CO600 Project 2007/2008

Daniel Ives
Mohammed Khan
Heidi Lees
Nathan Phillips
Graham Ridley

University of Kent

Virtual Doctor

An Intelligent system that offer diagnoses on user symptoms

Daniel Ives di6@kent.ac.uk
Mohammed Khan msk20@kent.ac.uk
Heidi Lees hsl2@kent.ac.uk
Nathan Phillips np43@kent.ac.uk
Graham Ridley gar8@kent.ac.uk

Abstract

In recent years, the population of the UK has been steadily rising; however the number of GPs has been decreasing [1]. This has meant that many doctors around the country have had to take on more patients, placing more workload on already overworked GP surgeries. This in turn has meant that patients have had to endure longer waiting times or having to register with a doctor who is not their first choice. As a result of this, more and more people with minor illnesses or busy lifestyles are looking for self diagnoses to get a general idea of what may be wrong with them. Thus empowering them with more knowledge and education of their condition, and letting them know if they need to see a doctor. This in turn acts as a screening service for GPs who do not have to see patients with non-serious conditions and reduces their workload.

1. Introduction

While there are number of such medical self-diagnosis services, they are currently limited in the type of diagnosis they can give. They cannot for example take your blood pressure, measure a user's blood sugar level etc. We want to take this further.

What has been proposed by our supervisor Duncan Langford is to develop an application that provides the first step in implementing a kiosk facility, potentially placed in a supermarket or shopping centre. Users can enter their symptoms, have their weight checked, have their blood pressure taken, and possibly be connected to a GP for further diagnosis through a webcam.

The main feature of this application is its intelligent system. By analysing the user's age, weight, height, symptoms, and possibly blood pressure and blood sugar level, the system will ascertain what may be wrong with the user. The application in its entirety can be classified as a knowledge based system, containing a relational database on information regarding symptoms and illnesses; which the intelligent system in turn analyses, taking into account the user details to present a possible result.

2. Background

The idea for this project was conceived by Duncan Langford, who provided us with the basic concept and left us to use our own initiative to develop the different functions and methods for producing Virtual Doctor.

When considering implementation of a system that is 'intelligent', we do not mean creating an application that contains human elements of intuition and common sense that could lead to replacing a professional physician. Rather it is a system that has programming features that make it aesthetically seem as though it has taken the relevant data inputs and 'thought' about what to give back to the user. Further information on what was used to make our system intelligent is included further on in this technical report.

While there are a number of online services that currently offer similar types of facilities such as WebMD's symptom checker, most are based on the Internet, and rely solely on the user's symptoms. [2][3] At present there is no application based in a tangible location where users can sit down and have their vital signs taken to be analysed, or offers the user a chance to talk to an actual doctor if necessary. This would also enable people who are computer illiterate, or do not have access to the Internet to make use of such a service.

2.1 Aims

Our main objective for this project is to design, create, develop and evaluate the first step of the Virtual Doctor system. Ergo to develop this system as far as possible within the time frame given and leave open the potential to have any features we did not have the opportunity to implement, to be placed in later if this project was to go further in the future. This project must also take into account the fact that it is potentially to be placed in a kiosk, ideally situated in a public place such as a supermarket. Therefore the user interface must be designed so that it can operate without much difficulty.

It must also have a sense of intelligence, in that it can offer an accurate result given the user's symptoms and vitals. As an example, we do not want a user to tell the system that he or she has a pain in their neck, and the system in turn, to tell them they may have malaria. But rather it should display the potential ailments and highlight those that are most likely. So that if the user does complain about neck pain, it can display either whiplash, or meningitis, but have an idea of which one is the most accurate given the user's other symptoms and possibly their weight and age.

3. Features of the Application

The application already has numerous features which allow the user to easily diagnose their symptoms and also has a separate administration tool to update the database via its own interface.

