, steps of calculation is below: (Figure 5)

- Find coordinate of the first intersection (point A in this example)
- Increase the line PA and check the end point. If there is a wall, stop and calculate the distance. If there is no wall, extend to the next point until there is a wall.
- Calculate  $\text{newDy} = PD \cdot \sin A$  (because the previous wall is Horizontal, if the previous wall is vertical then we calculate the newDx)
- If  $\text{newDy} \neq \text{oldDy}$ , then it means that we meet an corner and we write down the data, else we increase the angle and go back to step1.

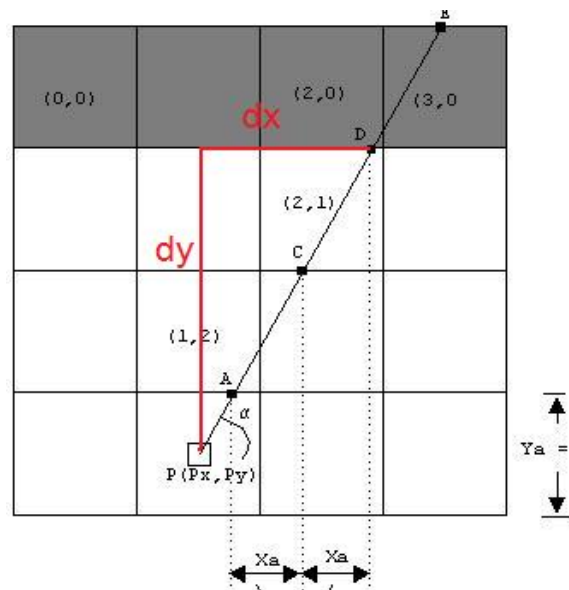
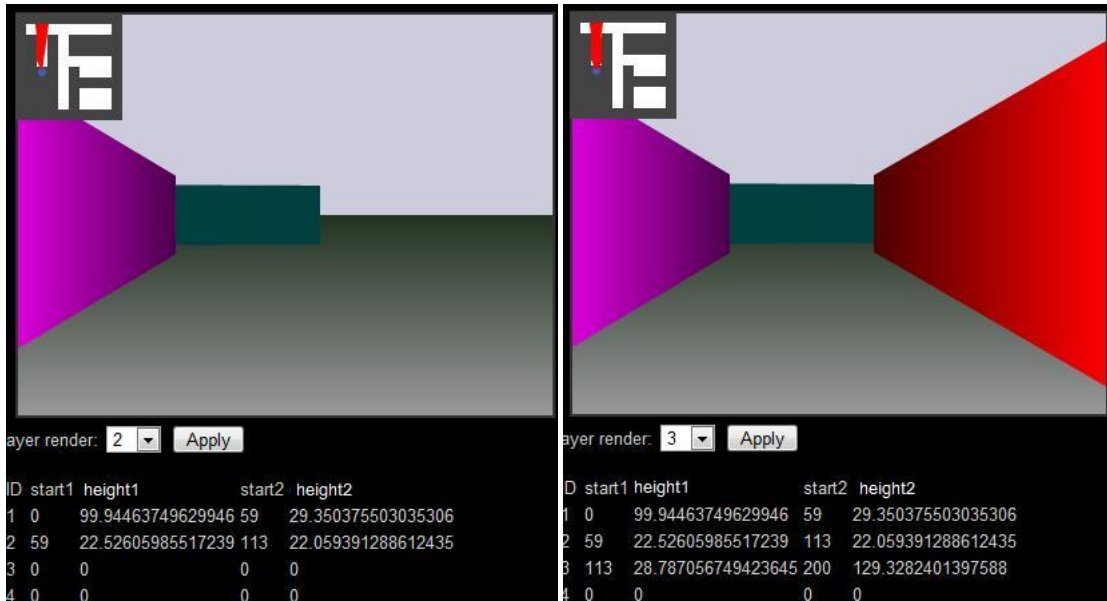


Figure 5

Finally, after we calculate all the rays, we get a series of data about the wall (start point,

start point height, end point, end point height). As result, we can build the 3D map easily by this data as Figure 6 below.

