SUDAN UNIVERSITY OF SCIENCE & TECHNOLOGY

COLLEGE OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

COMPUTER SCIENCE DEPARTMENT

INTELLIGENT SYSTEM FOR ROUTING AN AMBULANCE

THE DISSERTATION SUBMITTED AS A PARTIAL FULFILLMENT FOR THE REQUIREMENT OF BSC (HONOUR) DEGREE IN COMPUTER SCIENCE

October 2015

بسم الله الرحمن الرحيم

SUDAN UNIVERSITY OF SCIENCE & TECHNOLOGY

COLLEGE OF COMPUTER SCIENCE & INFORMATION TECHNOLOGY

COMPUTER SCIENCE DEPARTMENT

INTELLIGENT SYSTEM FOR ROUTING AN AMBULANCE

October 2015

PREPARED BY: **SUPERVISOR:**

Ahmed Mohammed Alkhair Abdalgadir **Dr. Hoida Ali**

Jihad Fayez Abd Elmajed **SIGNITURE: ……………** **DATE: ……/………../…………….**

الآية

قال تعالى﴿ اللّهُ لاَ إِلَـهَ إِلاَّ هُوَ الْحَيُّ الْقَيُّومُ لاَ تَأْخُذُهُ سِنَةٌ وَلاَ نَوْمٌ لَّهُ مَا فِي السَّمَاوَاتِ وَمَا فِي الأَرْضِ مَن ذَا الَّذِي يَشْفَعُ عِنْدَهُ إِلاَّ بِإِذْنِهِ يَعْلَمُ مَا بَيْنَ أَيْدِيهِمْ وَمَا خَلْفَهُمْ وَلاَ يُحِيطُونَ بِشَيْءٍ مِّنْ عِلْمِهِ إِلاَّ بِمَا شَاء وَسِعَ كُرْسِيُّهُ السَّمَاوَاتِ وَالأَرْضَ وَلاَ يَؤُودُهُ حِفْظُهُمَا وَهُوَ الْعَلِيُّ الْعَظِيمُ ﴾

البقرة الآية ﴿255﴾

الحمد

الحمـد للـه بعـدد كلمـاته الـتي لا تنفـذ الحمـد للـه بسـعة علمـه الـذي لا ينفـذ الحمد لله منذ ان كان وحده ولم يكن سواه احد ، الحمد لله منذ ان خلق القلـم وخلـق السموات والأرض ، الحمد لله حين أسـتوى علـى العـرش ، الحمـد للـه حيـن خلـق آدم وسواه وكرمه على كثير مما خلق ، الحمد للـه الـذي علمـه الأسـماء وخلـق لـه حـواء ، الحمد لله الذي أمر الملائكة بالسجود لـه ، الحمـد للـه الـذي علمـه التوبـة فتـاب عليـه الحمد لله الذي جعله خليفة في الأرض ، لك الحمد يا لله بالأيمان ، ولك الحمـد بالأهـل والمال والمعافاة ، بعثت فينا افضل انبيائك بأفضل كتبك وجعلتنا من افضل الامم فلــك الحمد على ذلك كله.

DEDICATION

For the one who hit the message and led the Secretariat advised the nation to the Prophet of mercy and the light of the Worlds
Prophet Muhammad peace be upon him

To the one who teach me tender without waiting, to carry his name proudly, I ask God to reach at your age to see the fruit picking has come after a long wait and your star will remain guided by today and tomorrow and forever
to my dear father

To the greatest unconditional and infinite love we will ever experience in our existence...
To my dear mother

To my brothers and sisters: For their unconditional love, faith, Understanding, and support**.**

ACKNOWLEDGMENTS

we would like to express our appreciation to our supervisor Dr. Hoida Ali Abdalgadir Ahmed who has carefully answered our queries, provided us with materials, checked our examples, and assisted us in a myriad ways by her helpful comments on every stage and process of this project.

We are also very grateful to our friends, family for their good support and assistance throughout the production of this project.

ABSTRACT

The aim of this study is to find the route for an Ambulance from the hospital - where the Ambulance is located - to the casualty location through the Khartoum City road Network, by displaying the route path to the user of the system.

To study the problem and test the proposed solution the following steps has been performed: Create the Global Positioning System (GPS) data base for Khartoum City road network from the GPS data files, specify the hospital location where the ambulance is located, specify the casualty location, find the route between these two locations, and display the route path on the screen.

We have used SWI-Prolog to find the route path. We have also created a java library (GPXParse) to parse the GPS data file (.gpx) to strings to be used in creating prolog predicates and also to be stored in the data base. Another java library (JPL) has been used to call the prolog program from java and call java from the prolog program.

المستخلص

الهدف من هذه الدراسة ايجاد المسار لسيارة الإسعاف من المستشفى الموجودة بها السيارة الى مكان المصاب (المريض) في شبكة شوارع مدينة الخرطوم ، بعرض المسار على شاشة مستخدم النظام.

لدراسة المشكلة واختبار الحل المقترح تم تطبيق الخطوات التالية: انشاء قاعدة بيانات جغرافية لشبكة شوارع مدينة الخرطوم من ملفات بيانات نظـام تحديـد المواقـع العـالمي ، تحديـد موقـع المستشـفى الـتي توجـد بهـا سـيارة الإسـعاف ، تحديـد موقـع المصاب (المريض) ، ايجاد الطريق بين هذين الموقعين ، وعرض المسـار علـى شاشـة مستخدم النظام**.**

لقد قمنا بإستخدام ال(SWI-Prolog) لإيجاد المسار. كما قمنا بإنشاء مكتبة جافا (GPXParse) لتحويل ملف البيانـات الجغرافيـة (.gpx) الـى سلاسـل مـن الحـروف (strings) ليتم إستخدامها في انشاء فرضـيات الـبرولوق (predicates) وكـذلك فـي إنشاء قاعدة البيانات الجغرافية. كمـا قمنـا بإسـتخدام مكتبـة جافـا اخـرى (JPL) لنـداء برنامج البرولوق من برنامج الجافا والعكس.

TABLE OF TERMS

Term abbreviation	terminology
ISRA	Intelligent System for Routing an

	Ambulance
AI	Artificial Intelligence
TSP	Travelling Salesman Problem
CLP	Constraint Logic Programming
CSP	Constraint Satisfaction Problem
COP	Constraint Optimization Problem
OP	Operation Research
I/O	Input/output
VM	Virtual Machine
GIS	Geographical Information System
GPS	Global Positioning System
IDE	Integrated Development Environment
JDK	Java Development Kit
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet
PHP	Personal Home Page
OOP	Object Oriented Programming

TABLE OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

MOTIVATION 1.1

In today’s traffic world, when an accident occur on the road or emergency situations at home, ambulance is so important to save valuable human life. Transportation of patients to the hospital seems simple but it is quite hard. Khartoum is metropolitan city with rapid increase in population and vehicles, which result in high road density.

