

## CHAPTER FOUR

### SOFTWARE DESIGN, ANALYSIS AND EVALUATION

In this chapter we represent a computer implementation system called Facility Layout Design and Evaluation (FlayDE), which is developed to enable the user solve the equal-area FLP, using the traditional layout methodology described in chapter three. The fully coded version of the algorithm is available upon request. The system also produces an effectiveness value of the displayed layout. The system was designed and built with JAVA Script.

The system is Object Oriented; its functionality is based on the object structure, which consists of main functional objects accompanied by useful supporting objects. The system is events-driven: the objects mostly communicate among themselves as well as the Graphical User Interface GUI elements (user control); thus by raising and watching for certain events. Objects and their structure are explained in detail in section 4.5.

User can communicate with the system through a GUI. It is a multi-window interface consisting of three main windows and one supporting window. These windows have their own functional purpose and elements. Most of the elements are user controls: controls designed to fit the exact problem or deliver the exact functionality. That enhances the usability of the GUI, and accompanied with appliance of error management using error handling routines, makes the system more stable and robust. The GUI is explained in detail in Chapter 4.6.

This application is an on-line integrated in the system, this makes the application accessible “on the move” it can be easily accessed anywhere with computers and mobile devices. This app can provide useful information to the user about the system and how to use it. Alongside this, there is also a table of codes and there corresponding values imbedded in the system to aid the conversion of relationship codes to their values to derive an result.

#### **4.1 SOFTWARE AND HARDWARE REQUIREMENT**

In order to run and use FlayDE, certain minimum requirements of hardware, software and memory must be met. These requirements are listed below:

##### Hardware

FlayDE requires the following hardware:

- IBM PC compatible with Pentium class processor or later model
- Hard disk drive with a minimum of 120Mb of free space, and
- At least 32Mb of RAM memory

##### Software

FlayDE requires the following software:

- MS Windows 98/2000 SP4/CE/Millennium Edition/Server 2000/XP Mediacentre Edition/XP Professional x64 Edition/XP SP2 or later operating system.
- .NET Framework 2.0 or later.

## 4.2 INSTALLING FlayDE

The FlayDE is an online application along with other supporting application and documentation. The following steps describe how to install FlayDE on any windows Operating System which meets the minimum software and hardware requirement:

- Internet connection
- Enter the FlayDe URL
- Once the site opens, enter username and password
- The home page opens on the GUI and the user can start the optimization processes.

## 4.3 DATA ORGANIZATION AND MANAGEMENT

Data is organized in three groups: **definition data, relationships data and result data.**

The definition and relationships data for each layout problem are stored in one plain text file. The first part of the file contains general data of the problem such as name, source and some comments attributable to the facility. It also holds data such as the number of departments to be contained in the facility. The second part contains data of the department. This data includes the name, color coding, and area of the departments.

The third part of the file holds the departmental flow information of the facility.

The fourth part of the file is created when the layout is executed. It represents the result data which consist of the optimization data and layout summary data. The optimization data is made up of the nodal diagram, plant layout diagram, and cost effectiveness of the optimized layout that was produced.

FlayDE manages these categories of data separately so that small changes in one category do not affect the others, ones the user logs in; every work done by the user is saved on his profile. This allows the testing of several layout configurations without having to input all the information each time. Users enter this information directly into the system using textboxes, which also applies for inputs in the subsystem - each one dedicated to a specific category and for as long the user uses the application, past works can be recalled and readjusted if need be to suit a new work or creating a new work if need be.

#### **4.4 IMPLEMENTATION OF THE ALGORITHM**

The layout algorithm described in Chapter Four of this report is implemented in a single class (JAVA Script) called ClassMain. This class implements the algorithm using a number of functions which are designed to return precise information in specific object types.

The system uses the algorithm implementation class by calling two class procedures (subroutine), which acts as the class entry points. The parameter passed is a reference to the problem object, where the results are stored and returned to the system.