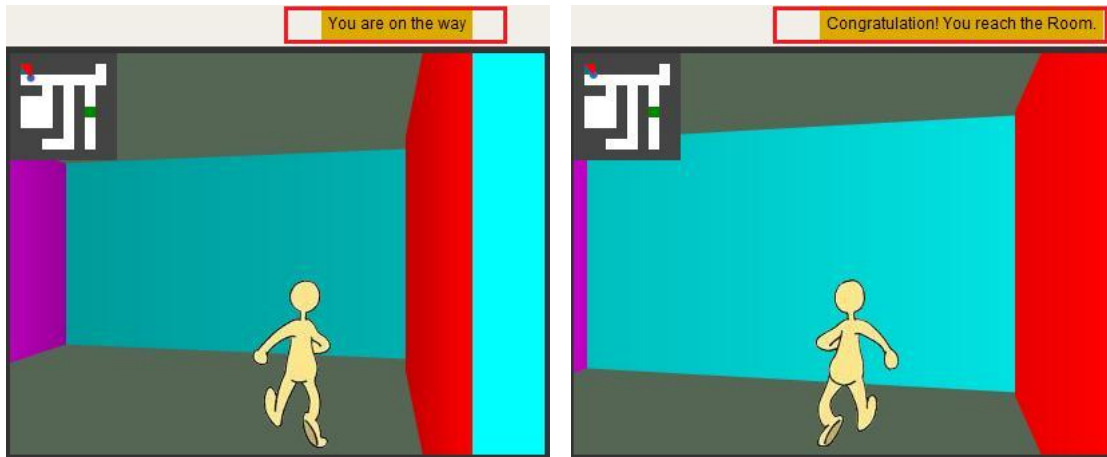
Figure 6



### 2.2.4 Add something interest

The general step of 3D map has been completed. Because the canvas is a native element of html5, it can be control by JavaScript easily. We can something interest, such as a moving person, reminder of location and so on. (Figure 7)

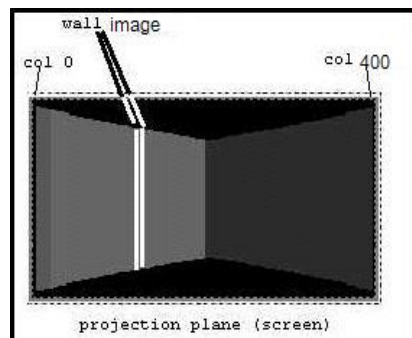
Figure 7



### 2.2.5 Texture

In 2.2.3 we use ray-casting algorithm to build the wall, but we only draw the wall by fill with gradient color. We can fill the wall with images to make it look more realistic.

The idea is similar with 2.2.3, we use ray-casting algorithm to get view field data, but we don't write down the corner only, we need to write down all rays. Assume that our view field is 400 pixels and we divide it into 200 pieces, each piece is one Image and it is