PROBLEM STATEMENT 1.2

Metropolitan areas in Sudan facing the problem of high population density which cause many emergency medical casualties. Every year much valuable life lost due to delay in providing medical care via ambulance and the main reason is that the ambulance driver has no clear route to the casualty location. Even the government had provided more road ways and bridges the problem still arises. Most of the emergency hospital ambulances are unable to reach their destination at time because of that reason. It is obvious what happens to the patient till the ambulance reaches? Due to lack in verification sometimes ambulance driver is unable to reach his destination as reported.

PURPOSE/GOAL 1.3

The main objective of this project is to build an expert system for the ambulance routing on Khartoum road network. This Intelligent System for Routing an Ambulance (ISRA) used for solving the routing and location problems:

-i .Define the ambulance location

-ii .Define the casualty location

-iii .Find the route between the two locations

1.4 IMPORTANCE OF THE PROJECT

Our project is useful for all mankind especially those who get serious injuries and need to get immediate medical attention. The time to transfer patients to the hospital is quite vital and seconds are so valuable to save ones life.

So we are looking forward to lives in Sudan by our project Intelligent System for Routing an Ambulance (ISRA).

1.5 SCOPE AND LIMITATION

In our system the dispatch center admin must enter the location of the hospital where the ambulance is locate, then enter the location of the casualty and the program will show him the route.

Our provided routing path doesn’t depend on time period or distance. To select the shortest path we need all traffic signals in that path to be green and that needs another technology to be implemented in the system we leave that as a future challenge.

1.6 RESEARCH STRUCTURE

This research consists of five chapters, the second chapter talks about the research background and previous theoretical studies, the third chapter includes the tools and techniques which are used in the project, the fourth chapter includes the implementation and programming and some scenarios for the system ,and the fifth chapter includes the results and recommendations and conclusions.

CHAPTER 2

LITRETURE REVIEW

CHAPTER 2

LITRETURE REVIEW

2.1 ARTIFICIAL INTELLIGENCE

2.1.1 DEFINITION

Artificial intelligence (AI) is the intelligence exhibited by machines or software. It is an academic field of study which studies the goal of creating intelligence. Major AI researchers and textbooks define this field as "the study and design of intelligent agents".

In which an intelligent agent is a system that perceives its environment and takes actions that maximize its chances of success. John McCarthy, who coined the term in 1955, defines it as "the science and engineering of making intelligent machines". [3]

2.2 TRAVELLING SALESMAN PROBLEM SHORT PATH

2.2.1 HISTORY

The origins of the travelling salesman problem (TSP) are unclear. A handbook for travelling salesmen from 1832 mentions the problem and includes example tours through Germany and Switzerland, but contains no mathematical treatment. The travelling salesman problem was mathematically formulated in the 1800s by the Irish mathematician W. R. Hamilton and by the British mathematician Thomas Kirkman. Hamilton's Game was a recreational puzzle based on finding a Hamiltonian cycle. The general form of the TSP appears to have been first studied by mathematicians during the 1930s in Vienna and at Harvard, notably by Karl Manger, who defines the problem.[7]

2.2.2 DEFINITION

TSP is a special case of the travelling purchaser problem. In the theory of computational complexity, the decision version of the TSP (where, given a length L, the task is to decide whether the graph has any tour shorter than L) belongs to the class of NP-complete problems. Thus, it is possible that the worst case running time for any algorithm for the TSP increases super polynomial with the number of cities. Starting at a home city, a traveling salesman must visit several cities and then return home. The distance between every city pair is specified and the salesman is to visit each city once and only once.

2.3 CONSTRAINT LOGIC PROGRAMMING

2.3.1 DEFINITION

Constraint Logic Programming (CLP) is a tool for solving constraint satisfaction problem (CSP). CSP is characterized by the following features:

- A finite set S of integer variables X1, …, Xn, with values ;from finite domains D1, …, Dn
- .A set of constraints between variables
- A CSP solution is given by any assignment of domain values to variables that satisfies all constraints. It may .non-unique or unique
- A CSP solution may additionally minimize or maximize an objective function. Then it is usually refers to constraint optimization problem (COP), and its solution as optimum .solution

2.3.2 THE REASON BEHIND CLP

A salient feature of CSP and COP is that all variables take values from finite domains. In theory any CSP and COP can either have no solution or be solved using an algorithmically simple exhaustive search or direct enumeration approach.

So we use it to solving our problem because CLP provides very good tools in our research domain.

2.3.3 CONSTRAINTS CONCEPT

It is understood to mean any-thing that limits the freedom of action. However their meaning in imperative languages (like Pascal, C, C ++) differs considerably from their meaning in CLP languages.

In imperative languages constraints are passive; that means they may be used only if all their variables are grounded, and they are used as tests for choosing the next step taken.

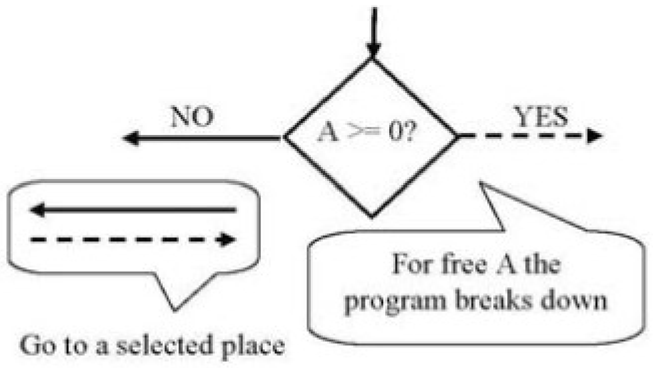


Figure 2.1: A passive Constraint example

Constraints in CLP languages are active; that means they may be used also if some or all their variables are free. Active constraints (denoted by various symbols like # for finite domains or $ for real or symbolic domains) are used for initiating a search for such variable groundings that satisfies them. [4]

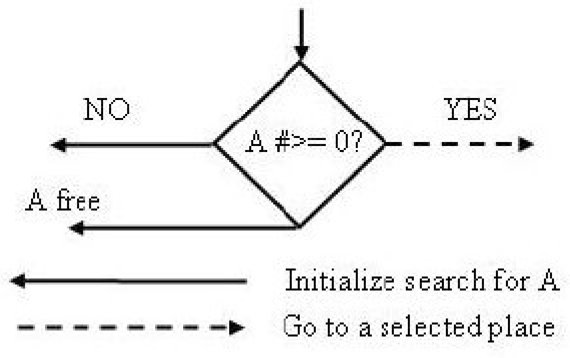


Figure 2.2: An active constraint example

2.3.4 CLP AND ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) is usually understood to be this branch of computer science that deals with creating tools for jobs usually considered to need considerable human intelligence. Solving these kinds of problems manually can put a high demand on the intelligence of humans doing it, because they need to take into account a huge numbers of relations, conflicting factors, and trade-offs.

Researchers, designers and users of AI products have always been confronted with the need to solve difficult complex problems. Exactly the same problems are solved using constraint programming technology.

2.3.5 CLP AND OPERATIONS RESEARCH

Operations Research (OR) is a discipline that aims to calculate optimum or sub-optimum solutions to complex decision-making problems, characterized by some clearly defined objective function and limited resources. It is basically concerned with optimizing the objective function which depends upon some decision variables that can be manipulated to achieve the aim.

