Environmentally-Oriented Travel Advisor

Andrzej Laczny
Adam Szarowicz
Ireneusz Szczesniak

Socrates exchange students at the Nottingham Trent University, Nottingham, UK
from the Silesian Technical University, Gliwice, Poland

Supervised by Dr Joanna Hartley

The Nottingham Trent University, Nottingham, UK

May 2000

ABSTRACT

The mission of this project was to introduce to the public the advantages of using a travel information device whilst travelling through the city of Nottingham using a car and bus. This aim was accomplished through the medium of a game. The project is a continuation of the research conducted by the Simulation and Modelling group within the Department of Computing at the Nottingham Trent University.

The development of the traffic game involved the incorporation of a graphical user interface, a database and a shortest route algorithm. The project delivered a working application as well as research results in the field of the three mentioned subsystems. The deliverables form the solid platform for further development based on enhancements as well as adding new features. Due to the low design cost achieved so far, any future development is estimated to be very cost effective as well.

1. INTRODUCTION

1.1 Overview

Increasing traffic, congestion and environmental pollution are one of the main problems of every city. To cope with the problem efficiently government agencies (such as the Nottingham Traffic Control Centre) have been established to control urban traffic flow. These agencies also provide the general public with travel information so that they help to resolve the problems by choosing appropriate travel routes. There is however a lack of software to inform the traveller of the optimal journey through the car network and public

transport network. In order to fill this gap, the Environmentally-Oriented Travel Advisor has been developed.

The application is a new solution in the field of traffic control. It was developed for a PC-compatible platform using the Windows 95/98/NT system environment. It combines entertainment with educational features and also helps users in optimising their travel (in terms of how long it takes to get to work or school). The application also takes into consideration environmental issues in terms of pollution caused by vehicles travelling through the city and allows users to differentiate the pollution levels caused by different types of vehicles. The application does not use any 'on-shelf' products.

The project was divided into three parts - each implemented by one member of the team. The subsystems are as follows:

- graphical user interface allowing the user to interact with the game, displaying the map and giving appropriate feedback;
- database containing all data about the city - maps, bus routes, bus timetables, lengths of queues forming at the junctions at various times of the day and pollution factors;
- algorithms for finding the shortest route in the city.

Let the reader note that the three mentioned subsystems which were developed simultaneously. Therefore they differ in terms of approach, design and implementation methodology and issues considered in the development. The subsections *Graphical User Interface*, *Database* and *The Shortest Path Algorithms* were provided by the team members responsible for the subsystems and address their main principles.

The research has been carried out in the Simulation and Modelling Group at the Nottingham Trent University. The group cooperates with local agencies, such as Nottingham Traffic Control Centre, and international corporations, such as PSION.

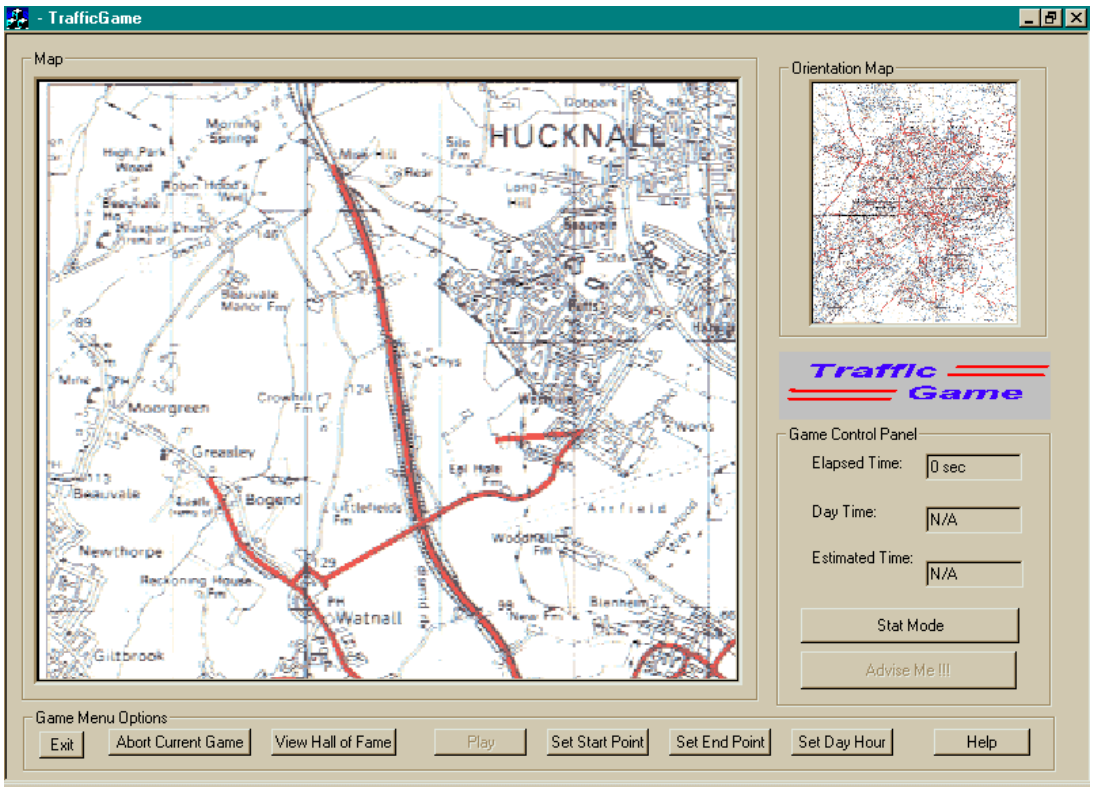

Fig. 1.1. Final Version of the Graphical User Interface Main Screen.

1.2 Communication Between the Subsystems of the Traffic Game

The application displays (in the main window) a map of the city and a set of buttons (see Fig. 1.1) used to control the game. The map of the city is in fact a coloured bitmap, easy to read and clear to the user.

Apart from the visible bitmap, the application also has access to a second bitmap (loaded by the interface part) on which all nodes and sections are coloured in different colours so the colour number is a unique identifier for every node or section. Thus it is very easy to differentiate which object is actually pointed to. Coordinates of all objects are stored in the database so it is possible to perform desired operations on the selected area (e.g. highlighting it).

A three-layer graphical representation is depicted in Fig.1.

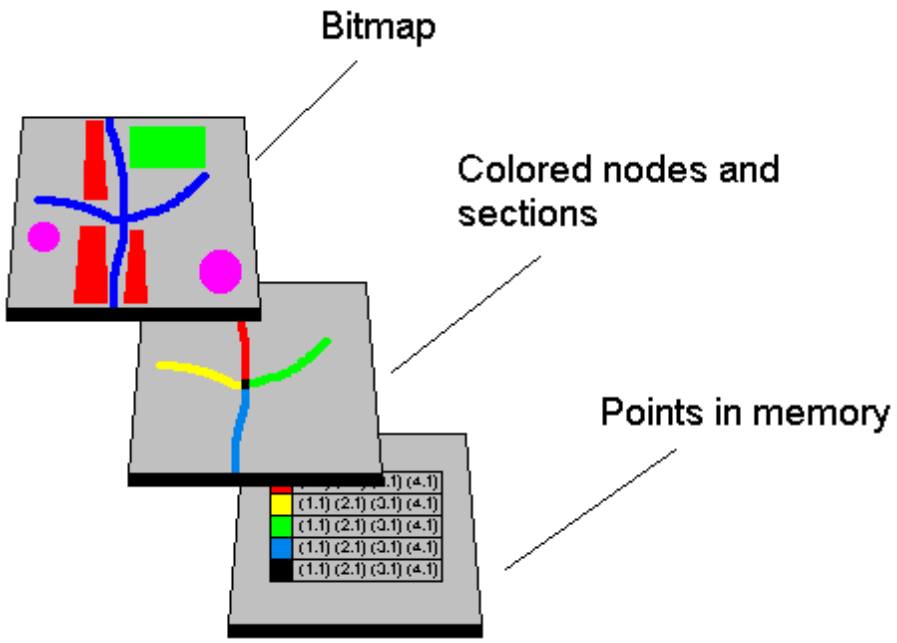

Fig. 1.1. Application Layers.