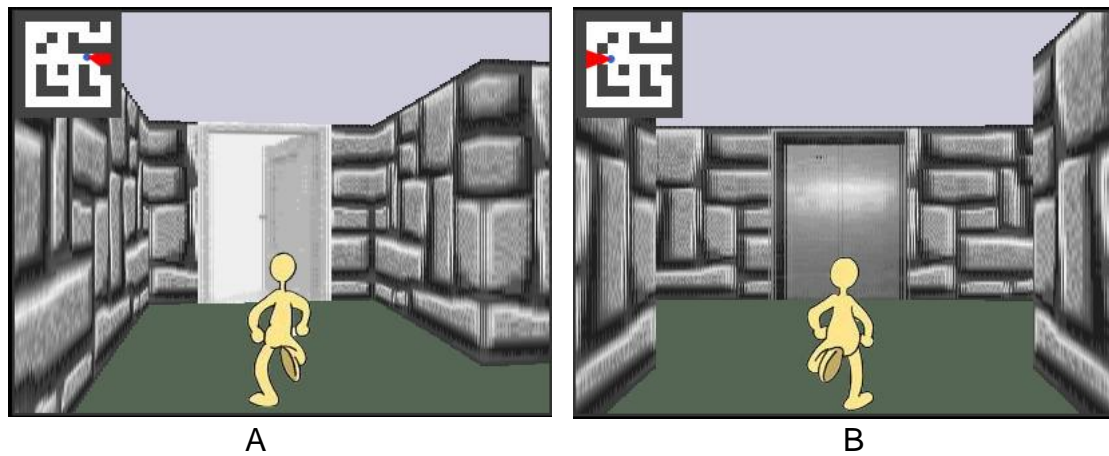


calculate from the rays by ray-casting algorithm. (Figure 8)

Figure 8

We should notice that the image must tileable in x axis. Result is below. (Figure 9) It shows the Lift texture (A) and the door texture (B).

Figure 9



## 2.3 Ajax

In order to provide better user experience, I add AJAX technology in the home page. It includes two parts:

### 1. Auto complete

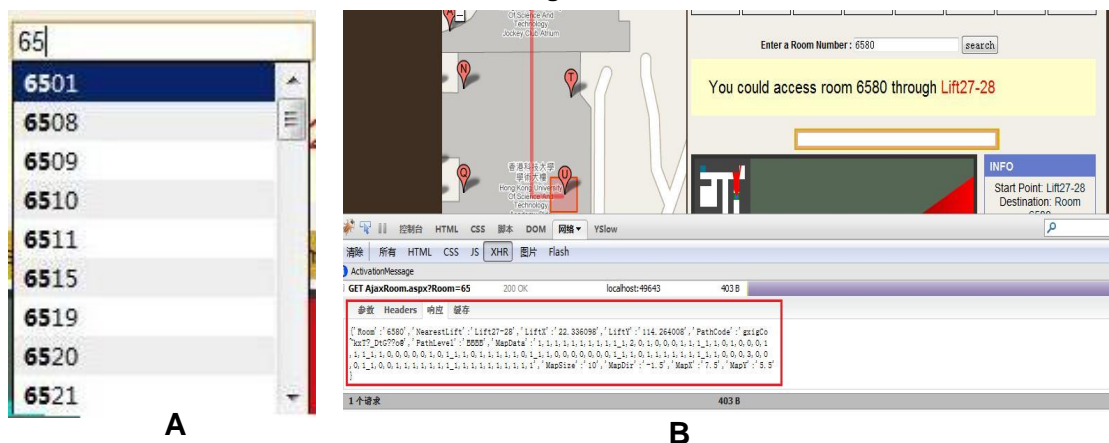
It provides the front-end logic for text-entry suggestion and completion functionality.

We can see the example in Figure 8.A.

### 2. Load room data.

By the help of Firebug, we can see the remote data from server after we press the search button. (Figure 8.B) Then we can get the room information without refresh the page.