2.4 SWI-PROLOG

2.4.1 SWI-PROLOG HISTORY

SWI-Prolog started back in 1986 with the requirement for a Prolog that could handle recursive interaction with the C-language: Prolog calling C and C calling Prolog recursively. In those days Prolog systems were not very aware of their environment and we needed such a system to support interactive applications. Since then, SWI-Prolog's development has been guided by requests from the user community, especially focusing on (in arbitrary order) interaction with the environment, scalability, (I/O) performance, standard compliance, teaching and the program development environment. SWI-Prolog is based on a simple Prolog virtual machine called ZIP [Bowen et al., 1983] which defines only 7 instructions. Prolog can easily be compiled into this language, and the abstract machine code is easily decompiled back into Prolog. As it is also possible to wire a standard 4-port debugger in the virtual machine, there is no need for a distinction between compiled and interpreted code. Besides simplifying the design of the Prolog system itself, this approach has advantages for program development: the compiler is simple and fast, the user does not have to decide in advance whether debugging is required, and the system only runs slightly slower in debug mode compared to normal execution. The price we have to pay is some performance degradation (taking out the debugger from the VM interpreter improves performance by about 20%) and somewhat additional memory usage to help the compiler and debugger.

SWI-Prolog extends the minimal set of instructions described in [Bowen et al., 1983] to improve performance. While extending this set, care has been taken to maintain the advantages of recompilation and tracing of compiled code. The extensions include specialized instructions for unification, predicate invocation, some frequently used built-in predicates, arithmetic, and control (;/2, |/2), if-then (->/2) and negation-by-failure (\+/1).

2.4.2 SWI-Prolog positioning

Most implementations of the Prolog language are designed to serve a limited set of use cases. SWI-Prolog is no exception to this rule. SWI-Prolog positions itself primarily as a Prolog environment for ‘programming in the large’ and use cases where it plays a central role in an application. At the same time, SWI-Prolog aims at providing a productive rapid prototyping environment. Its orientation towards programming in the large is backed up by scalability, compiler speed, program structuring (modules), support for multithreading to accommodate servers, Unicode and interfaces to a large number of document formats, protocols and programming languages. Prototyping is facilitated by good development tools, both for command line usage as for usage with graphical development tools. Demand loading of predicates from the library and a ‘make’ facility avoids the requirement for using declarations and reduces typing.

SWI-Prolog is traditionally strong in education because it is free and portable, but also because of its compatibility with textbooks and its easy-to-use environment.

2.4.3 THE REASONS BEHIND SWI-PROLOG

There are numbers of reasons motivate us to use SWI-Prolog as implementation language such as:

- nice environment

- SWI-Prolog provides a good command line environment, including ‘Do What I Mean’, auto completion, history and a .tracer that operates on single key strokes
- Fast compiler
- Even very large applications can be loaded in seconds on most machines. If this is not enough, there is the Quick .Load Format
- Transparent compiled code
- SWI-Prolog compiled code can be treated just as interpreted code, implies you do not have to decide beforehand whether a module should be loaded for debugging or not, and the performance of debugged code .is close to that of normal operation
- Source level debugger
- The source level debugger provides a good overview of your current location in the search tree, variable bindings, .your source code and open choice points
- Profiling
- SWI-Prolog offers an execution profiler with either textual output or graphical output. Finding and improving hotspots .in a Prolog program may result in huge speedups
- Flexibility
- SWI-Prolog can easily be integrated with C, supporting non-determinism in Prolog calling C as well as C calling Prolog. It can also be embedded in external programs. System predicates can be redefined locally to provide .compatibility with other Prolog systems
- Threads

- Robust support for multiple threads may improve performance and is a key enabling factor for deploying .Prolog in server applications
- Interfaces
- SWI-Prolog ships with many extension packages that provide robust interfaces to processes, encryption, TCP/IP, TIPC, ODBC, SGML/XML/HTML, RDF, HTTP, graphics and .much more

2.5 Previous Studies

In this section we will mention about previous works which have motivated us in implementing this project. In ‘Ambulance Management System using GIS’ by student Linkoping University in Sweden, we see that he used GPS and GSM to find the accident location and locates the nearest hospital, also he used VBA, ArcGIS (network analyst) to develop the user interfaces.

In our system we will use GPS data to create road network data base using MySQL, and we will design our interfaces using Java (NetBeans) because we want our interfaces to be more user friendly than interfaces in the mentioned project

CHAPTER 3

REQUIREMENTS AND ANALYSIS

CHAPTER 3

REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENTS

3.1.1 USER REQUIREMENTS

1. Enter GPS data file to create GPS database.
2. Enter caller location.
3. Enter hospital location where ambulance is located in.
4. Find the route and display routing path in the screen.

3.1.2 SYSTEM REQUIREMENTS

1. Caller location must be string.
2. GPS data file must be .gpx file.
3. Hospital location must be string.
4. Both caller and hospital locations should be drop down menu to avoid user data entering errors.

3.1.3 NON-FUNCTIONAL REQUIREMENTS

1. Ease of use:
   i. A friendly Graphical User Interface.
   ii. Does not need training to interact with the system.
2. Decrease the ratio of system failure probability.

Reliability of the system depends on how accurate does the caller give his full address.