3.1 Intelligent System

The problem that we faced when presenting a self-diagnosis system to the general public was that a lot of people would present the system with symptoms that would not relate to certain illnesses, or be circumstantial to the day. An example, may be that a user presents all the symptoms of a common cold, but a skin rash on the day that may have come from something not related to the illness, prevents an accurate result from being shown.

With this in mind, a user could potentially diagnose themself with an illness which the system will suggest to them to see their doctor about. This could then lead the user to panic. Therefore we implemented a system which presents the user with a range of illnesses relating to the symptoms entered, but are honed down to eliminate unlikely results, and present the user with a scale of possible ones to show how likely it is that they could have one of the ailments presented. This is how we developed our 'intelligent' system.

3.1.1 Symptom Search - Spelling

As our system is designed to be used by a wide variety of people, including those with handicaps, and those who are computer illiterate. It is very likely that when entering symptoms into the system some users will incorrectly spell words.

We have therefore utilised an 'intelligent' text searching system, should the user misspell the symptom they are looking for [4]. This utilises a phonetic spell checker, so should the user search for the symptom "coff", the phonetic system can work out how it sounds and check it against various phonetic codes within the database. This works because when a symptom is added into the database via our administrative side, it calculates the phonetic code and adds it to the appropriate table within the database. Then on searching, if the symptom is not found via normal string matching within the database, we convert the symptom entered into a phonetic code and search over the relevant database entries.

To first get this phonetic code, we have to add it into the database. This happens automatically through our administrative software. If the symptom is not previously in existence within the database, it adds it into a specialist table, along with a generated phonetic code. If the symptom already exists within this table, it is simply added to an illness.

In order to provide more functionality and more results with our front-end system, we changed the SQL search so that the user is presented with symptoms that also contain the word, or phonetic code, within them. Therefore if the user were to search for a cough, we could present a variety of results, such as chesty cough or dry cough. This also works with the phonetic code, and it searches for any similar match with the phonetic code provided.

3.1.2 Symptom Search – Optimal results

When the user has selected the appropriate symptoms they wish to search for, the system runs a calculation over them. It calculates the percentage of the symptoms input by the user, against the symptoms in a relating illness. A percentage multiplier is increased in each case if the user's BMI and age range have any effect on any of the possible ailments, such as being overweight would increase your chances of having diabetes. If this result presents itself to be over 20%, the user is shown the related illness, and depending on the percentage, it is given a coloured ranking that shows the user how likely it is that they may have that particular problem. The results are also ranked in order of most likely to least likely, so that the user is presented with the top recommendations first.

3.2 Graphical User Interfaces

3.2.1 Front-End

One aspect of Virtual Doctor's visual functions is the front-end GUI which the users will be interacting with and using for diagnosis. Initially a lo-fi prototype was created using techniques learnt in CO529 – Human Computer Interaction.

There were a number of considerations that were thought out when designing the lo-fi prototype of the front-end GUI, primarily the fact that Virtual Doctor will be potentially used by people of a wide age-range and of mixed computing ability, therefore the interface needed to be clear, efficient and essentially simple to use. It was important that the interface was uncomplicated and consistent throughout the process of use. This was reflected and tested in the lo-fi prototype, which through the effective and unobtrusive use of colour combined with a consistent appearance, makes for a pleasant experience.

The front-end was originally designed in NetBeans, however it was imported into BlueJ for consistency reasons and to allow it to cooperate and integrate with the functional classes.

A CardLayout was designed and implemented for the front-end GUI. CardLayout allows selection through multiple Panels, showing only one Panel at a time. The reason for choosing this layout system is that it is very useful because the Virtual Doctor program has various screens which are needed to be displayed at different times, in or out of order.

The GUI is organised into relatively independent components which communicate via the Main class. The components learn about the state of each other from this class. Main.java pulls together all the UI components into a single interface. There are five panels in total and their main purpose is described as follows:

Welcome Panel: This class is the first panel the user is presented with when the application is started. It contains the logo of our program and information about what the software does. It is merely aesthetic and does not contain any valuable methods, but is a definite requirement for our product to welcome new and returning users.