The user clicks on the map in order to choose the starting and ending node of the journey. The interface gets co-ordinates of the pointed pixel and recognises its colour (= identifier) on the second layer bitmap. Next, the query to the database is generated and all points making the chosen object are extracted so that the object can be highlighted (if necessary). If both points are set and the user chooses the ‘Advise route’ command from the menu the algorithm function is called. The function - based on the logical (connections), numerical (queue lengths) information and current mode (time, length, pollution priority) - finds the optimal route (using the algorithmic object) and returns a set of object identifiers (nodes and sections). Given the identifiers the application can again demand geographical data about them (from the database) in order to highlight the route. The user can also find his own way through the city or set the actual hour. The example sequence of operations is depicted in Fig.1.2.

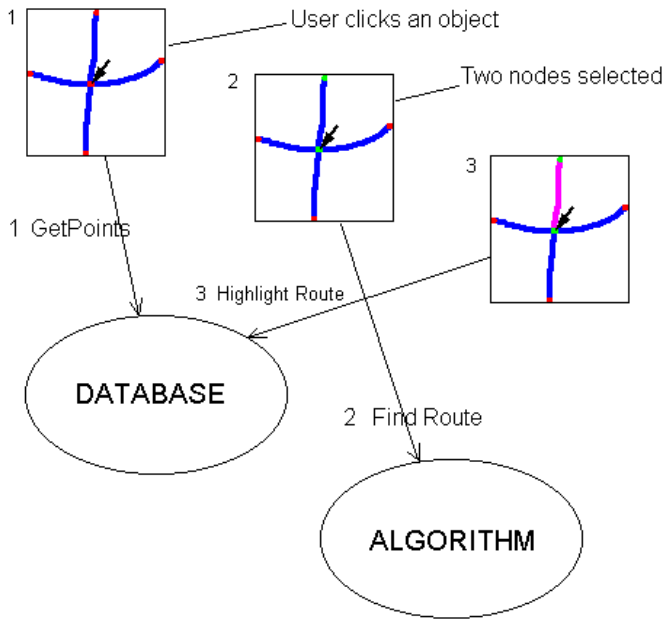

Fig. 1.2. Cooperation of Subsystems.

Communication with other subsystems developed by the team members was assured through the concept of interfaces. In the very beginning of the project implementation, when other subsystems were designed and their tasks were known, such interfaces were created. They were represented by C++ header files containing declarations of functions, which were accessible for other subsystems in order to request data, perform calculations, etc.

The next three sections address the three subsystems of the Traffic Game. The first one to be described is the *Graphical User Interface*.

2. GRAPHICAL USER INTERFACE

2.1 Introduction

The Graphical User Interface (GUI) was developed using an object-oriented model and the subsystem was developed based on the prototyping approach. The prototyping approach is recommended for user interface design rather than the popular waterfall software model, because of the fact that requirements concerning this subsystem often are subject to change. The product can finally meet the requirements by many diverse ways. While according to Redmond-Pyle and Moore (1995) in the waterfall software model (see Sommerville, 1995) requirements must be frozen for a long period of time. Let the reader note that the GUI features highly depend on the algorithm and database subsystems. Therefore such a static approach to the requirements may not be the appropriate one. A more appropriate approach would be an iterative approach, namely the prototyping approach.

The design prototypes of the Graphical User Interface were discussed with future users and the optimal solution was achieved. The decision whether the subsystem meets requirements were made after its evaluation by the supervisor, other programmers and users. The biggest disadvantage of this approach is that it is very time consuming. This disadvantage can be reduced using Rapid Application Development (RAD) tools and library packages (such as Microsoft Foundation Class (MFC) Library supported by Visual C++ 6.0 development environment).

The subsystem had been evolving till it reached a satisfactory level. However software created in an evolutionary way has a very bad structure, which is hard to maintain, understand and tends to be defect prone. It was only created to "simulate" the application environment. So finally the prototype was thrown away, and the subsystem was coded once again and prepared for integration with other subsystems. As it can be noticed the description above integrates three software processes: prototyping, RAD and package based development. The object-oriented approach was used to a certain extent. The MFC library forces its own class hierarchy. Therefore the design is not hardware and system platform "independent". This is one of the disadvantages of RAD and package based development models. They support creation of software dedicated for particular system platforms.

The results of the GUI as well as the whole Traffic Game evaluation can be found later in the paper.

2.2 Implementation of General Design Principles

This section addresses the main principles of the user interface design. The framework was provided by Faulkner (1998) and applied to the Traffic Game. The principles were implied to many elements of the design. The most important implications are discussed below. Let the reader note that often more than one principle influences the layout and function of the discussed elements, therefore only the main principle influencing the element is discussed.

The key issue in UI design is usability. Gardiner and Christie (1987) define usability as the joint function of the system and user. Therefore it can not be fully analysed, understood and evaluated (usability problem). The cognitive psychology is the tool, which helps the designers to overcome the usability problem. Gardiner and Christie (1987) define cognitive psychology as the term referring to the scientific study of acquisition, storage and use of knowledge by the individual. The above definition is very broad and the following principles clarify it referring to the practical implementation – the Traffic Game GUI (Fig. 1.1).

Supportiveness

Since such users have less understanding of the task domain as well, the computer should have the initiative throughout the application. Therefore the Traffic Game guides the user through all the game lifecycle letting the user decide when to proceed. The novice user must have time to analyse the screen before he is ready to proceed. Apart from the actual game, when the time is measured to assess the user performance, all the dialog boxes stop the application waiting for the user to press a button.

The decision making is prompted by the system since the user usually does not know when he is supposed to make decisions. These prompts must be minimised as should the number of options. Many applications disable options, which can not be accessed from a certain place within the program or take advantage of short and long menus. The user must be in control of the pace of interaction with the application, too. The novice user usually wants to see and read everything, therefore he should decide when he is ready to move on.

Relevance

The user input in the Traffic Game is brief, because users may not have many experiences with keyboard or mouse. However the latter is easier for novice users to operate. If the application has additional parameters, their configurations should be put away from the critical path in the game lifecycle. This was enforced in the Traffic Game by introducing the default values of parameters (such as time of day). Thus the first couple of tries of the game can be run without any additional changes. Such default parameters are implemented in

appropriate dialog boxes throughout the game. The dialog boxes include short and clear information.

Flexibility

One of the most important principles is that there should be no special training necessary to play the game. All the information concerning the game should be within it. The user must not be forced to remember things. Everything should be shown for him/her on the screen. Let the reader note that the application usage learning process is more effective when it is a *learning by doing* approach. The Traffic Game is a quite simple piece of software at this implementation stage which supports flexibility.

The shown messages should be obviously clear and adjusted to the appropriate user level. Different messages are clear and understandable by experts and different by novice users.

Consistency

The options are not ambiguous and every one of them performs a clearly defined action if allowed at a certain point in the game lifecycle. Intermediate users usually are able to maintain semantic knowledge of the task they want to perform and computer concepts involved in it. They often make assumptions about options similar to other applications. Therefore the application must be consistent in this sense as well as consistent internally. In other words, options must always take the same actions in the same way, because this is what the user expects from them.

Additionally let the reader note that the game was developed using the MFC library. Therefore the general layout is similar to other windows applications. This shortened the development time and made the application easier to master.