3.2 ANALYSIS

3.2.1 SYSTEM ENVIRONMENT

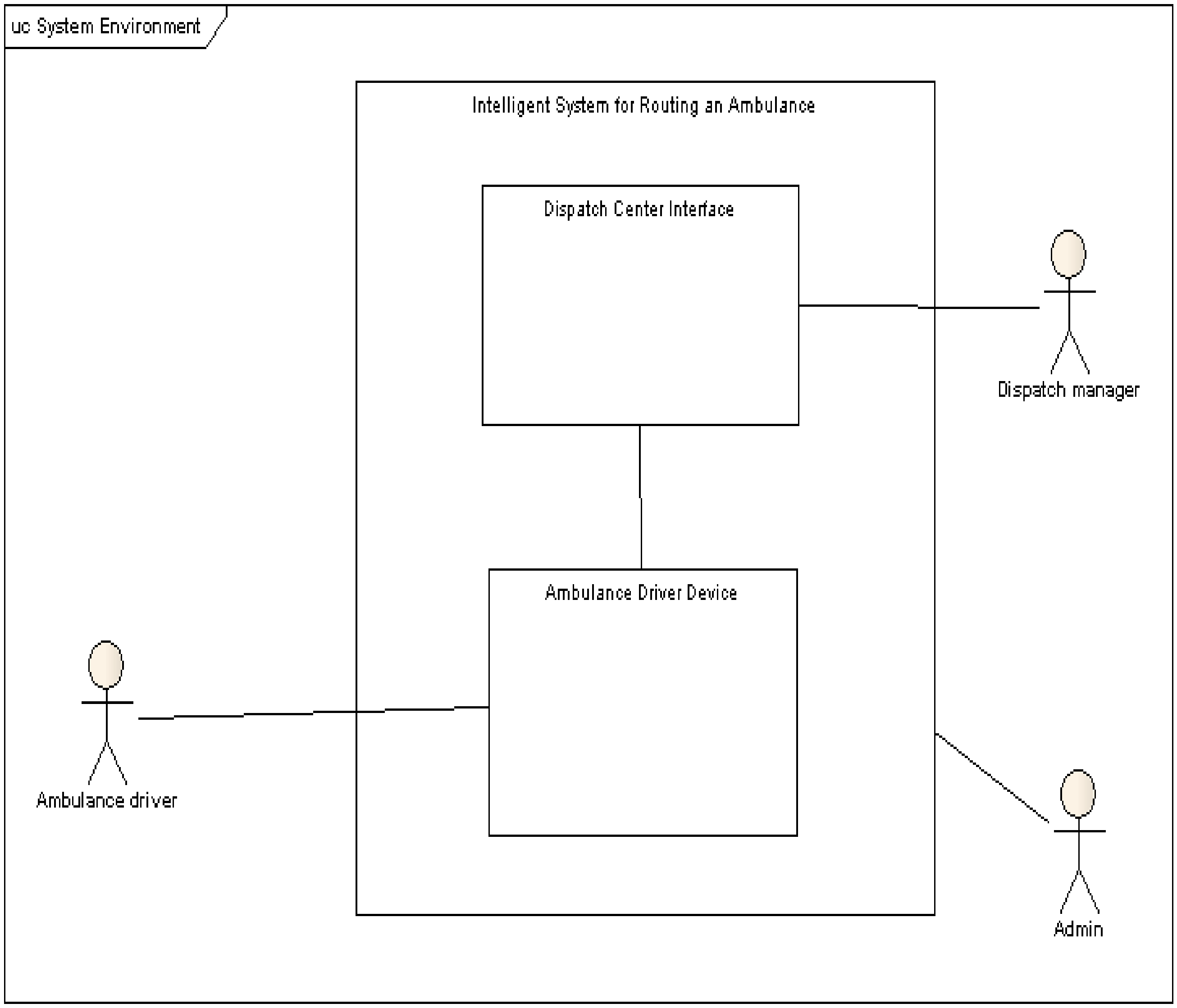


Figure 3.1: System Environment.

3.2.2 SYSTEM FLOW CHART

3.2.2.1 CREATING DATA FLOW CHART

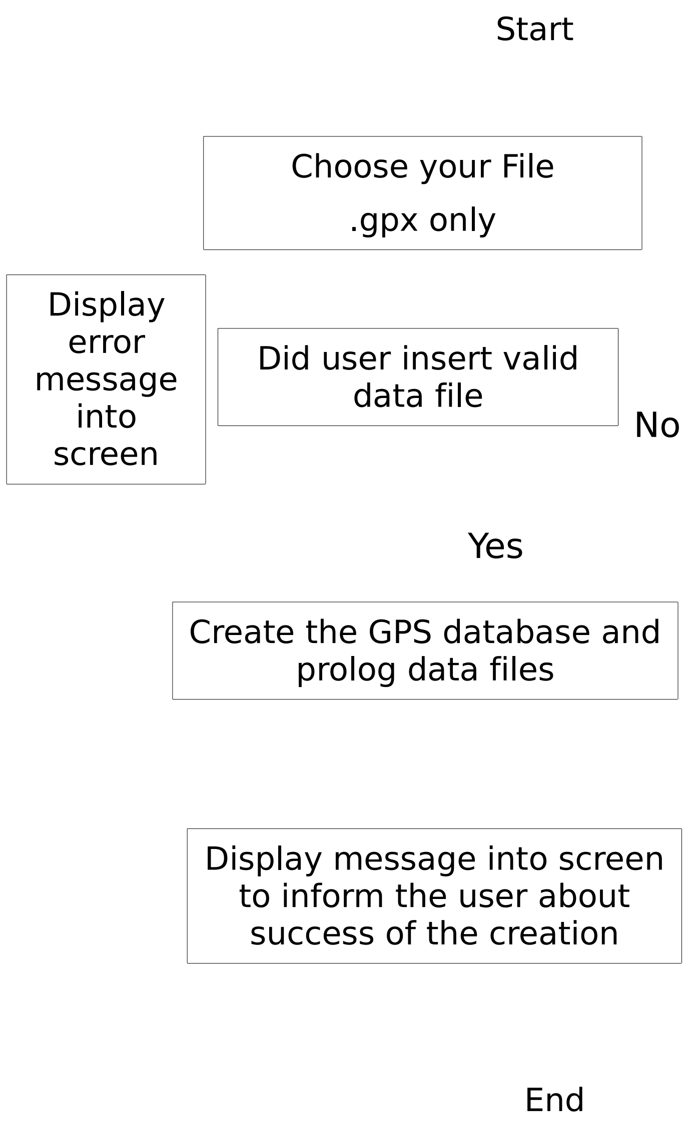
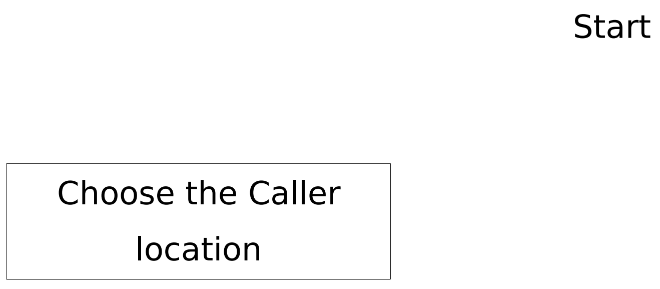

Start
Choose your File .gpx only
Display error message into screen
Did user insert valid data file
No
Yes
Create the GPS database and prolog data files
Display message into screen to inform the user about success of the creation
End