Terms Panel: This class is the second panel the user will see. It describes to the user the terms and conditions that need to be agreed to in order to proceed with use. We have chosen to include this because of the type of information we are providing to the user. As we are offering a type of medical diagnosis, we need to inform them that this system is for informational purposes only, and any information they derive from the program through misuse or poor information quality, is their responsibility. It ensures that we do not face any legal action as creators in the case of a misdiagnosis. The user cannot proceed unless they have agreed to the disclaimer.

Details Panel: The third panel which is presented to the user. It performs a number of functions and takes a series of inputs from the user. The details required are the users' sex, age, weight and height. These details are taken in order to provide the user with refined illness results based on these factors. The user can also view their body mass index (BMI) and be informed of their weight class visually with a graph. Currently, the class does not allow weight entry, as ideally with the commercial release of the product, a special chair would measure the user's weight and fill in the field automatically.

Symptoms Panel: This is the penultimate panel and is an extremely important part of the program. This panel contains the image of a human body, which the user can use to find symptoms associated with the corresponding body part. This was implemented again for the ease of use of some users. It was implemented using JPanels, each 'holding' a section of the human body. All the images were then placed together to look like they are one image. Action listeners are then added to each panel, so that when a user clicks on, for example the head, the possible symptoms are displayed in a large text box where they can select the symptoms they have, and add them to a personalised list. There is also a text search function that allows the users to search for symptoms by name. (This is implemented by the intelligent spell checker as explained previously).

The class contains "return" methods in order to pass data through the main class and utilises methods from a number of different classes in order to retrieve information and display it.

Report Panel: The last panel, which details to the user the possible illnesses they could have. Upon clicking the illness name, details about the illness are displayed and advice given to the user about what action they might want to take. The results are displayed in priority order, with the first illness being the most likely candidate. The illnesses are also colour coded so that the user can quickly identify which ones are the most likely.

3.2.2 Back-End

For the Virtual Doctor project we decided to split up the GUI that the user would be presented with, and the GUI that a database administrator would use, because we felt it would be best to develop them as separate modules. This allows them to work as two separate stand alone applications that will interact with the same database.

The back-end application is where the administrative users will be interacting with the database to initially populate data and then make any additional amendments that might be needed in the future.

The GUI will allow the user to perform many functions on the database:

1. Add new illness
2. Search for illness
3. View and amend illness
4. Add symptoms to an illness
5. View and amend a symptom
6. Delete an illness or symptom

Initially a lo-fi prototype was created for the back-end application's GUI to establish how a user might use the system. We had to consider how the user would interact with the database and what functions they might want to perform, creating an iterative user centred design. We created different panels for the various functions a user may wish to perform so that everything was not cluttered on one screen. Prototyping this way allowed us to evaluate the design without having to constantly change code to create different looks.

Originally we had created a design that would have multiple screens, which would then pop-up for the user depending on what function they wanted to perform. We realised that this would be frustrating to the user as they would have screens all over their monitor.

We decided that a CardLayout design should be used for the back-end application GUI as this would fix all the separate JPanels into one frame that would only be shown to the user when they were needed. The reason for choosing this layout system is that it accommodates the various panels needed for the Virtual Doctor back-end application, enabling these panels to be displayed at different times, in or out of order.

The frame that the panels load into has a side bar that contains all the buttons needed for the user to interact with the panels and perform tasks. These buttons change, with different ones being displayed depending on what panel is being displayed to the user. This is done by creating JPanels for each button set and setting visibility true or false depending on what screen is being shown.