Classes are an important part of objected-oriented programming because they allow you to group related items as a unit, as well as control their visibility and accessibility to other procedures. Classes can also inherit and reuse code defined in other classes.

The advantage of the class implementation lies in the possibility of using the DLL (Dynamic Link Library) technology. With a little system redesign the implementation can be advanced upon and compiled into DLL. This would enable testing, analyzing, and

using advanced implementations without needing to apply any changes to the FlayDE system.

## **4.5 OBJECT STRUCTURE**

### **MAIN OBJECTS**

A group of main objects are used to handle the facility layout problem, from problem definition to the results given by the system. Classes representing those objects are: frmLayD, frmRelationships, frmOptimization, and diaNameDepartments.

The frmLayD class is a parent class for frmRelationships and frmOptimization classes, and represents a single problem on a higher level. It consists of methods that support operations like creating new problem object structure and its initiation methods for working with reading and writing data from files as well as methods that cause different ways of displaying a problem. The class through its properties holds information about a problem object structure, by maintaining references to child object structure, by maintaining references to child objects, and links to user interface, by maintaining references to user controls that handle different ways of displaying a problem data. Its methods also support operations like reading and writing data to file as well definition data editing with data validation.

The frmRelationships class is a child of the frmLayD class. It handles departmental flow data in details. Its methods support an operation that displays the objects available to users for input only as required, and controls the labeling of the facility layout. This class is very dynamic and contains error handling procedures.

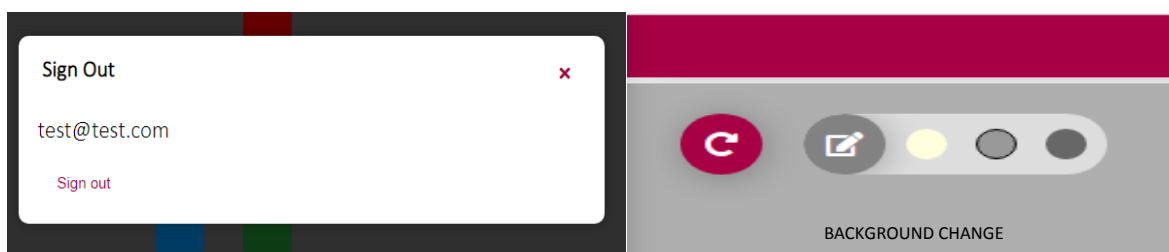
The frmOptimization class is also a child of the frmLayD class. It has two instances, for each type of layout display produced by the algorithm. It handles problem result data in detail through methods that support different display operations for the resulting part data.

The diaNameDepartments is a supporting class of frmLayD class. It handles departments name and color data through methods that support read/write operations as well as different display operations for the output as well as the user interface.