Figure 3.2: creating data flow chart

3.2.2.2 FINDING ROUTE FLOW CHART



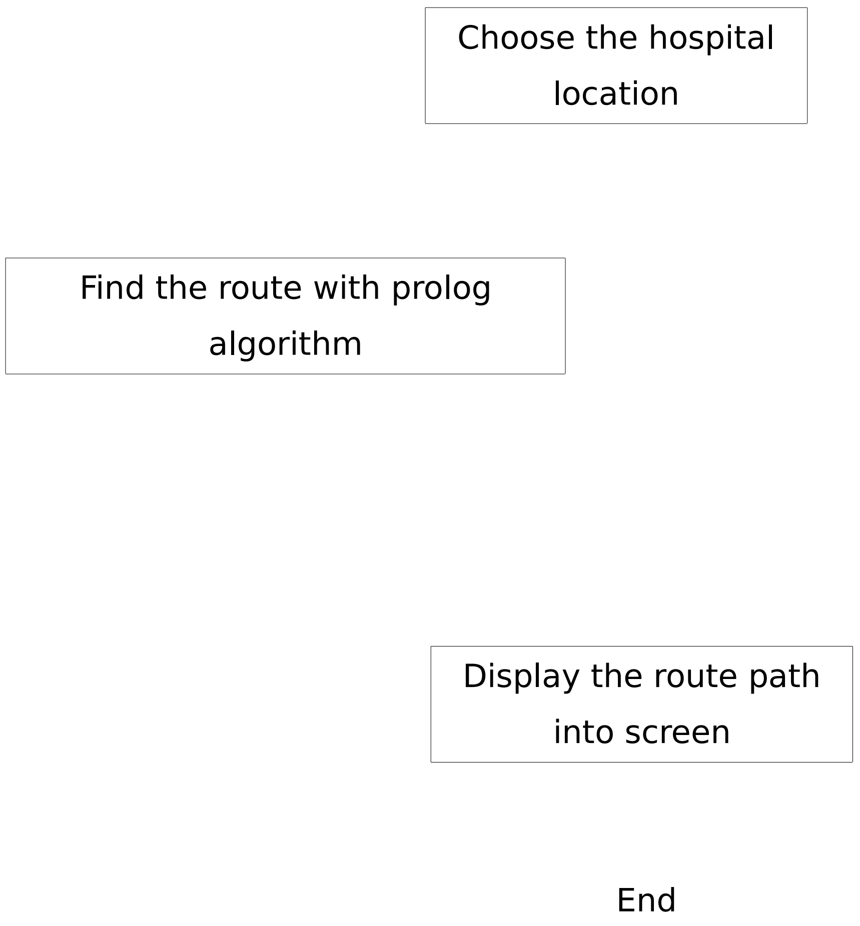

Choose the hospital location
Find the route with prolog algorithm
Display the route path into screen
End

Figure 3.3: Finding route flow chart

3.3 LANGUAGES USED

3.3.1 SWI-PROLOG

SWI-prolog is an advance edition of Prolog, it provides full support of constraint logic programming which is our study area.

We use version 6.6.6 because it was the last stable version at project implementation time, it has its own terminal to interact with it. So we have done prolog code with that terminal running in windows platform.

We used SWI-Prolog because it has very fast compiler which is needed in our project because time is critical factor in our system, and because its flexibility easily can be embedded in external programs, beside it support multiple threads which can improve performance, and it ships with many extension packages that provide robust interfaces to processes, encryption, graphics and much more.

3.3.2 NETBEANS IDE AND JAVA JDK

NetBeans IDE is the best tool to easily develop Java desktop, mobile and web applications, as well as HTML5 applications with HTML, JavaScript, and CSS. The IDE also provides a great set of tools for PHP and C/C++ developers. It is free and open source and has a large community of users and developers around the world.

We used it because it provides fast and smart code editing, besides providing easy and efficient project management and it supports rapid user interface development.

Java is an Object Oriented Programming (OOP) language provides full support to build desktop, mobile and web applications.

We used java because it supports our study area with very good functionality and reliability, as it also provides great libraries in our domain.

CHAPTER 4

IMPLEMENTATION

CHAPTER 4

IMPLEMENTATION

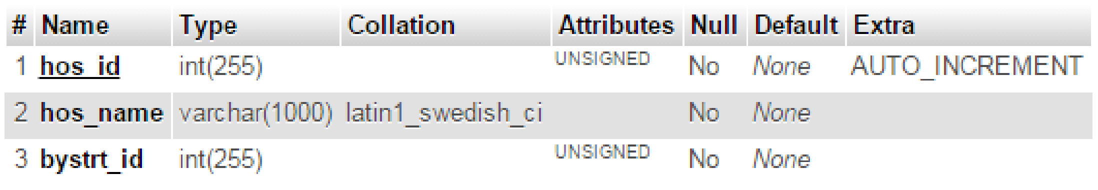
4.1 Introduction:

This Chapter describes how we build our system and configure it, it is also shows the final screens and the functionality of the system as follows:

- Download and install Java JDK 8, NetBeans IDE 6.9M1 and SWI-Prolog 6.6.6.
- Screens were designed by using Java Interface Builder which in NetBeans.
- Download and install WAMPSERVER 2.5.
- Create database tables and create the relations between them from phpMyAdmin.

4.2 The Database

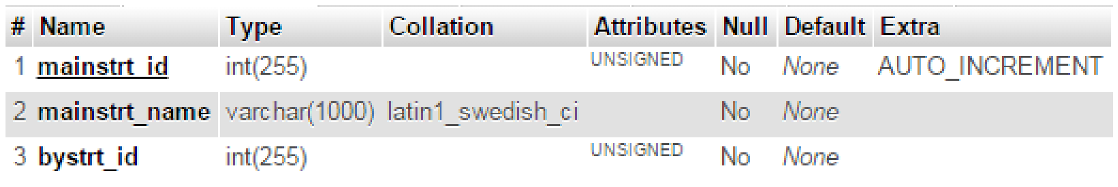
4.2.1 Table shows the field of hospitals database table



#	Name	Type	Collation	Attributes	Null	Default	Extra
1	hos_id	int(255)		UNSIGNED	No	None	AUTO_INCREMENT
2	hos_name	varchar(1000)	latin1_swedish_ci		No	None	
3	bystrt_id	int(255)		UNSIGNED	No	None	

Figure (4.1): hospitals database table

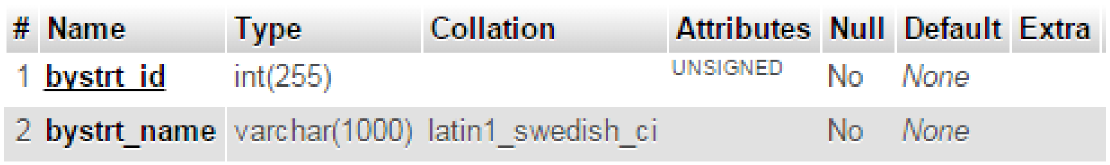
4.2.2 Table shows the field of main streets database table



#	Name	Type	Collation	Attributes	Null	Default	Extra
1	mainstrt_id	int(255)		UNSIGNED	No	None	AUTO_INCREMENT
2	mainstrt_name	varchar(1000)	latin1_swedish_ci		No	None	
3	bystrt_id	int(255)		UNSIGNED	No	None	

Figure (4.2): main streets database table

4.2.3 Table shows the field of bystreets database table



#	Name	Type	Collation	Attributes	Null	Default	Extra
1	bystrt_id	int(255)		UNSIGNED	No	*None*	
2	bystrt_name	varchar(1000)	latin1_swedish_ci		No	*None*	