Naturalness

The language and symbolic information is simple and clear throughout the application. For instance the colours used can be related to their real world applications. If the link is highlighted with the green colour, it is the possible travel route from the current location on the map.

These were the main principles considered while designing the Traffic Game user interface. The factors such as reaction time and attention are discussed below. The movement time factor was not considered in the design.

Attention

The kind of attention required from a user while playing the Traffic Game was identified as selective. There is more than one information channel on the game screen. The user must track the changes on the map, which shows changes using graphics, and numerical values, which represent his score and other important elements. The channels are divided by symbolic lines grouping information within one channel. In order not to make the user tire quickly the number of channels should be reduced to a minimum. The designer must make it clear to the user what are the relevance levels for all the channels. The clear descriptions of screen elements tell the user where he/she should expect the result of the undertaken action.

There are three visual communication channels in the Traffic Game. The main one is the map and attracts most of the user's attention for most of the time. The second one is the small orientation map, which provides an overview of the whole available city area. This channel is additional meaning that the user may finish the game without taking advantage of it. The third communication channel is also used rarely during the game. It is the information

and option buttons area. The user refers to it mainly during the initial stage of the game.

The project team considered changing the above attention model to be more focused. If the numerical information showed in the other part of the screen was combined with graphical information on the map, the number of information channels could be reduced to one. This would ensure that the user was not distracted by other elements. However, the richer the symbolic information on the map would be, the more cautious the designers should be about the clarity of passed information.

Reaction Time

The dialog boxes are situated in the centre of the screen. Such a solution enhances the reaction time for the information in the dialog box.

The Traffic Game takes advantage of only one communication channel such as visual channel. This is due to time constraints. However as it was proved (see Faulkner, 1998) that reaction for the audio stimuli is quicker, than for the visual one. The combination of the two communication channels would be very effective. Therefore it will be introduced in future releases of the software.

The next section deals with issues concerning the database subsystem.

3. DATABASE

3.1 Database Subsystem

The database for the traffic game contains both graphical and logical information and supplies suitable interfaces for GUI and algorithm parts. Graphical information includes localisation (coordinates) and area (included points) of all graphical entities (namely nodes and sections). Logical information involves links between nodes and bus routes. The database also contains additional information such as bus timetables, pollution factors and traffic intensity function. Data about nodes and the links between them are used by the algorithmic part of the program, spatial data (graphical information) are supplied to the user interface part. Appropriate methods are applied to meet all defined requirements. All types of data stored in the database are described in detail in the following sections. The database was designed using object-oriented methodology and documented via UML notation, which is a widely recognised standard notation for object-oriented development.

3.2 Database Layers

As mentioned before, the traffic game application operates on five main information layers:

- bitmap of the city - this layer makes the visible part of the application. It is a raster view (bitmap) of the city, containing all graphical information - roads, lanes, parks, lakes, rivers, etc. - in different colours. The city bitmap is not stored in the database but in a separate .bmp file supplied by the map editor (Geographers' A-Z Map Company Ltd).
- geographical information about spatial objects - coordinates of raster points which create roads and nodes (junctions) together with unique numerical identifiers.
- logical data about connections between nodes - remembered as triples of numbers a-b-c, where a and c are node identifiers, and b is the section identifier.
- textual (descriptive) information about database entities - road names, junction information (names of buildings, node labels, etc.).
- other numerical data - pollution factors, distance factors, traffic intensity function.

The last four types of objects are actually stored in the database.

3.3 Preprocessing Data

In order to effectively load and access the data the preprocessing stage was required, which involved both graphical and logical data.

As mentioned in the *Introduction* the graphical information was split into three layers: city bitmap, coloured objects surface (mask) and numerical coordinates of all points. The latter two layers are actually stored in the database.

The process of preparing the data was divided into several stages:

a) marking the relevant roads with a different colour and cutting out all other parts of the bitmap (Fig. 3.1b)

b) marking nodes (Fig. 3.1c)

c) colouring every object (node or section) into a different colour (Fig. 3.1d)

d) creating connections between nodes and saving them into a file (Fig. 3.1e)

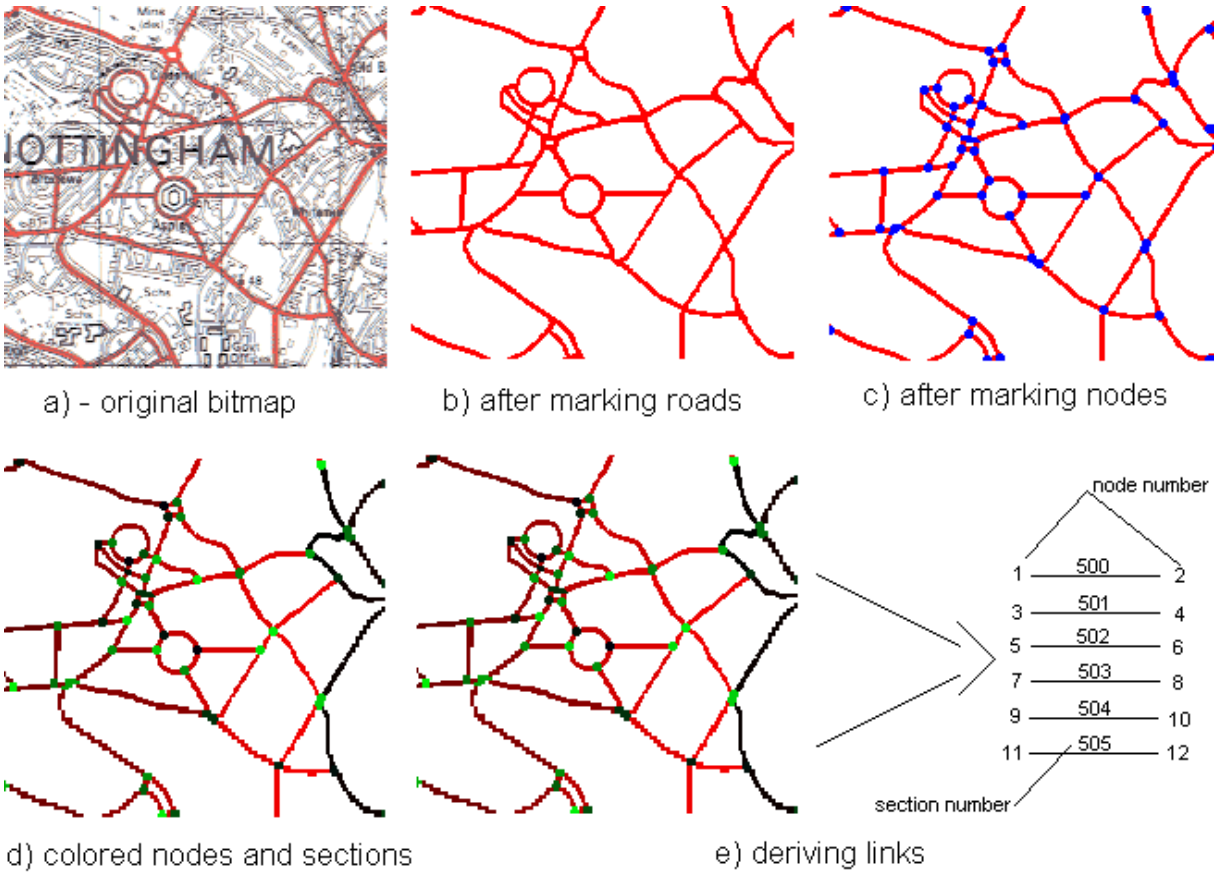

Fig. 3.1. Bitmap Preprocessing Scheme.