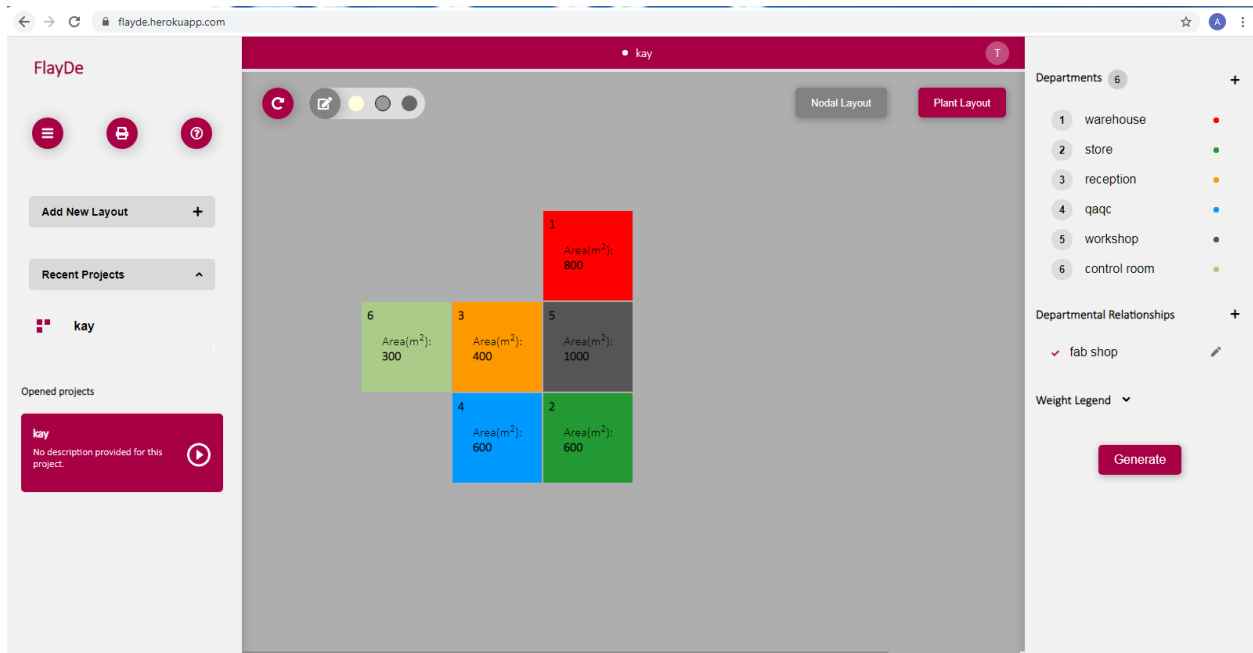
## 4.6 USER INTERFACE

The FlayDE User Interface is a Multi-Document Interface using an MDI Form (the Main Window) as a container for two functional windows: the Relationships Window and the Optimization Window, as can be seen in figure 4.2.

All the features, mentioned in the following text, are explained in detail in the FlayDE on-line help file.



**Fig 4.1: SIGN IN TASK BAR**

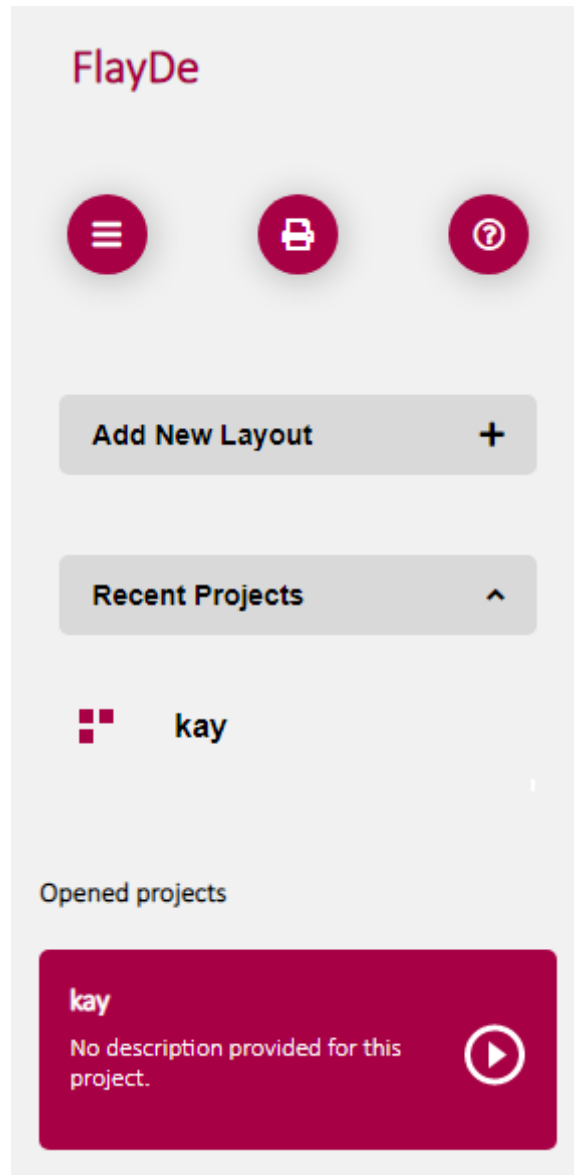


**Fig 4.2: FLayDE**

#### **4.6.1 THE MAIN WINDOW**

The purpose of the main window is to help the user work with a single problem and its details more easily.