Figure (4.3): bystreets database table

4.3 Screens:

4.3.1 The main Screen

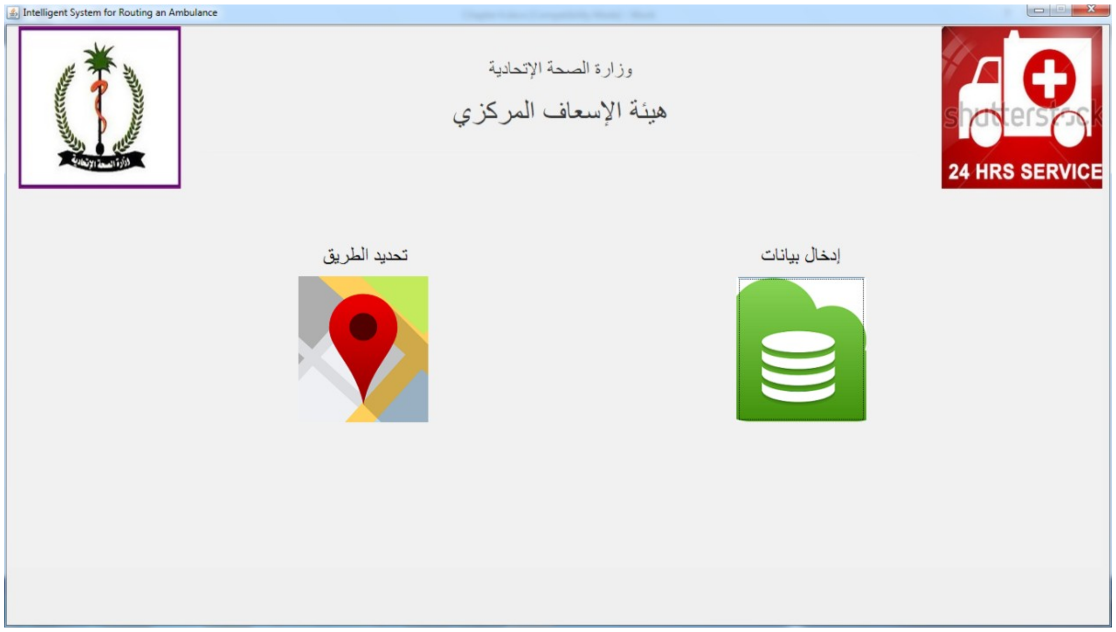


Figure (4.4): System main screen

Figure (4.4) shows the main screen of the system where the name and the logo of the system are shown.

It has two buttons:

- The first button is for entering data to the system when user .click it will open new window
- The second button is for getting route between the ambulance location and the caller location

4.3.2 Entering Data File

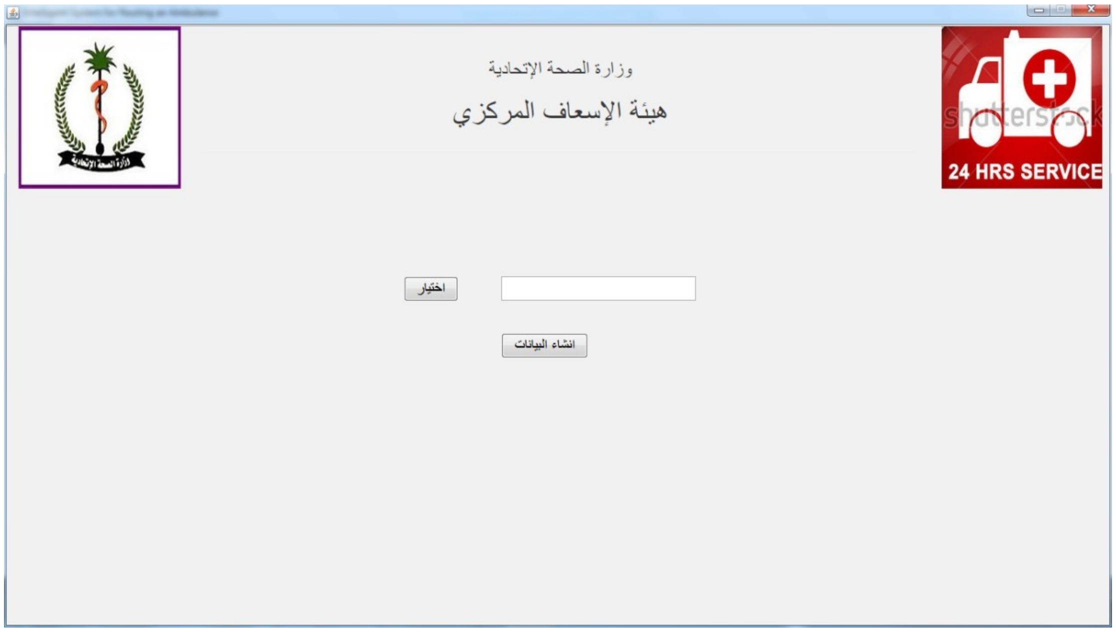


Figure (4.5): Entering data file

Figure (4.5) shows the screen for entering data file from the local disks or removable disks.

It has two buttons and a text field as follow:

- The text field is un-editable it shows the full path and the name .of the file which user was chosen
- The first button is a choose button when the user click it will open pop-up window to choose the file from local storages as .figure (4.6) shows
- The second button is creating data when the user click it will check the text field, if there is a text in it then will convert the data file from .gpx to prolog predicate and save it in prolog file and show message dialog of operation success as figure (4.9) shows, if it is empty it will show message dialog with the error .of no file has been selected as figure (4.10) shows