The frame also stores all the information for the current illness being viewed by the user so that the frame can parse illness parameters between the different panels (i.e. saving a new symptom to the correct illness in the database)

The GUI interacts with the database using various classes. These classes are as follows:

1. DatabaseControl (controls the database connection and SQL commands).
2. DatabaseCollection (collects all information and retrieves data from the database when the user needs it).
3. DataEntry (saves information to the database).
4. DataUpdate (updates information in the database and also used to delete entries).

The back-end application allows the administrator of the database to effectively perform all the tasks they would need to easily and efficiently.

3.3 Database

3.3.1 Database purpose

The database is the knowledge base of the system and contains all the information on the illnesses that we have selected for this version of Virtual Doctor. In any future commercial developments the database would be fully populated, but for now we are choosing a small number of symptoms and illnesses just to test the functionality. The database has been designed to work as efficiently as possible with the Java code used in the application. The database is also essential for the spell checking functionality to work, as the database contains all the phoentic codes that are used to correctly identify misspelt symptoms entered by the user.

3.3.2 Database format

It was decided that it was in the best interest of the project to use Microsoft Access as the database management system. The database did not need to be extremely complicated and by using an advanced alternative such as mySQL, problems arising from the database would have been more likely. In the future however it may be more

beneficial to move the database to a dedicated mySQL server; this will be more efficient when more illnesses are added, as mySQL will be able to cope better with the increase in data.

Information about illnesses and symptoms were researched on the NHS website. [5]

3.3.3 Database design

Appendix 1 shows the final design of the database that has been implemented into the project

All bold attributes within the tables are the key fields.

3.3.4 Integration with Java

The Java code that was implemented in our Virtual Doctor application used many SQL queries to call the correct data from the database when it was needed. All the queries that were used in the Java code were in the SQL format and if the database is to be moved over to a mySQL server database then these SQL statements will need very little alteration if any. This leaves open the possibility of a near seamless change over in the future.

4. Testing

4.1 Purpose of testing

Software testing is the process used to help identify the Correctness, Completeness, Security and Quality of developed computer software. Testing software is essential in removing bugs and errors from applications.

4.2 Types of testing applied

We used three different types of testing for our system. These were:

1. Visual User Testing
2. Automated Unit Test Cases
3. End User Testing

Visual user testing meant a member of the group would actively work through the application and follow the set test cases to see if the expect results actually occurred. Any differences throughout any of the steps were noted down, documented and then processed for changing within the program. See **Appendix 2** for an exemplar test case

The Automated test cases that were created executed certain parts of the system's code that to us seemed likely to be prone to errors. These include the spell checking system, and the ordering of illnesses ranked by severity. These test cases were run throughout the creation of the program so that when new code had been implemented we knew that our working code was never affected.

End user testing is an extremely essential part of the testing cycle, and it is here that feedback on our system from real users can be acquired. By testing the system with the type of people who would be using the system in real life, we can get an idea of how our system would be used in the real world. Testing by non computer literate users also shows how user friendly the system is and will point out improvements for errors that may have initially been passed over.

With all testing carried out it is then necessary to review the results and then act on them. This means carrying out extensive bug fixing on parts that failed, or looking into aspects that haven't failed but have been of inconvenience. An example of this is slow processing between screens.

5. Project Management

5.1 Quality Assurance

Quality Assurance was split into two sections for the Virtual Doctor project. Firstly, a document was created and adhered to, ensuring that all code complied with standard convention rules. Secondly, all documentation was created out of templates made especially for Virtual Doctor and was checked and signed off before being finalised.

5.2 Risk Analysis

Risk analysis is the technique to identify and assess factors that may jeopardize the success of a project or achieving a goal. This technique also helps to define preventive measures to reduce the probability of these factors from occurring and to identify countermeasures to successfully deal with these constraints when they develop. This will help to avert possible negative effects on the project.

A risk analysis was performed for the Virtual Doctor project to minimise the impact of any such risks should they arise.