The Main Window consists, as shown in Figure 4.2, five main elements: the main menu, two panels, and two child windows.

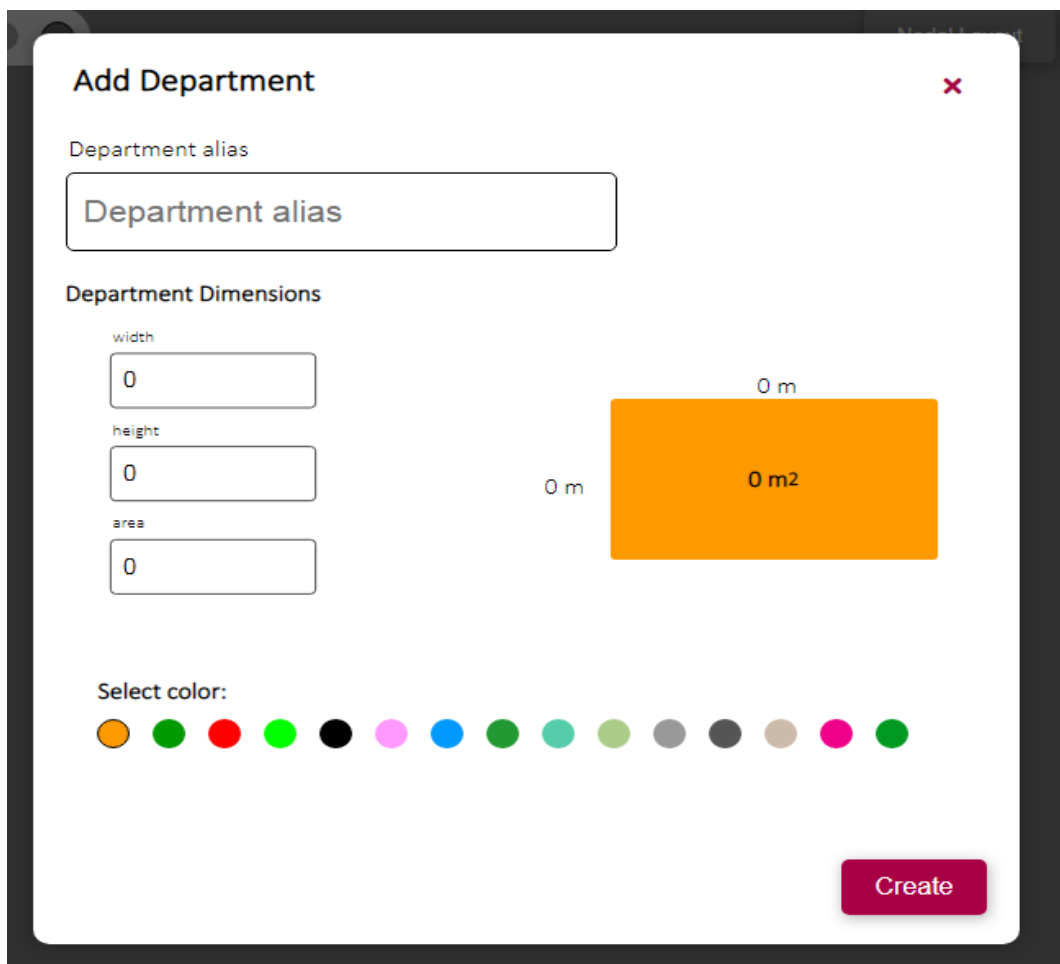


**Fig 4.3: FlayDE Main Window**

The main menu helps execute tasks, such as saving and printing problems, and executing, profile name and resetting the various child display style. It also contains the help menu and a means for exiting the program.