Stages a) and b) were done manually using a bitmap editor. To perform stages c) and d) a dedicated program was written. This program (coded in Borland Delphi 3.0) allowed a user to load bitmap obtained from stage c). Next it assigned a different colour to every node and section and created two-directional links between every pair of nodes. The program's window is shown in Fig. 3.2.

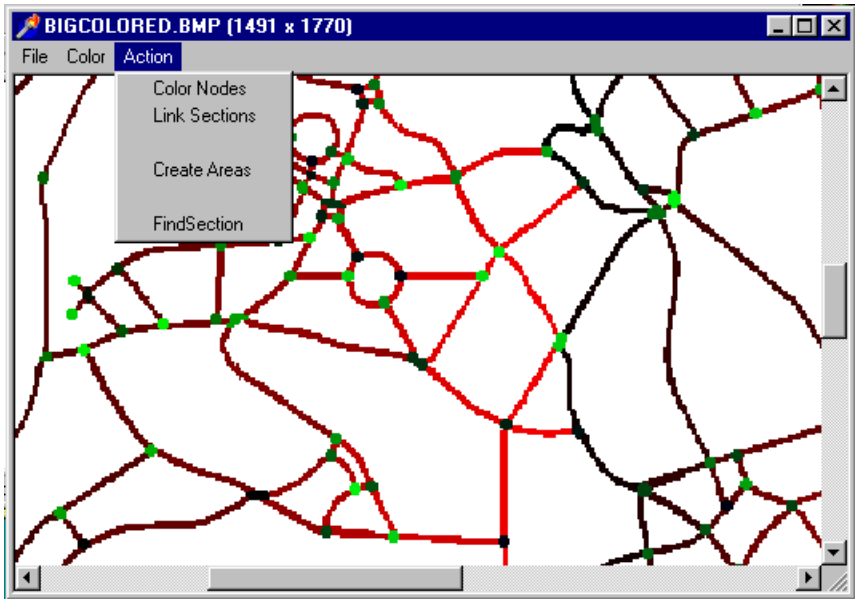

Fig 3.2. The Preprocessing Tool.

Having prepared the map and the link file it was also necessary to produce a file with bus routes. To allow this another Delphi application was written. The application supported inputting nodes visited by every bus line.

3.4 Alternative Solution

Another possible solution to developing the database subsystem was to use ready numerical data available from a traffic map editor TEDI developed at the Universitat Politecnica de Barcelona. This approach was also investigated and the results are presented in this section.

During previous projects conducted by The Simulation and Modelling Group all main roads in Nottingham were input in TEDI. Thus the team was given ready city data at the beginning of the project.

TEDI allows the user to load maps of city traffic network and outputs them as text files thus constructing a foundation for a geographical database. The data form a skeleton of the city road network with lanes, junctions and nodes.

Unfortunately the information extracted from TEDI did not comprise any additional graphical elements such as parks, rivers or street names. Therefore such city map was very unclear and difficult to read. Moreover preprocessing the numerical data turned out to be very labour-consuming and the program itself showed to be unreliable. The map used was also not complete. Main problems encountered were as follows:

- the program did not allow the user to remove duplicate nodes,
- the program produced output in a very ‘poor’ form - sections were built of rectangles and no other graphical data were available,
- the program needed a lot of time to load into memory and to save data to disk,
- it was necessary to often save changes due to high possibility of crashes,
- the available map had inconsistent links (not always connected, or connected to wrong nodes),
- the data were missing (not all nodes and sections were present on the map).

- all produced files needed preprocessing before being used in this project (in order to do the full preprocessing a number of simple programs were written, which transformed available files into desired format - this is depicted in Fig. 3.3).

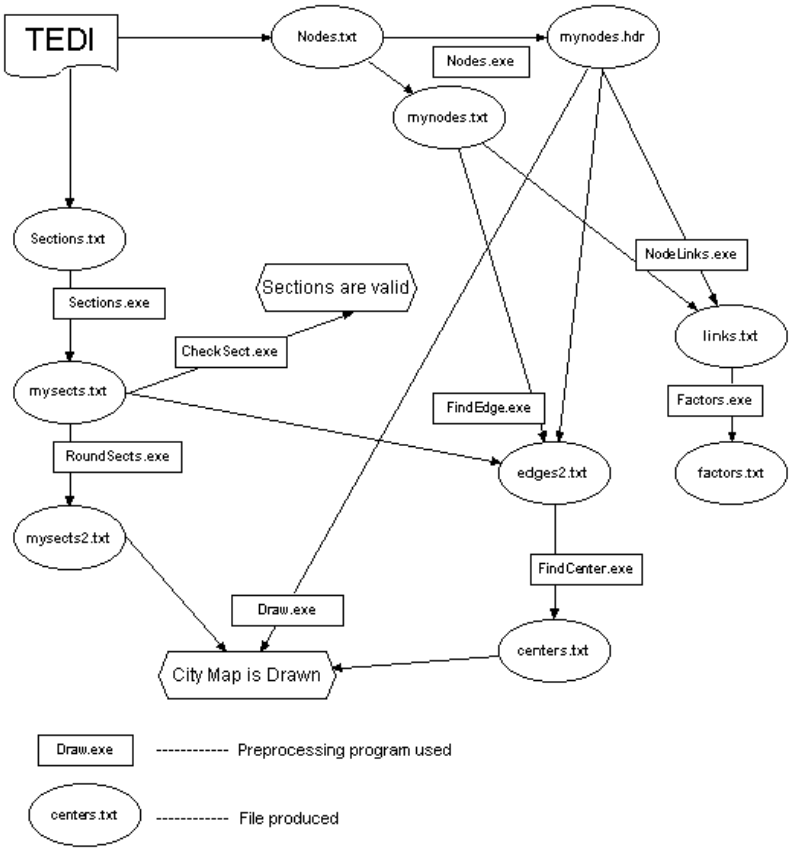

Fig. 3.3. Preprocessing Schema.

Finally after assessing the gained results the team decided to abandon this solution and develop the application using the bitmap approach described before. The comparison of outcomes from these two approaches is depicted in Fig. 3.4.

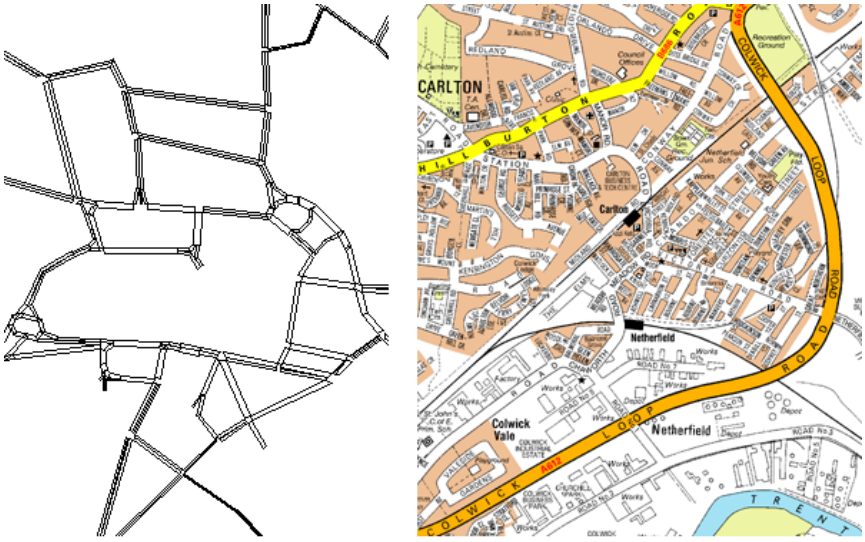

Fig. 3.4. Different Approaches of Displaying Map.