6. Conclusion

6.1 Overall Conclusion

Looking back on the overall development of the project, we believe we have succeeded in designing and creating a viable product that has demonstrated our wide range of computing skills gained in our years at university and from our year in industry. Our project does not only have potential commercial benefits, but is something that can benefit society as it was designed to provide a solution to a problem currently being experienced in the UK.

Given more time and resources we believe we would have been able to develop this project further to fulfill the complete aims and be able to present it to a commercial investor for deployment into the public environment.

6.2 Known bugs

While populating the database using the back end application we found that it was not possible to add a symptom to the database with the same name as another symptom that is already saved in the database even if the new symptom has a different location. For example the symptom name 'swelling' could not be added for the wrist and then added for the ankle.

When trying to do this, the instances of each symptom would be added to the symptom code table in the database but there would be no unique field to link swelling in different locations to different illnesses. Therefore when viewing the details of both illnesses only one type of swelling would be shown, and obviously in one case, would be the wrong location. This fault would clearly be repeated if the database was to be populated with a lot of data, making a diagnosis hard to formulate.

This design fault came about due to our lack of domain knowledge in medical conditions. Also, while testing the application during development we had not entered illnesses that would have alerted us to this fault.

It wasn't until late in development that we found this fault in our design, and due to time restrictions, we did not have the available resources to fix the problem. This would have needed us to make changes to the whole of the virtual doctor system.

A short term solution to this problem would be to include the location in the name of the symptom, for example Swelling (Ankle). This obviously wouldn't be suitable for a commercial product so a re-design to the system would need to be made.

To rectify this design fault we proposed that a new field would be created in the symptom code database that would hold a unique code for a symptom that would also take into account the primary and (possibly) the secondary location. The code would consist of the phonetic code created from the symptom name and also codes for the location. This would allow a symptom to be unique for an illness, but some code re-designing will need to be done on both the front-end and back-end applications so that the system will take into account these design and database changes.

6.3 Future improvements

There are numerous ways of extending the Virtual Doctor project. The program was written with extensibility in mind and one of the major ways it could be developed further would be to implement a "Weighted Seat" to automatically input a user's weight. For a commercial product we have envisaged that the user would sit on a weighing seat to determine how much they weigh, this would then be input into the application instead of or as well as the user being able to enter their weight. Under the same umbrella a "Height Measure" could also be created, whereby next to the booth a tool to measure your height would stand, and the user would bring it in line with their height, and the results would feed back into the application. Both of these methods would potentially increase the accuracy of the data being fed into Virtual Doctor and help improve the results of the diagnosis.

Virtual Doctor could also be developed by the use of plug-in devices (e.g. a heart rate monitor). Plug-in devices would be used so that their inputs are put into the system to help determine what illness a user is more likely to have. A heart rate monitor is an obvious first addition as many illnesses can be linked to a person having a high or low heart rate. Other such devices could include blood pressure measurer, digital pulse oximeter or a device that measures a user's temperature.

Currently the system is not set up so that it will distinguish between female or male only illnesses; therefore a female user could have testicular cancer as a result which is obviously an undesirable flaw. The project could be extended by categorizing certain illnesses as male or female only. This can be

rectified by creating a simple flag in the database similar to how the BMI flag has been implemented so that illnesses can be set so that they can affect only females, only males or affect both sexes.

Another way the system could be developed is that information could be uploaded to a hub to calculate and graph illnesses in various locations. If Virtual Doctor was placed in local areas the returned diagnosis results for users could be anonymously uploaded to a central hub so that diagnosed illnesses could be recorded for different areas. This would allow the central hub to track if illnesses in areas were increasing, for example common cold or flu. The data could be analysed for a number of reasons and sent to local authorities, however the disclaimer would be updated to inform the user that this data collection was occurring.

Once a diagnosis has been displayed, a consultation could be set up with a local doctor via a webcam and microphone. This was something that we originally intended to accomplish but due to a restriction of time we were unable to undertake this task. If the illness results for a user are considered to be serious enough for GP consultation then the user would be connected to a doctor via a webcam link.