The problem general variables area is located immediately under the Main Menu. It consists of six general variables, defining the FLP: the name of facility, two comment variables (the source of the problem and a general comment the user can give to the problem), area of the departments, number of departments, and a link to a dialog where the name of the departments (see Figure 4.4) as well as a user chosen color. The color selection is made possible by the color dialog provided for each department (see Figure 4.4).



The image shows a dialog box titled "Add Department" with a close button (X) in the top right corner. The dialog is divided into three main sections:

- Department alias:** A text input field containing the placeholder text "Department alias".
- Department Dimensions:** This section contains three input fields for "width", "height", and "area", each with the value "0". To the right of these fields is a visual representation of a rectangle. The rectangle is orange and labeled "0 m2". The width and height are both labeled "0 m".
- Select color:** A row of 15 colored circles for selection. The colors are: orange, green, red, yellow, black, pink, blue, dark green, teal, light green, grey, dark grey, brown, magenta, and dark green.

A "Create" button is located in the bottom right corner of the dialog.

**Fig 4.4: Name Departments Dialog**

#### 4.6.2 THE RELATIONSHIP WINDOW

The Relationships Window is a highly dynamic child window in the MDI system of FlayDE. It consists of three (beside a standard window bar with the name of currently opened problem) elements and controls: the Chart, the Conversion Aid, and the Show Department Name Checkbox. (Figure 4.5)

The screenshot displays the 'Edit Relationship for : fab shop' window. It features a 6x6 matrix for defining relationships between departments 1 through 6. The matrix cells contain numerical values representing the relationship strength, with some cells highlighted in red. To the right of the matrix is a 'Weight Legend' section that maps letters to relationship types and their corresponding numerical weights. A 'Generate' button is located at the bottom right of the legend area.

	1	2	3	4	5	6
1		4	2	3	4	2
2			3	4	4	3
3				4	3	2
4					4	3
5						4
6						

Weight Legend		
A	Absolutely Necessary	4
E	Especially important	3
I	Important	2
O	Ordinary	1
U	Unimportant	0
X	Undesirable	-1

**Fig 4.5: Relationships Window**

The Chart (as in Figure 4.5) is the major element of the Relationships Window. It provides a 2-Dimensional array of textboxes where users can input departmental relationships for up to twenty departments. This array can be reduced, based on users' input of the number of departments that is required for the FLP, thus providing a dynamic chart system. The chart only accepts values of relationships and not the code, and it has an error handling system to ensure users input valid values. Based on this

error handling system, each time a textbox loses focus, the system checks that the variable entered IsNumeric and falls within the range of values (i.e. between 4 and -1), and is not a letter or IsNothing (i.e. empty). If the validation process returns correct, users can proceed to the next textbox, else the textbox would refuse to lose focus. The validation process serves to check for system crashes.