3.5 Database Testing

The database was intensively tested especially regarding memory usage. This was due to the high possibility of memory leaks caused by an extensive usage of dynamic structures. The database and the dedicated preprocessing application were also tested on different machines to see if the loading times depended on the machine configuration. The results are shown in Fig. 3.5.

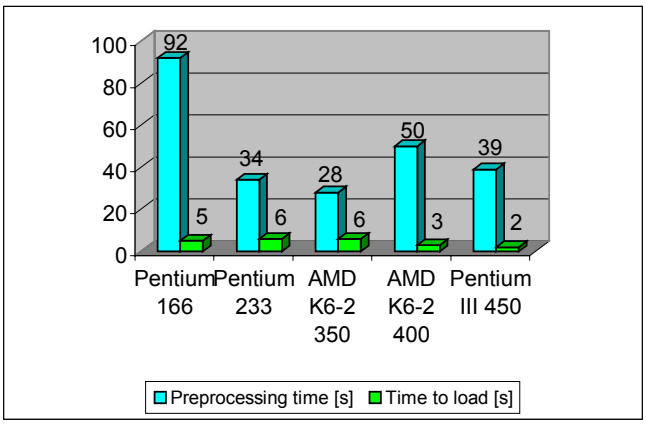

Fig. 3.5 Preprocessing/Loading Times.

3.6 Summary - Constructed Database versus GIS systems

Although the database contains many elements typical for GIS systems, it is not a complete GIS environment. The reasons for the differences are depicted below.

There are 3 possible architectures of GIS (Geographic Information Systems):

- ad hoc systems
- systems based on relational DBMS (Database Management Systems)
- systems based on extended DBMS or objects oriented databases.

It was neither possible nor desirable to write a complex DBMS for the purpose of this project. The aim of this work excluded also usage of any existing commercial DBMS (the application should be standalone and available for an ordinary user what implied low cost of the software). The database part was expected to be simple and efficient, it was not intended to be used in any other application. In these circumstances the best idea was to design and code a dedicated system. According to Agnes Voisard (1998) this approach allows the construction of a database which:

- is not modular
- is not extensible
- is not friendly
- **is very efficient**

Additional features of such a database would be low cost and independence of other software.

Reusability and modularity were not crucial because the database is used only in this project, lack of 'friendliness' is not a big problem because the database is accessed through a well designed interface and the end-user is not expected to make any directed use of the database. The chosen approach allowed constructing an efficient, dedicated, pseudo-GIS database system with elements of object-oriented databases, which fulfils all the requirements defined at the beginning of the project.

4. THE SHORTEST PATH ALGORITHMS

4.1 Introduction

The shortest path algorithm helps the travellers to optimise their journey through the urban network. Each path can make use of a car, buses and the Park & Ride system. A Park & Ride location is a car park with a bus stop. Such a place is useful for commuting from home to work – one can drive a car to a Park & Ride location, park there and then take a bus from there to work.

The algorithmic part implements three algorithms. The first is the Dijkstra algorithm for car networks and the next two are the author's algorithms: for finding a route in a bus network and for finding a path composed of both car and bus travel.

4.2 Environmental Issue

Suppose there is a need to find a path which implies the smallest emission of pollutants. The shortest path in terms of pollution emitted needs to be evaluated from the environmental point of view.

The approach to the problem is to ascribe a cost to every link that expresses the impact on the environment. Therefore a higher cost will be attached to a car link, and a smaller cost will be attached to a bus link. The cost of a link should be dependent on the length of this link. According to the cost defined in such a way the shortest environmental path is found.

4.3 Definition of the Desired Shortest Path Algorithm

The algorithm has to find the shortest path from one given place to another given place using both a car and means of public transportation based on timetables (the project uses buses). The road network and the public transport network based on timetables are given as the algorithm's input. The required output is composed of car links and bus links.

Fig. 4.1. depicts the four forms of shortest paths that the algorithm can generate. In the first and second case the algorithm can advise travel by buses or car only. The third and

the fourth case involve both a bus and a car. The third form of the shortest path starts with bus links, reaches a Park & Ride and continues with a car to the destination. The fourth form of the shortest path begins with a car, continues to a Park & Ride and reaches the destination by buses (a user can change buses any number of times).

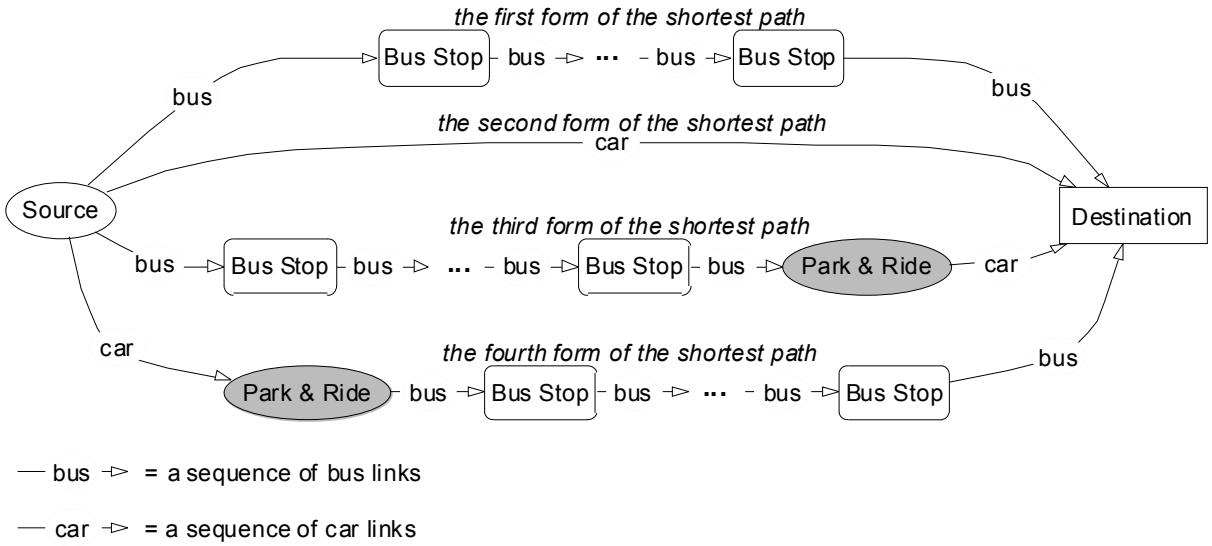

Fig. 4.1. Desired forms of a shortest path.

The desired shortest path (SP) algorithm is a combination of the bus algorithm described in 4.5. and the Dijkstra algorithm. The bus algorithm finds the bus links of the shortest path and the Dijkstra algorithm finds the car links of the shortest paths.

4.4 The Dijkstra Algorithm

Dijkstra (1959) proposed a sequential algorithm for networks with non-negative costs of links. This algorithm is efficient and requires little memory and therefore it has been chosen to search for the shortest path in a car network.

4.5 The Bus Routing Algorithm

The bus algorithm presented here was designed to solve the problem of finding the shortest path in a bus network. The new algorithm had to be designed in order to meet the special needs of a bus transportation network such as timetables and the possibility of waiting at bus stops. The chief difference between the standard shortest path problem and this one is that here links' costs vary with time and that it is allowed to wait at nodes as long as it is necessary to obtain the minimal time cost. The problem can be classified as the shortest path problem with time dependent costs of links and the allowance of waiting at nodes. A time cost of every link may differ in any desired way. The solution is based on the Dijkstra (1959) algorithm. Dreyfus (1969) proposed an algorithm for networks with time dependent costs of links. The author's algorithm uses principles of the Dreyfus algorithm and extra constraints due to specifics of bus networks.