With regards to the front-end GUI: In the future there is the potential to have the front-end personalised to conform to specific environments. For example, if the application is placed in a company's lobby, it can be changed to have that company's colours and logos. Or if placed into a specific supermarket, the GUI could be designed to match the look and feel of the store. So the user would feel they are using something that affiliated with the store itself.

7. Acknowledgements

The group would like to thank Dr. Duncan Langford for his help, support and advice throughout the whole project.

We would also like to thank Pulitha Liyanagamama for his assistance with the implementation of the image map.

8. Bibliography

[1] "GPs 'refusing new patients'" – *BBC News Online*
http://news.bbc.co.uk/1/hi/health/2204851.stm

[2] WebMD Symptom Checker
http://symptoms.webmd.com/

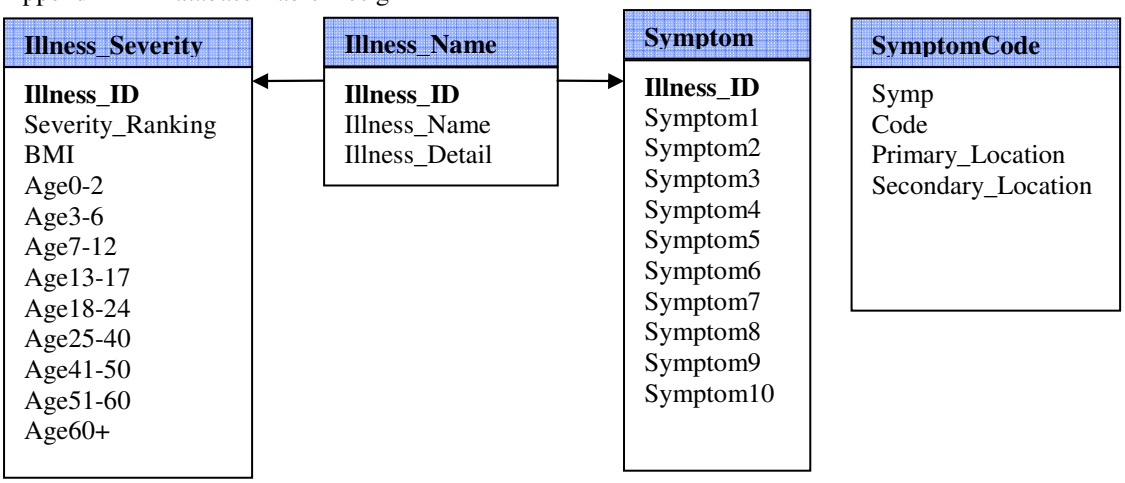
[3] MayoClinic Symptom Checker
http://www.mayoclinic.com/health/symptom-checker/DS00671

[4] Phoenetix -
http://www.companywebstore.de/tangentum/mirror/en/products/phonetix/index.html

[5] NHS – http://www.nhs.gov.uk

Appendix

Appendix 1 – Database Table Design



Appendix 2 – An exemplar user test case

Step-by-Step Test Case

Test Case # and Name: 001 # GUI – Screen change 1 to 2	Module/Function/Unit:	Build #: 1	Test Run #: 1
	Requirement #:		
Test Case Description (purpose and method): To see if screen 1 and 2 of the GUI are linked correctly using the NEXT button		Automated Script Name (if applicable): N/A	
Setup for Test (hardware, software, prerequisite tests, data, timing, security): Blue J must be running with the latest project version loaded			
Success End Condition: Screen 2 should appear	Failed End Condition: Error dialog box will appear.		
Open Issues: none			

Step	Action	Expected Results	Actual Result	Pass/Fail
1	Execute the Main GUI class	Virtual Doctor main screen should appear – Full screen		P
2	Click the NEXT button located at the bottom of the screen	The second GUI screen should appear displaying the disclaimer		P