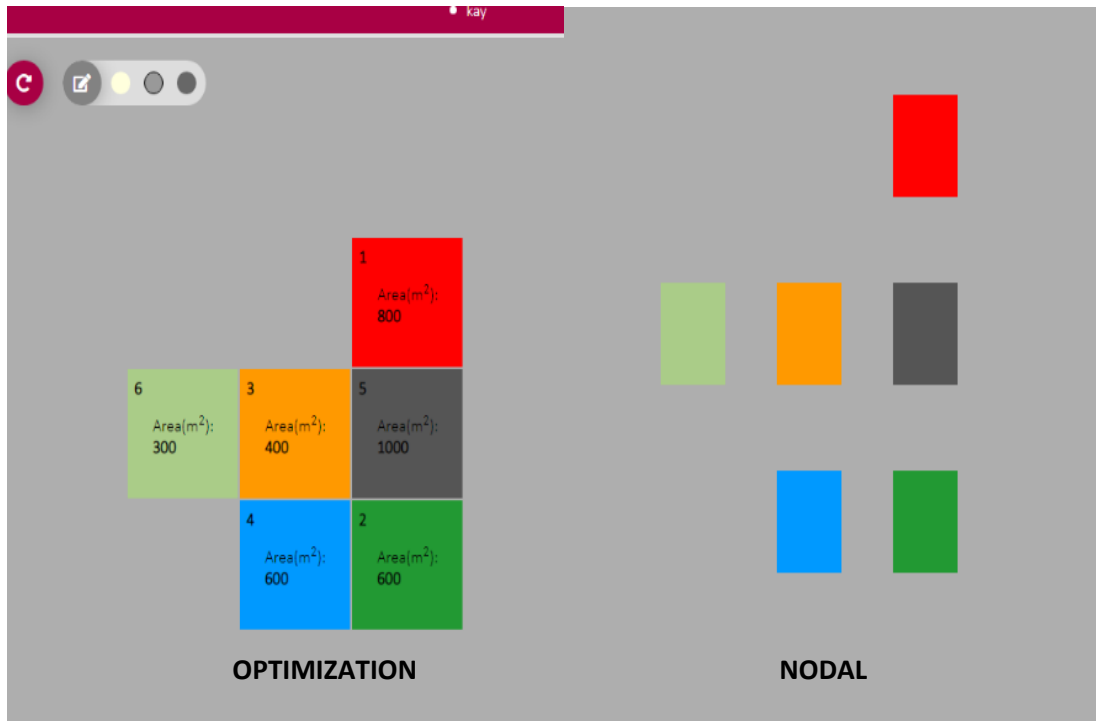
The Conversion Aid has no system functionality except to aid the user in the conversion of the Rel-Chart to the Value Chart. It is a repetition of that **shown in Table 3.1.**

The decision to show the name of the department in the layout or just a reference number corresponding to the department is made by the class ClassMain each time the “Optimize” button is clicked. It uses the “Show Department Name” checkbox immediately above the Optimize button. The ClassMain evaluates if the Show Department Name checkbox is checked. If it is, the name of departments entered in the Name Departments Window is displayed in there corresponding modules, else the reference numbers are used.

#### **4.6.3 THE OPTIMIZATION WINDOW**

The Optimization Window serves as an output element which displays the layout as well as has some visual-aid functionality.

It consists of three elements and control: the Nodal Diagram, the Plant Layout, and some visual-aid buttons. Figure 4.6



**Fig 4.6: Optimization Window**

The Nodal Diagram and the Plant Layout elements are contained in the TabOptimization TabControl (TabControls are explained in section 4.6.4). The Nodal Diagram displays (as the name implies) the nodal diagram of the layout while the Plant Layout displays the Plant layout of the FLP. These displays combine to give the user an elaborate idea of the position of the various departments in the facility.

On the right corner of the Optimization Window consist of three buttons which constitute the visual-aid. The “ZOOM+” button enlarges the display of the TabControl that has focus 25 percent on each click. The “ZOOM-”button reduces the display on the TabControl that has focus 25 percent on each click. The “RESTORE” button, as the name implies, restores the display to its original dimension.

## **4.7 ClassMain**

The ClassMain is the main class, and is responsible for the implementation of the algorithm described in Chapter Three.

It exposes a number of methods, namely “DepartmentalScripts”, “DepartmentalSequence”, “DepartmentColor”, “HorDepartmentalRelationships”, “Points”, “RelationshipSorts”, “VetDepartmentalRelationships”, “Main”, and “NodalMain”. These methods combine to make the last two functional and would be described in the preceding texts. The Main and NodalMain methods are the entry point of the ClassMain.

### **4.7.1. DepartmentalScripts()**

The DepartmentalScripts is a Function that if passed the department’s reference number (using the calling code) would return the Name of department. Details of these are contained in the fully coded version of the algorithm which is available upon request.