To understand the problem clearly, it is useful to visualise a user who wants to get from one bus stop to another in a city using buses only. The input data to the algorithm consists of a description of the bus transportation network (timetables, description of connections between bus stops), the bus stop where the journey begins (the source node) and the bus stop at which the journey ends (the destination node). The objective is to find the

shortest path between the two specified nodes, namely the path that requires the minimal amount of time.

The Dijkstra algorithm has to be modified because of two problems.

The first modification deals with a problem of fixed times at which a bus leaves a bus stop. This new attribute of a link is going to be named the departure time of a link. The Dijkstra algorithm is not concerned with a departure time of a link; the algorithm was designed to work with links which can be used at any time. In our problem the main constraint is that links cannot be used at any time, the time at which a bus link can be used is fixed according to a timetable.

The second problem is the actual cost of a bus connection between two nodes. In our case the cost of a link is not the criterion to judge the optimality of the link choice anymore. In the present problem the actual criterion is the sum of the waiting time and the link cost, or, in other words, the time of arrival at the finishing node of a link. In effect, we are also concerned with the waiting times at nodes. The need to wait at bus stops is a consequence of the departure time attribute (if a link cannot be used right now then it is necessary to wait for the departure).

Figures 4.2 and 4.3 depict the problem of the overall cost of a link. Suppose there are buses leaving a specific bus stop at 1, 2, …, 10 time units and arriving at the other specific bus stop after the time cost as shown in the figure 4.2. The time costs of the buses may differ considerably (as in the fig. 4.2) since the buses may be of different companies and they may take different routes. Having the data (presented in the figure 4.2), the task is to find the cheapest connection between two nodes. The task then is to find the link that has the minimum sum of the time cost of a link and the waiting time (that is necessary to wait for this link). For example, the cost of a link at the 5th time unit is 4 time units and therefore the destination is reached after 9 time units. This link is evidently the best choice. It makes no sense to take, for example, the first bus leaving at the 1st time unit and to travel at the cost of 18 time units since the overall time cost of the travel would be 19 time units.

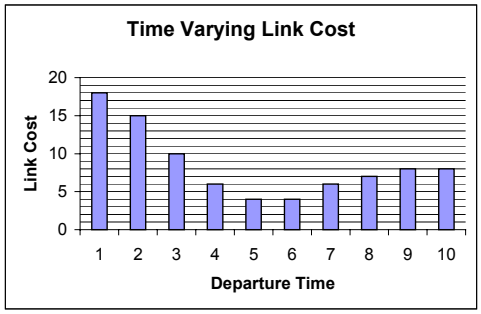

Fig. 4.2. Time varying link cost.

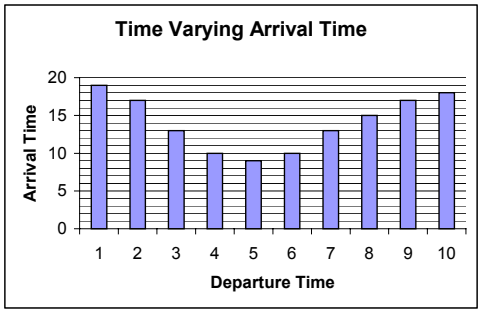

Fig. 4.3. Time varying arrival time.

Figure 4.3 shows the arrival times (link costs + departure times). Figure 4.3 is enough to find the minimal choice. The minimum is at the 5^{th} time unit. The Minimum is 9 time units.

We can phrase the solution to the problem in this way: the sought link is the link of which the arrival time is minimal. This formulation of the solution, i.e. finding the minimal arrival time, is going to be used as opposed to the summation of a waiting time and the time cost (which is the same but complicates the coming formulas).

4.5.1 The Model of a Bus Transportation Network

We need to represent a bus transportation network to model time tables. Therefore we are going to put a link into the directed graph to represent a connection between two bus stops. We need to associate a bus departure time from a bus stop, this departure time represent one entry in a timetable. We are also going to need the time of arrival to the next bus stop (this can be taken from the bus table of the next node). Each entry in a timetable will be represented by one link and therefore we need to operate on sets, not on matrices.

Table 4.1. A bus connection between two parts of Nottingham: the City Centre and Clifton.

Time of departure form the City Centre	**Time of arrival to Clifton**
7:10am	7:30am
12:35pm	13:00pm
17:20pm	17:50pm
22:50pm	23:05pm

A bus connection from table 4.1 is described by a subset of links from table 4.2.

Table 4.2. Representation of a bus connection from the table 4.1 in the form of links.

Starting node	**Finishing node**	**Departure time**	**Arrival time**
City Centre bus stop	Clifton bus stop	7:10am	7:30am
City Centre bus stop	Clifton bus stop	12:35pm	13:00pm
City Centre bus stop	Clifton bus stop	17:20pm	17:50pm
City Centre bus stop	Clifton bus stop	22:50pm	23:05pm

4.5.2 Definitions

We introduce new definitions and concepts to model the bus network conveniently and to describe the algorithm precisely.

Definition of the Network

We are given a directed graph $G(V, E, d, a)$, V is the set of nodes (Vertices) and E is the set of links (Edges). We denote the number of element of the set V of nodes as $n = |V|$ and the number of links of the set of E as m = |E|. We will refer to a node of a graph by 'x_j' and to a link by '*e*'. Function $d : E \mapsto R \cup \{\infty\}$ (the infinity can be ascribed to a link, therefore it is included) associates with every link a departure time (a real nonnegative number) from the staring node and the function $a : E \mapsto R \cup \{\infty\}$ ascribes the time of arrival (a real nonnegative number) at the finishing node. The departure time $d(x_j)$ of the x_j link (from the staring node) is the time at which the link may be used, it cannot be in use at any other time. The arrival time $a(x_j)$ of the x_j link is the time at which we get to the finishing node of the link. If it is

necessary to know the time cost of the x_j link (time required to traverse the link), we can calculate it in this way: $c(x_j) = a(x_j) - d(x_j)$.

We can describe uniquely a link of the graph only by four numbers $(x_i, x_{i+1}, d(x_i), a(x_i))$ where x_i is a starting node of the link, x_{i+1} is the finishing node of the link, $d(x_i)$ is the departure time and $a(x_i)$ is the arrival time. Referring to a specific link only by a pair (x_i, x_{i+1}) can be ambiguous since there can be two or more links between the x_i *and* x_{i+1} nodes, links may differ in the departure time or the arrival time.

The ℙ Symbol

The symbol $\mathbb{P}X$ represents a powerset of set X (i.e. the set of all subsets of the set X).

Definition of the slbtn Function

s*lbtn* : $V \times V \mapsto \mathbb{P}E$

Set of Links Between Two Nodes

The *slbtn* function takes a pair of nodes (x_i, x_{i+1}) and returns the set which comprises every link that starts at x_i node and finishes at the x_{i+1} node.

Definition of the slcsn Function

slcsn: $V \mapsto \mathbb{P}E$

Set of Links of Common Starting Node

The function returns all the links that have the node given as the argument as the starting node.

Definition of a Path

Let $P = (s = x_1, x_2, \ldots, x = x_r)$ be a path form the *s* node to the *x* node. The s node is the source node of the path and the x node is the destination node of the path.

Definition of the Arrival Time at a Specific Node

$T(x_i)$ is the time of arrival at the node x_i. Let t_0 be the time at which the journey starts. We define $T(x_i)$ recursively. The starting node of the shortest path is denoted by *s*.

$$T(s) = t_0$$

$$T(x_{i+1}) = \min_{\{e \in slbtn(x_i, x_{i+1}) \mid d(e) \geq T(x_i)\}} a(e) \tag{1}$$

Note that $T(x_i)$ is not the cost of the shortest path. It represents the time at which the node is reached. The cost of getting to a node let's say x_{17} may be only ten minutes, but $T(x_{17}) = 17{:}10$ if we started the journey at 17:00.