Figure 8

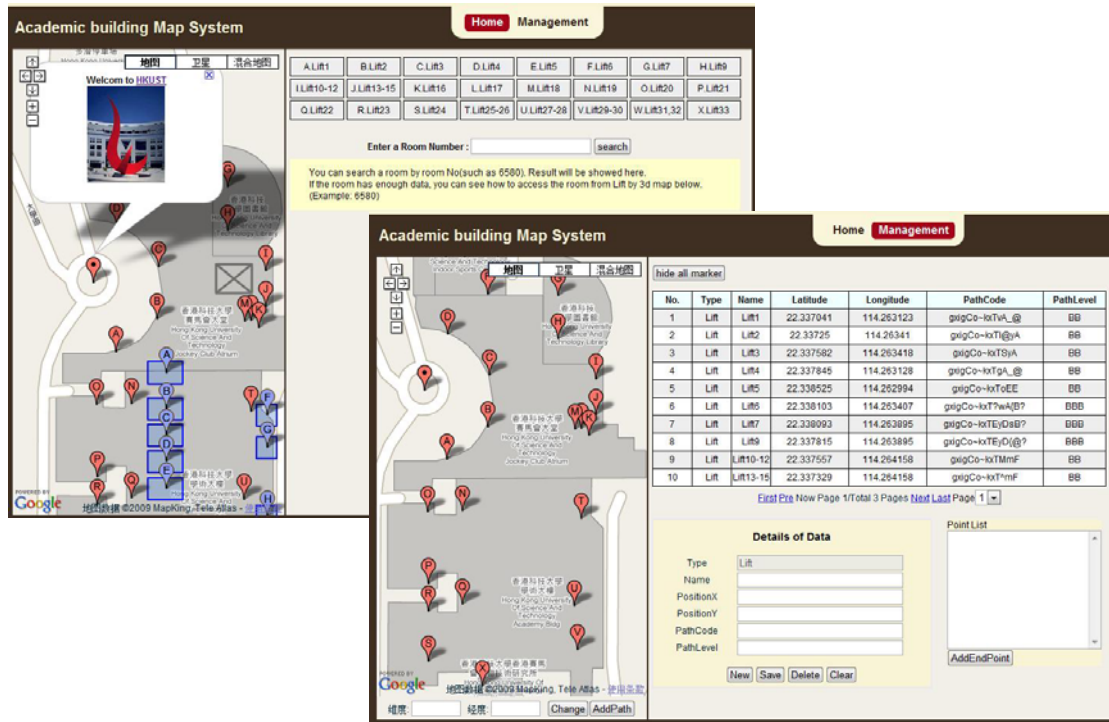


# 3 User Guide

## 3.1 Overview

The figure below shows the outline of Academic Building Electronic Map. There are two pages (the user page and the data management page) now. The structures of these two pages are similar. They include the header, Google map and the control panel.

Figure 9



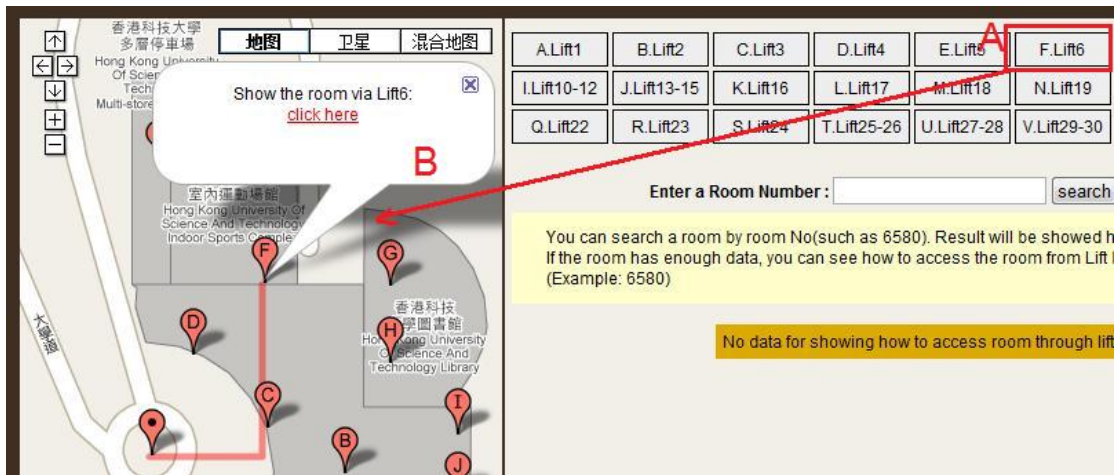
Name	illustrate
Header	It includes the title of the system and the tab page. When your mouse moves over the tab page, it kindly turns red to indicate you that you can click it and you will reach the other page.
Google map	We can get the outline of the academic building. It shows your current position, Lift/Theater position, Lift path, Lift coordinate and helps us create new item of the academic building.
Control panel	It is quite different between these two pages. I'll go through these two pages.