### **4.7.2. DepartmentalSequence()**

The DepartmentalSequence is a Function that accepts the sequence of entry number or element (using the calling code) takes the sum of the horizontal relationships as well as the vertical relationships (as in the chart in the Relationships Window) to get a total sum for each department. The departments are then sorted according to the total sum, in descending order, and inserted into an array. Whenever DepartmentalSequence is called it returns the department sequentially, as they are in the array. Details of these are contained in the fully coded version of the algorithm which is available upon request.

#### **4.7.3. DepartmentColor()**

The DepartmentColor is a Function that if passed the department's reference number (using the calling code) would return the color of department as inputted by the user in the "Name Departments" Window.

#### **4.7.4. HorDepartmentalRelationships()**

This is a Function that returns all the departments having the required horizontal relationship (4, 3, 2, and 1) with the specified department. It takes in the specified department as well as the required relationship as parameters in order to achieve its functionality.

#### **4.7.5. Points()**

The Points is a Function that returns an array of points holding the coordinates of the candidacy location of the next departments to be placed. It accepts the coordinates of the placed departments when it is called. It is this Function that affects the shape of the facility displayed in the Optimization window.

#### **4.7.6. VetDepartmentalRelationships()**

This is a Function that returns all the departments having the required vertical relationship (4, 3, 2, and 1) with the specified department. It takes in the specified department as well as the required relationship as parameters in order to achieve its functionality.

#### **4.7.7. RelationshipSorts()**

This Function is similar to the HorDepartmentalRelationships and the VetDepartmentalRelationships Functions described in section 4.7.6 and 4.7.4 respectively. The difference being that the RelationshipSorts checks for both vertical and horizontal relationships in on call. It first checks for horizontal relationship of 4, if it does not find any, then it checks for vertical relationship of 4. If it does not find any, it checks for 3 relationships, if it does not find any -it continues until it finds- to relationship -1. After carrying out the above process and finding any horizontal or vertical relationships at some stage, it returns an array holding all the departments with this relationship.