4.3.3 Choosing Data File from Local Storages

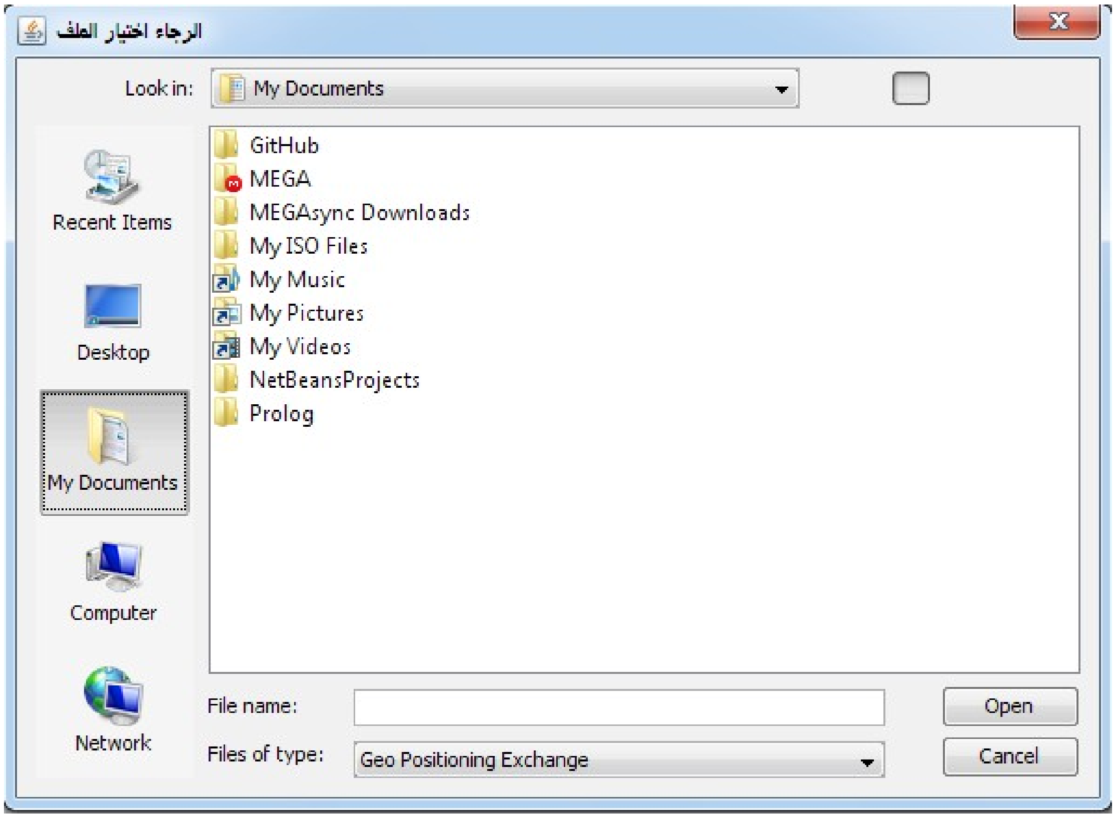


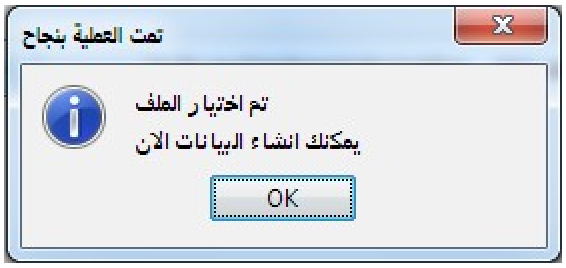
Figure (4.6): Choose data file

Figure (4.6) shows the screen for choosing data file from local storages it only shows .gpx files and the directories of your local storages and any other file will not show up here.

It has two buttons as follow:

- Open button when user click it will get the file which has been selected and edit the text field with the full path of the file and show message dialog of operation success as figure (4.7) .shows

- Cancel button when user click, it will cancel the choosing operation, close the current pop-up window and show message .dialog of operation has canceled as figure (4.8) shows

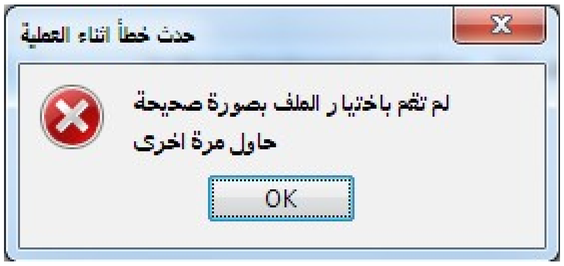


تمت العملية بنجاح

تم اختيا ر الملف
يمكنك انشاء البيانات الآن

Figure (4.7): File chosen done successfully

Figure (4.7) shows message dialog to inform the user that file has chosen successfully and ready to create the data.



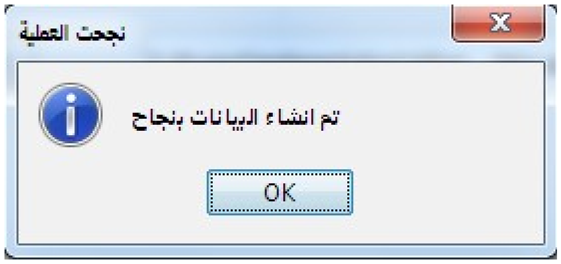
حدث خطأ اثناء العملية

لم تقم باختيا ر الملف بصورة صحيحة
حاول مرة اخرى

Figure (4.8): File chosen failed

Figure (4.8) shows message dialog to inform the user that error has occurred during choosing your file and tell him to try again.

4.3.4 Creating Data Done Successfully



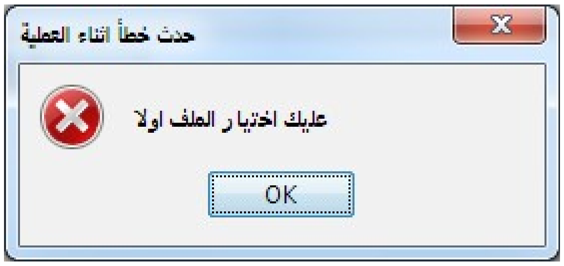
نجحت العملية

تم انشاء البيانات بنجاح

Figure (4.9): data created successfully

Figure (4.9) shows message dialog to inform user that the chosen file data successfully stored in the database and the prolog data also created successfully.

4.3.5 Creating Data Failed

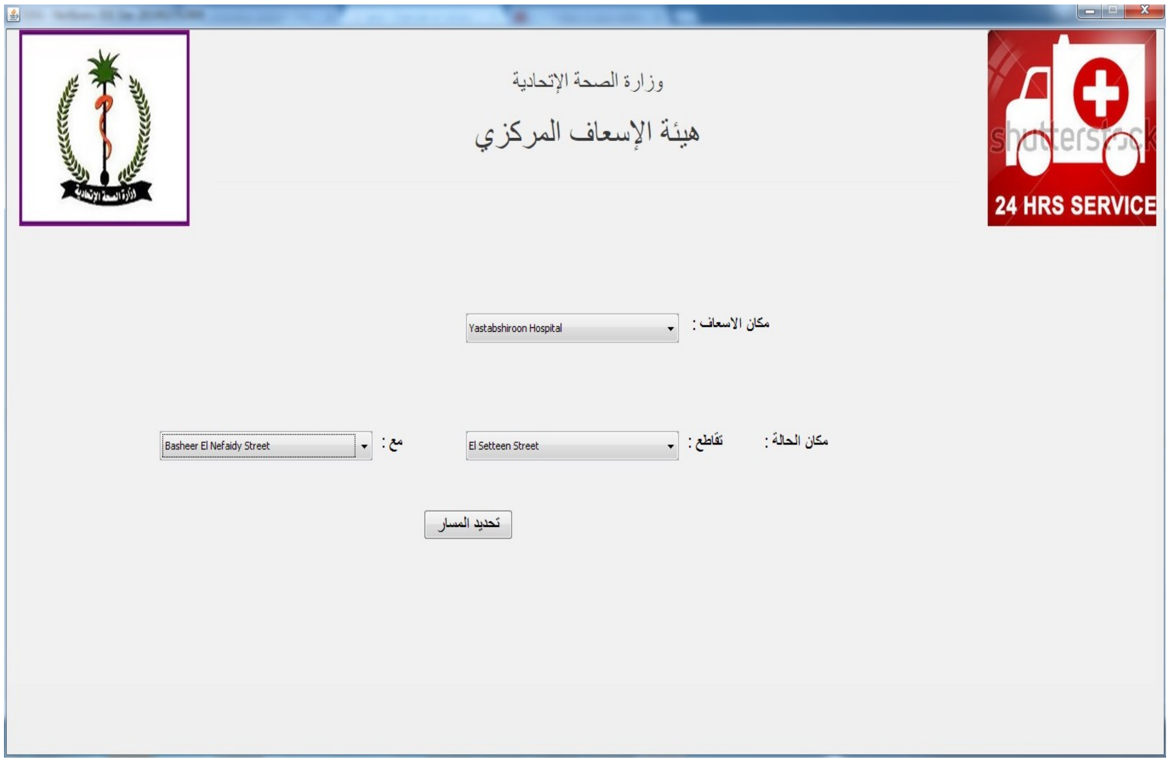


عليك اختيار الملف اولا

Figure (4.10): data creation failed

Figure (4.10) shows message dialog to inform user that he has to choose some .gpx file to create data from it.