## 3.2 The home page

### 3.2.1 Show Lift Information

In the home page, there are several lift buttons at the top of control panel. After you click one (Figure 10.A), the details information of the lift will show of the Google map, including name, path and others. (Figure 10.B)

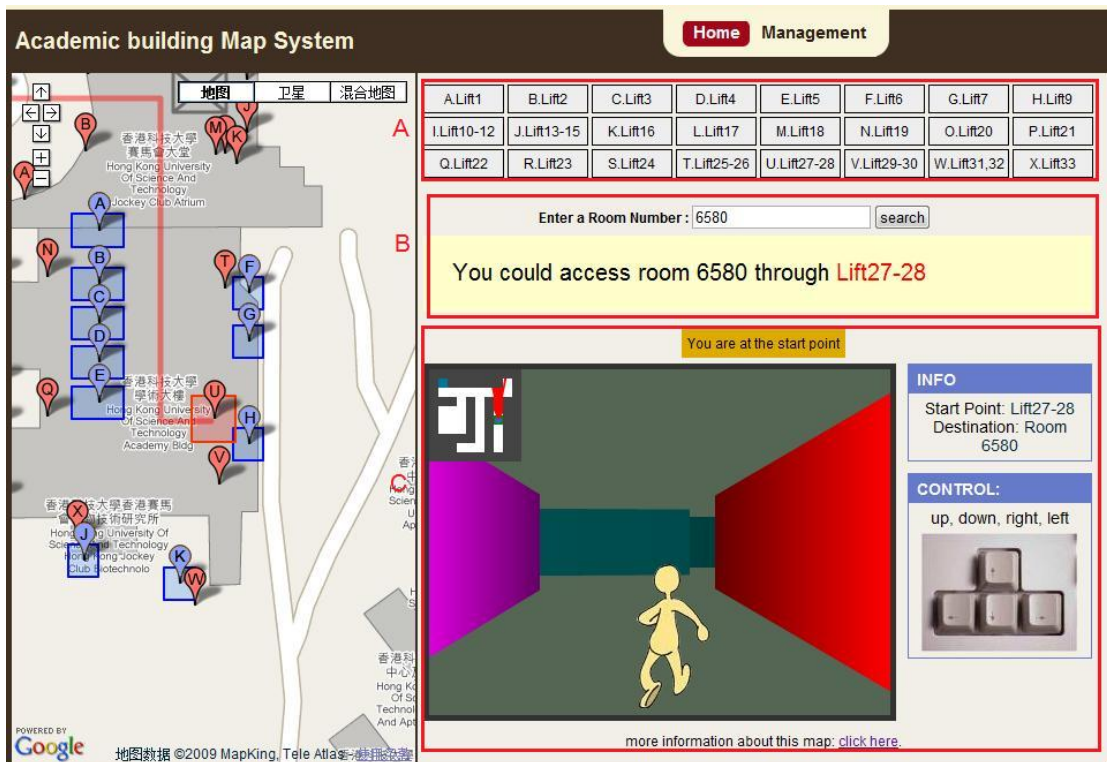
Figure10



### 3.2.2 Search room

Below the lift buttons, there is the Search panel. It provides the front-end logic for text-entry suggestion and completion functionality. (Figure 11.A) After we press the search button, the loading logo will show to remind us the data is loading. Immediately, the data come from the server, the information board shows the nearest lift and the Google map shows the path. If there is enough data, we can see the 3D map. (Figure 11.B)

Figure 11



### 3.2.3 3D map

3D map (Figure 11.C) is at the bottom of the control panel. It helps us to go to the room easily. At the right side, we can see the start point and destination. Off course, we can control the player by arrow keys and the information board shows our status.