#### **4.7.8. Main()**

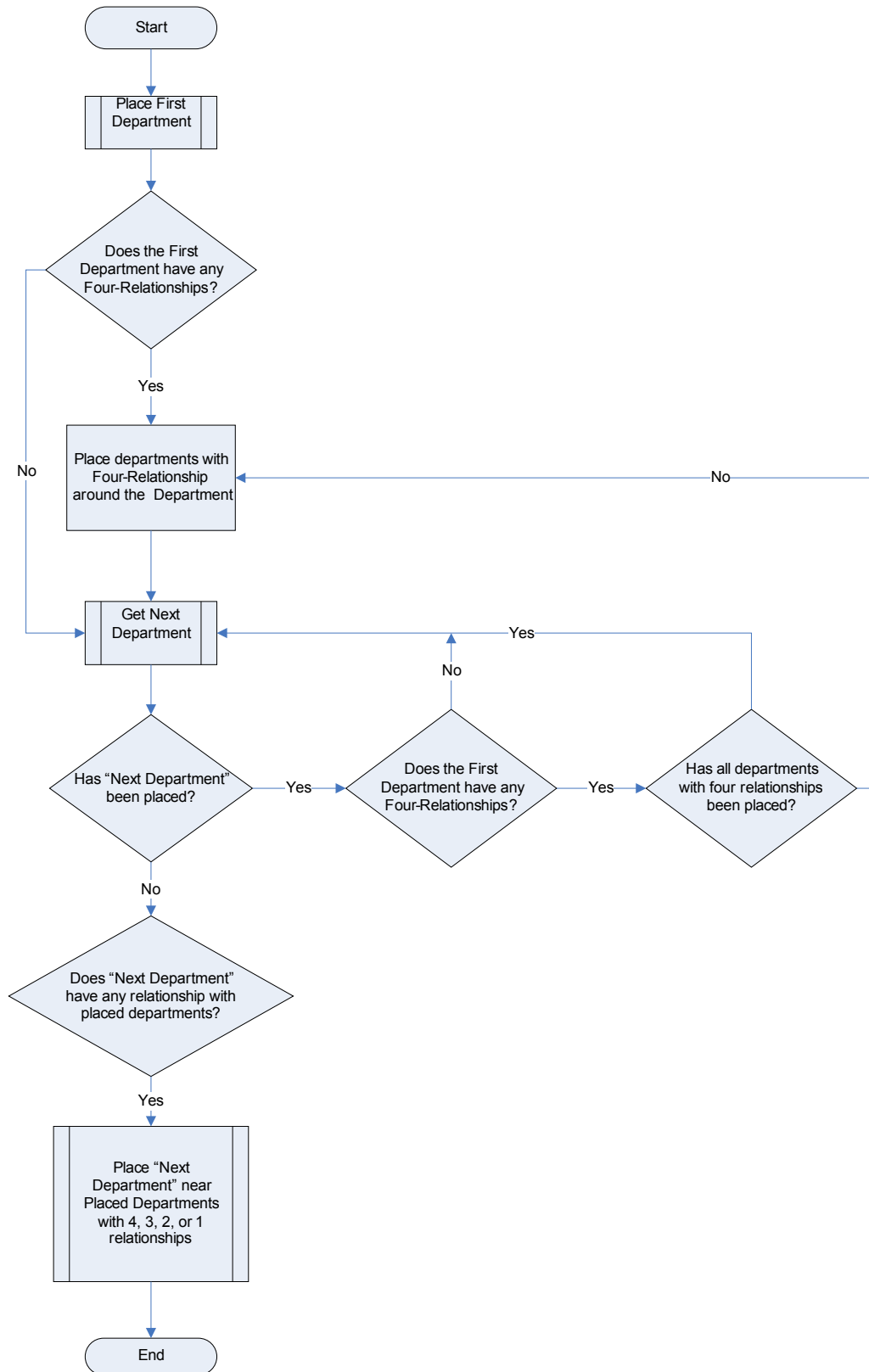
The “Main” is a very important procedure (subroutine) since it is the entry point to the class ClassMain which implements the algorithm described in Chapter Three. It also links all the Functions described above, to carry out its own functionality. The flow chart of the algorithm is shown in Figure 4.6.

The procedure starts by trying to place the first department in the centre of the Plant Layout. It does this by calling the DepartmentalSequence Function (described in section 4.7.2, and passing 1 as argument) which returns the department with the highest relationship; next, it gets the color of the first department by calling the DepartmentColor function (described in section 4.7.3, and passing the value returned after calling DepartmentalSequence as argument) which returns the color of the first department; the

centroid of the Plant Layout is then calculated and used as the coordinated of the first department.

Next, the HorDepartmentalRelationships Function is then called to ascertain if the first department has any four-relationships. If it does these departments are placed about the first department with the help of the Points Function. The Points Function is called passing the coordinates of the first department as argument. The coordinate points returned (section 4.7.5) are checked alternatively to determine if they are occupied. If they are the next coordinate point is checked, and so on, until a free location is discovered.





The DepartmentalSequence (this time passing two) is called again to get the “next department” that has the highest relationships. If this department has not been placed, the “Main”, calls RelationshipSorts (section 4.7.7) passing “next department” as argument. The RelationshipSorts returns the departments with the appropriate relationship, and the “next department” is placed next to these departments (those that have been placed) calling DepartmentColor and Points.

If “next department” have been placed, the “Main”, calls HorDepartmentalRelationships, passing “next department”, to get an array of the departments having four-relationship with it. This is then placed about the “next department”, each time calling DepartmentColor and Points. Details of these are contained in the fully coded version of the algorithm which is available upon request.

#### **4.7.9.        NodalMain()**

The NodalMain is closely related to the Main.