4.3.6 Find Route



مكان الاسعاف : Yastabshiroon Hospital

مكان الحالة : تقاطع : El Setteen Street مع : Basheer El Nefaidy Street

Figure (4.11): Find Route

Figure (4.11) shows the screen of finding route between ambulance location and emergency event location.

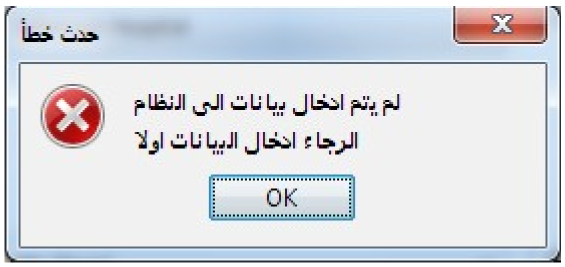
It has three drop-down lists and a button as follows:

- First drop-down list for choosing the ambulance location which could be a hospital from the hospitals that stored earlier in the .database from the .gpx file when creating data

- Second drop-down list is representing the main street in the intersection where the emergency event had occurred, it is .retrieved from the database as hospitals

- Third drop-down list is the bystreet in the intersection which retrieved from the database also, but the items of it depends on the main street, further more it only shows bystreets which .can intersect with current selected main street

- Button for finding the route and display it on the screen, but it validate the data if one of the drop-down lists is empty it will .display an error message on the screen as figure (4.12) show


حدث خطأ

لم يتم ادخال بيانات الى النظام
الرجاء ادخال البيانات اولا

OK

Figure (4.12): entering data error

Figure (4.12) shows error message to inform user that an error has occurred during entering data because one or more field is empty, and ask him to renter his data first.

4.4 The algorithm and prolog

The algorithm (maze solver)

We had took the road network as a maze with the start at the ambulance location and the finish at the emergency event location. The object is to find a path through the road network from the start to the finish. First, we must represent the road network in a form Prolog can use. So we had created a java library to convert the *.gpx* file to prolog predicates, the library used to create the prolog data file. If we can move from one position to another, we will say that these two positions are connected. We enter a single fact in the definition of the predicate *connect/*2 for each pair of connected locations. Then we define a *connected_to* predicate using the predicate *connect*:

```
connect('Khartoum ENT Hospital ',' El Qasr Street').
connect('Makka Hospital ',' Ebaid Khatim Street').
connect('Dream Le El Wilada Hospital ',' Basheer El Nefaidy Street').
...
connect('Khartoum Teaching Hospital ',' Army Street').
connect('El Khartoum Le Awram Hospital ',' El Mak Nimr Street').

connected_to(Location1,Location2) :- connect(Location1,Location2).
connected_to(Location1,Location2) :- connect(Location2,Location1).
```

A path throw road network is a list of position with *Start* in at one end of the list and *Finish* at the other, such that every position in the list is connected to the position before and after it. Initially our path contains a single position *Start* which we place into a list. From this initial path, we want to generate complete path from *Start* to *Finish*. Once a solution is found, we want to display it.

```
find_route(Start,Finish) :- path([Start],Solution), write(Solution).
```

The procedure *path* will find the solution, of course when we reach *Finish* we have a solution and our search is ended.

```
path([Finish|RestOfPath],[Finish|RestOfPath]).
```

At each intermediate step in our search for a solution to the path through the road network, we have a list of the positions we have already visited. The first member of this list is our current position. We proceed by looking for a new position that we can reach from our current position. This new position must be connected to our current position. We don’t want to move around in a circle or back and forth between the same positions; so our new position will also have to be a location that isn’t already in the path we are building.

```
path([CurrentLocation|RestOfPath],Solution) :-
    connected_to(CurrentLocation,NextLocation),
    \+ member(NextLocation,RestOfPath),
    path([NextLocation,CurrentLocation|RestOfPath],Solution).
```

If the procedure *path* reaches a point where it cannot find a new position, Prolog will backtrack. Positions will be dropped off the front of the path we have built until we reach a point where a new position can be reached. Then the search will move forward again until we reach Finish or another dead-end.

CHAPTER 5

RESULTS AND RECOMMINDATIONS

CHAPTER 5

RESULTS AND RECOMMINDATION

INTRODUCTION 5.1

This section discusses the most important results that we have achieved after the implementation of the system, and the recommendations that we recommend to improve or add new features can increase the interactive and efficiency of the system.

RESULTS 5.2

After the implementation of the system and conduct tests to verify the functionality required of the system has been reached routes an ambulance through Khartoum City road networks which offers:

.1 .Creating GPS data base from GPS data file

.2 Finding the route between ambulance location and casualty .location

.3 In the previous studies they used the GPS data file to create the map but we parsed it and created our data base from it. As we have more flexibility in our project because we can implement the project in any city by just its GPS data and the .system will do the rest by creating GPS data base of the city

After shifting from the manual system to our proposed system we expect to deliver the casualty to the hospital within golden time.

As this proposed solution will affect the average death rate because many casualties will be delivered in the best time to get medical care. Also this solution can be implemented to any another city or town and it will work well after creating GPS data base for that city or town.

RECOMMINDATIONS

1. Implement the project in all other Cities in Sudan.
2. When traffic data becomes available implement it in the project to find the route based on it.
3. Put the system online or in cloud computing.
4. Create mobile version of the system.
5. Use Python and ArcGIS to build the system and compare it with the existing one.

CONCLUSION

The Intelligent System for Routing an Ambulance (ISRA) provides route to the ambulance drivers to go to their destinations in the golden time and with that maybe save very valuable lives.

It also could be used to find route for another emergency situations like police vehicles or even fire vehicles.

REFERENCES

1- Poole, David; Mackworth, Alan; Goebel, Randy (1998). Computational Intelligence: A Logical Approach. New York: Oxford University Press. ISBN 0-19-510270-3.

2- McCarthy, John; Minsky, Marvin; Rochester, Nathan; Shannon, Claude (1955). "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence". Archived from the original on 26 August 2007. Retrieved 30 August 2007.

3- McCarthy, John (12 November 2007). "What Is Artificial Intelligence?".

4- Niederlinski , Antoni (2014). "A gentle guide to constraint logic programming via eClipse" . Gliwice : Jacek Skalmierski Computer Studio.

5- Wielimaker , Jan. (May 2014). "SWI-Prolog Reference Manual updated for version 6.6.6".

6- Bellman, R. (1962), "Dynamic Programming Treatment of the Travelling Salesman Problem".

7- Applegate, D. L.; Bixby, R. M.; Chvátal, V.; Cook, W. J. (2006), The Traveling Salesman Problem, ISBN 0-691-12993-2.