Definition of the Shortest Path

Let $P = (s = x_1, x_2, \ldots, x = x_r)$ be the shortest path from the source node s to the destination node *x*. The nodes of the shortest path are such that they result in the minimal arrival time to the x_r node. The time of arrival is given by $T(x_r)$ from (1).

The subsequent nodes of the shortest path are found by the use of $T(x_i)$ function from (1). To find x_{r-1} we have to find the link which led to x_r with minimal arrival time. Having the link, we have the starting node of the link. This starting node is the x_{r-1} node of the shortest path. This method has to be repeated until the source node is reached.

Definitions of the SPN, SSN, SNRN

Before we get to the algorithm description, there are some definitions to be introduced. We classify all the nodes of the graph into three sets, every node can be a member of only one of the following sets:

SPN: the Set of Permanent Nodes is the set of nodes which have been completely processed, the time of reaching these nodes has been computed and will not change;

SSN: the Set of Scanned Nodes is the set of nodes which have been reached, but have not been completely processed, the time cost of getting to them is known but may change;

SNRN: the Set of Not Reached Nodes is the set of nodes which have not been reached at all.

4.5.3 The Step Description of the Algorithm

The aim of the algorithm is to minimise $T(x)$ (x is the destination node). To minimise it we first have to minimise $T(x_i)$ for nodes x_i which are at the shortest path from s to x.

Step 0

The source node s is initialised as *scanned* ($s \in SSN$) and every other node $x_i \neq s$ of the graph is initialised as *not reached* ($x_i \in SNRN$). Furthermore, the arrival time of the source node is set to t_0 (t_0 is the time at which the journey starts), i.e. $T(s) = t_0$, and the arrival time of every other node $x_i \neq s$ of the graph is set to infinity, i.e. $T(x_i) = \infty$.

Step k

We process only one node during this step. We choose the node to be processed from the SSN (Set of Scanned Nodes). If the SSN is empty, this means there is no path between the source node and the destination nodes and the algorithm quits. If the SSN is not empty, we choose a x_i node from the SSN which has minimal $T(x_i)$. If there is more then one node with the minimal $T(x_i)$ then we choose one of them arbitrarily. Expression (2) describes the choice.

$$x_i \in SSN \bullet T(x_i) = \min_{x_j \in SSN} T(x_j) \tag{2}$$

Once the x_i node is chosen, we proceed to process it. First the node x_i is excluded from the SSN and becomes a member of SPN (Set of Permanent Nodes). At this stage it is certain that the arrival time $T(x_i)$ is minimal (it may only get larger since taking another link will increase the cost; link costs are always positive) and this is the reason for moving the x_i node to SPN.

Next the links leaving the x_i node are handled. From the set E (the set of links of the graph) every link which has the x_i node as a starting one is selected. The retrieved set is further constrained to links of the departure time greater or equal to $T(x_i)$ (only buses that will arrive to a bus stop can be taken, not those which have left). Formula (3) gives the restricted set.

$$\alpha(x_i) = \{e \in sllsn(x_i) \mid T(x_i) \leq d(e)\} \tag{3}$$

The empty $\alpha(x_i)$ set denotes there are no buses leaving the x_i node after $T(x_i)$ or there are no buses at all.

The set $\alpha(x_i)$ is divided into subsets, each subset having links of a distinct finishing x_{i+1} node. In consequence, all subsets have the same starting x_i node and each subset has a unique finishing x_{i+1} node. Therefore, for example, if it is possible to get from the x_i node to other, let's say, 3 nodes, that means there are going to be exactly 3 subsets.

For each subset we examine if the finishing node x_{i+1} of the subset is in the SPN. The statement $x_{i+1} \in SPN$ implies that the minimal value $T(x_{i+1})$ has been found and we can proceed to process a next subset. If $x_{i+1} \notin SPN$, we have to find in the subset a link e for which $a(e)$ is minimal. If it occurs that the value $a(e)$ is smaller then $T(x_i)$ previously found, i.e. $a(e)<T(x_{i+1})$, then the new value of $T(x_{i+1})$ is set $T(x_{i+1}) = a(e)$. At the end of processing each subset we make sure that the x_{i+1} node is a member of SSN (if it is not, it has to be moved to the SSN).

4.5.4 The Bus Routing Algorithm Implementation Using a Pseudo Code

Let *p_queue* be a priority queue of nodes. The order of nodes x_j in the queue is pointed out by the increasing values of $T(x_j)$, so that at the beginning of the queue there is a node which has the smallest $T(x_j)$.

$p\_queue = \emptyset$
for every node $x_k \in V$ set the value $T(x_k) = \infty$
set value $T(s)=t_o$
push s into the priority queue *p_queue*
while($p\_queue \neq \emptyset$)
    x_i = get_first(*p_queue*)
        for every $x_{i+1} \in V$ such that $(x_i, x_{i+1}) \in V$
            calculate:

$$T = \min_{\{e \in slbtn(x_i, x_{i+1}) \mid d(e) \geq T(x_i)\}} a(e)$$

            if $T<T(x_{i+1})$ then
                if $T(x_{i+1}) = \infty$ then
                    push x_{i+1} into the priority queue *p_queue*
                $T(x_{i+1}) = T$

4.6 Combination of the Algorithms

The explained bus algorithm can be used separately. It is complete and ready to use. However it can be combined with other algorithms to be used for different purposes. In this section we will have a look at the use of the algorithm in a combination with a Dijkstra algorithm.

The problem is straightforward: to find the shortest route from home to work using both: a car and a mean of public transportation. At the beginning of the journey the user takes a car. He can reach work by car only but it may happen that using a bus he will get to work in a shorter time. Changing to a bus may shorten the time because buses use separate lanes during the rush hour.

There are a number of constraints concerning car to bus changes. The fundamental one is that the user can leave the car at the Park & Ride locations and nowhere else. There is no restriction to the number of bus changes. Therefore every route from home to work looks as shown in Fig. 4.1.

On the way from home to work the user can go directly to work or can change to a bus. The change has to be made at a Park & Ride location. After the change the user can take any number of buses to finally reach work.

Conversely, on the way from work to home the user can take a car and make his way straight home but he/she is left with the choice to take any number of buses to reach the Park & Ride location and to go home finally from there.

The way the combined path algorithm calculates the journey from home to work is very intuitive. First the shortest path algorithm for the car links is run. The algorithm is called to find the shortest route to work and all Park & Ride locations. Next the algorithm for buses is run for each Park & Ride location to find the shortest path from this Park & Ride location to work. In this way we get several shortest paths: straight from home to work, and from all Park & Ride locations to work. From these paths the best path can be chosen.

In the same way the algorithm from work to home works. First the algorithm for buses is called to find the shortest route to each Park & Ride and home. Next the shortest path algorithm for a car is called to find the shortest paths from work to home and from work to all Park & Rides.

5 RESULTS OF THE TRAFFIC GAME USER EVALUATION

A survey to evaluate the Traffic Game was carried out during the Faculty Open Day on the Nottingham Trent University (9th March 2000). The game was presented to the evaluators, who afterwards with the support of questionnaires assessed the software.

Questions covered the whole project, however the major part related to the user interface. Since the user interface is the part of the system that directly interacts with the user, it has the biggest influence on the fact of whether he likes the system or not.

The profile of users who have taken part in the survey was also identified. The purpose of the Open Day was to encourage young people undertaking higher education in the field of engineering to choose Faculty of Electrical Engineering, Electronics and Computer Science at Nottingham Trent University. Let the reader note that those users mostly have some prior experience with computers. There were a significant number of university tutors among the evaluators as well. Because the available evaluators did not exactly match the target user group, the results may seem to be irrelevant. However the received feedback helped to significantly improve the game. This is due to the evaluators' experience with computer applications. Young users have experience with various computer games, thus their evaluation is expected to cover the level of entertainment the game provides. Since the entertainment is not the most important purpose of the game, the academic tutors' experience in designing and developing systems is hoped to provide valuable and mature feedback.