## 3.3 The data management page



Figure 12

The screenshot shows a web application interface for lift data management. It consists of three main parts:

- Figure 12.A: Lift Data Table**

No.	Type	Name	Latitude	Longitude	PathCode	PathLevel
1	Lift	Lift1	22.337041	114.263123	gxigCo~kxTvA_@	BB
2	Lift	Lift2	22.33742	114.26372	gxigCo~kxTI@_A wA	BBB
3	Lift	Lift3	22.337582	114.263418	gxigCo~kxT\$yA	BB
4	Lift	Lift4	22.337845	114.263128	gxigCo~kxTgA_@	BB
5	Lift	Lift5	22.338525	114.262994	gxigCo~kxToEE	BB
6	Lift	Lift6	22.338103	114.263407	gxigCo~kxT?wA B?	BBB
7	Lift	Lift7	22.338093	114.263895	gxigCo~kxTEyDsB?	BBB
8	Lift	Lift8	22.337815	114.263895	gxigCo~kxTEyD @?	BBB
9	Lift	Lift10-12	22.337557	114.264158	gxigCo~kxTMmF	BB
10	Lift	Lift13-15	22.337329	114.264158	gxigCo~kxT^mF	BB
- Figure 12.B: Details of Data Editor**

This form allows editing lift data. The current data shown is:

  - Type: Lift
  - Name: Lift6
  - PositionX: 22.338103
  - PositionY: 114.263407
  - PathCode: gxigCo~kxT?wA|B?
  - PathLevel: BBB

Buttons: New, Save, Delete, Clear
- Figure 12.C: Point List**

This list shows the current points in the path:

  - (22.33748, 114.26296) Level: 3
  - (22.33748, 114.26340) Level: 3
  - (22.33810, 114.26340) Level: 3

Buttons: AddEndPoint

### 3.3.1 Lift Data Table

Lift Data table (Figure 12.A) is at the top of the control panel. It shows all data of the lifts and kindly corresponds with the mouse movement. After we click one row of the data, this lift data will show in the data editor for your operation.

### 3.3.1 Lift Data Editor

This editor (Figure 12.B) helps us to management the data by new, edit, delete operation. By the help of the Google map and path code generator, we can change the coordinate and the path code easily.

### 3.3.2 Path Code generator

Path code generator (Figure 12.C) turns the points into ASCII immediately. What we need to do is click the points of the path and add them into the points list. (Figure 13)

- Choose a point in the Google map, the latitude and longitude show on Figure 13.A
- We can press “change” button to update the lift position or press “AddPath” button to add it to the point list. (Generate path code immediately)
- The point list add the current point by default, we can delete a point by double click it. After we choose the point, we can press “AddEndPoint” button to add the lift coordinate to finish the path. The figure below shows the process.

Figure 13

This figure shows a detailed view of the lift data editor and point list. The 'Details of Data' form is highlighted with a red box, and the 'Point List' is also highlighted with a red box. A red arrow points from the 'AddPath' button in the map area to the 'Point List'.

**Details of Data:**

- Type: Lift
- Name: Lift6
- PositionX: 22.338103
- PositionY: 114.263407
- PathCode: gxigCo~kxT?wA|B?
- PathLevel: BBB

**Point List:**

- (22.33748, 114.26296) Level: 3
- (22.33748, 114.26340) Level: 3
- (22.33810, 114.26340) Level: 3

Buttons: New, Save, Delete, Clear, AddEndPoint

## 4. Reference

1. Ray-Casting Tutorial, F.Permadi, [link](#), 2005
2. Google map API [link](#)

## 5. Appendix

Academic Building Electronic Map is implemented in Visual C++ 2008 (asp.net 3.0) and Access database.

It's strongly Recommend that you run the 3D canvas map in Chrome, Opera and Firefox. You may have bad experience in IE.