The questions that participants were asked after playing the game were as follows:

1) *How interesting and fun did you find the game? (Very, Quite, Not very)*
2) *How aesthetically appealing did you find the Graphics User Interface? (Very, Quite, Not very)*
3) *Are you convinced that the shortest route was found? (Yes, No)*
4) *How easy did you find the program to use? (Very, Quite, Not very)*
5) *What did you like the most in the game? (The idea of the game, The user interface, The graphics)*
6) *What would you change or add to the game to improve it?*
7) *Please grade the program. (A,B,C,D,E)*

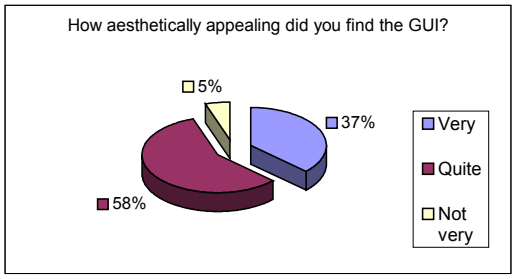

Fig 5.1 How aesthetically appealing did you find the GUI.

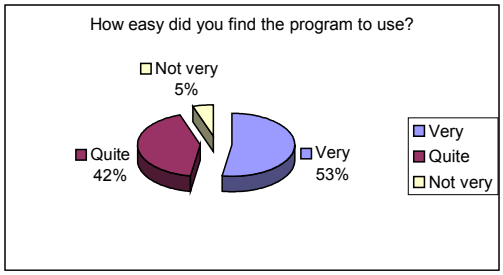

Fig5.2 How easy did you find the program to use.

The GUI design was well accepted by the evaluators, what can be concluded from Fig 5.1. The general idea of the software itself is not complex, but in addition to well designed user interface creates a simple in use product, what can be concluded from Fig 5.2.
The users suggested the following changes in order to improve the quality of the project. Below some of the issues considering the possible game improvements are highlighted.

Some of the suggestions from participants were not detailed enough and therefore could not be taken under consideration and implemented. They highlighted better graphics and interaction as well as enhancement of the visual input. Such general ideas can be implemented in many ways therefore they were not considered to be relevant. However the existence of such suggestions imply that something must be done towards meeting the mentioned remarks and thus additional analysis could be performed in the future.

The clear remarks generally pointed out two areas for potential improvement such as the quality of map and user interface.

The remarks concerning the map pointed out mainly its low quality. In order to become readable the map was enlarged and thus lost its quality. The better quality map was delivered after the evaluation and was implemented in the Traffic Game. The new map was obtained from a professional company (Geographers' A-Z Map Company Ltd) dealing with cartography thus its quality as well as detail level are very high. However in order to introduce the map into the software some memory management issues had to be addressed since it demands much more storage space than the previous one. The new map covers a bigger area – another issue addressed by one of the evaluators. Such facilities would however demand bigger computational power and memory resources. It would drastically decrease the application's platform independence.

The suggestions dealing directly with the user interface of the game considered mainly making it easy and "more handy" to use. The prototype put forward for evaluation indeed had some problems with accepting commands on the map (mouse clicking). Since the map was small the users had problems indicating the directions (links) they wanted to take next. The area of mouse sensitivity was indeed very small and was not indicated by any change of mouse pointer or colour. Subsequently the area of mouse sensitivity was increased from a node to the whole link.

The other remark taken into consideration was to introduce default values in the entries. The designer considered reducing the number of questions the user has to answer before playing the game. For instance, the question of user identity was placed at the end of the game and will be asked only if the user's score is good enough to be put into the Hall of Fame.

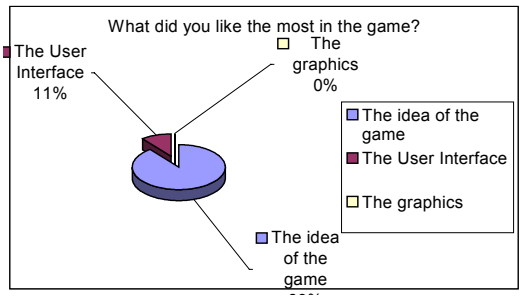

Fig 5.3 What did you like the most in the game.

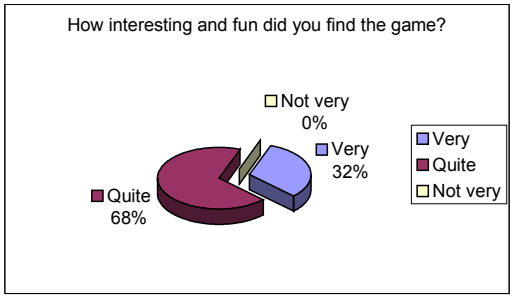

Fig 5.4 How interesting and fun did you find the game.

As it can be drawn from the Fig 5.3 the idea of the game was quite well welcomed by the users. A similar conclusion can be drawn from Fig 5.4. This implies that such a piece of software will be probably well accepted by the potential users.

6. CONCLUSIONS

The project proved to be a successful and fully working application. Several new concepts were incorporated into it and their impact on the application was very positive.

The three-layer made it very comfortable to work with and there was a great deal of naturalness in visualisation of the cartographic elements. Additionally the concept of the object-oriented database turned out to be well suited for this particular project. Obviously possibilities in further improvement of this approach still remain.

The algorithms, which were based on the previous research in the field of network optimisation, successfully addressed the problem of finding the shortest communication path and the issue of environment preservation in the modern city.

The project proved to be successful in terms of user evaluation as well. The results of the survey imply that the idea of the game as well as the Graphical User Interface was especially welcomed by the users. This gives the future development of the Traffic Game a green light. Let the reader note that being well supported by the previous research conducted at the Nottingham Trent University, this piece of software is a platform for further research and development in the field of environmental issues within urban communication. With the increasing number of cars on the roads and frustrated drivers, applications such as the Traffic Game incorporating entertainment and environmental education can make a difference.

References and Bibliography

Alagic, S., (1988), Object-Oriented Database Programming, Springer-Verlag.

Hughes, J. G., (1991), Object-Oriented Databases, Prentice Hall.

Khoshafian, S., (1993), Object-Oriented Databases, Wiley.

Kozielski, S., (1998/99), Databases, Lectures, Silesian Technical University, Gliwice, Poland.

Peytchev, E. T., (1999), Integrative Framework for Discrete Systems Simulation and Monitoring, Nottingham Trent University - PhD Dissertation.

Stevens, P., Pooley, R., (1999), Using UML, Addison-Wesley.

Voisard, A., (1998), Spatial Databases, Application to GIS [online], Available at: <http://www.inf.fu-berlin.de/~voisard/Teaching/Spatial/index.html> [Accessed 19.12.1999].

Dijkstra, E. W., (1959), A Note on Two Problems in Connection with Graphs, Numerishe Mathematic 1, 269-271.

Dreyfus, S. E., (1969), An Appraisal of Some Shortest-Path Algorithms, Operations Research 17, 395-412.

Gardiner, M.M., Christie, B., (eds.), (1987), Applying Cognitive Psychology to User Interface Design, John Wiley & Sons Ltd.

Faulkner, C., (1998), The Essence of Human-Computer Interaction, Prentice Hall.

Redmond-Pyle D., Moore A., (1995) Graphical User Interface Design and Evaluation. A Practical Process, Prentice Hall.

Sommerville, I. (1995), Software Engineering (5th ed.), Addison-Wesley.