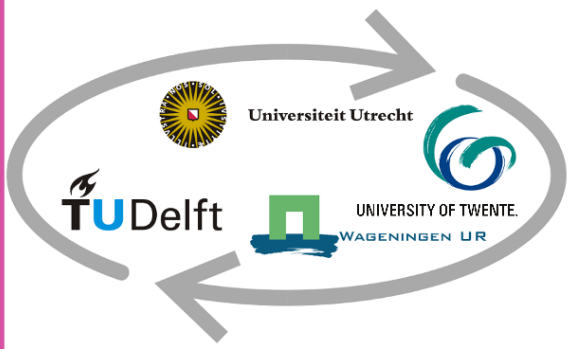
GIMA
Geographical Information Management and Applications

Interactively Planning Resilient and Connected Field Hospital Locations in a Conflict Area

Case study Mosul, Iraq

Thijs van der Caaij
ITC: s6031250 UU:6035736
thijsvandercaaij@gmail.com
+31 (0)6 45025304

Supervisor:	Ellen-Wien Augustijn
Date:	22-08-2019
Version:	Final Report Revised



CONTENTS

1 INTRODUCTION

1.1 Research Context

The Islamic State of Iraq and the Levant (ISIL) seized the city Mosul, Iraq in June 2014. This led to attempts to retake the city from 2015 till 2017 by Iraqi, Peshmerga and international forces. Although the Iraqi Prime Minister proclaimed victory over ISIL on 10 July 2017, inhabitants of Mosul have continued to be displaced and in need of medical aid. The International Organization for Migration (IOM) reports 793,422 people from Mosul to be displaced by 18 October 2017 (IOM, 2017) and the World Health Organization (WHO) states that 2.7 million people were in need of health services (WHO, 2017). Together with its partners, WHO has been providing medical aid since the start of the Mosul crisis. These efforts can, among others, be seen in the form of Trauma Stabilization Points (TSP), Primary Health Care Centers (PHCC) and Mobile Medical Clinics (MMC). The MMCs are flexibly employable field hospitals which, should an alteration in the ground situation ask for it, allow them to change locations. These field hospitals are defined by the WHO as "mobile, self-contained, self-sufficient health care facility capable of rapid deployment and expansion or contraction to meet immediate emergency requirements for a specified period of time" (WHO-PAHO, 2003, p.6). Currently, their locations are chosen by a multi-disciplinary group consisting of different stakeholders, such as WHO staff, multiple armed forces and Kurdish representatives. Each of these stakeholders has their own preferences when it comes to the placing of field hospitals. The WHO, for example, manages and supplies the field hospitals, but they often need protection as they are more and more becoming tactical targets even though they are neutral (Nickerson, 2015). Then armed forces have to provide this protection, and they seek hospital locations that require minimal protection.

Resources are often limited in conflict aid, and such was the case in Mosul, as only a limited part of the population could be provided with medical assistance (IOM, 2017). The locations of a limited amount of field hospitals should be optimized, to at least fully utilize the available medical capacity. For this task, GIS has been used to get a somewhat objective analysis on the siting of field hospitals. Unfortunately, a conflict such as in Mosul proved to be unpredictable and having quite some uncertain aspects. It, for example, can be hard to predict the flows of refugees with respect to the time of relocation, the number of refugees and the destination of these refugees. This difficulty has previously led to an over assumption on the number of people that would flee to refugee camps outside of Mosul. As the distribution of people is an important factor in finding optimal locations for field hospitals, this over assumption led to field hospitals being placed near refugee camps and their capacity not fully being used, while there was a need for these hospitals elsewhere.

Besides population distribution, there are other uncertain factors, like the accessibility of the field hospital. In a normal situation, the accessibility to a certain point in a street network is somewhat fixed, but in a conflict situation, routes can be blocked by enemy territory and network bottlenecks like bridges can be damaged or destroyed, preventing people from using these points to get to field hospitals. This uncertainty can partly be addressed by choosing resilient field hospital locations. The Multidisciplinary Center for Earthquake Engineering (MCEE) characterizes resiliency of both social and physical systems with the four “R’s” robustness, redundancy, resourcefulness and rapidity (Bruneau et al., 2003). Redundancy is most relevant in the context of a street network in a conflict situation, Bruneau et al. (2003, p.284) define it as “the extent to which elements, systems, or other units of analysis exist that are substitutable, i.e., capable of satisfying functional requirements in the event of disruption, degradation, or loss of function”. In the context of this research resiliency and redundancy are treated as the same concept, and the most resilient locations are defined as those that will remain available longest to most people, when an increasing number of routes become unavailable. Xu et al. also mention that "redundancy is vital for transportation networks to provide utility during disastrous events" (2015, p.1). Redundancy is said to be only one of the measurements for resiliency though. A further exploration on the fitness of these measures should be made to come to more resilient field hospital locations. The concept of resilient locations will be further addressed in the theoretical part of this research.

While GIS can arguably prove useful in the planning of field hospitals, it relies heavily on the accuracy of provided data. If this data comes with big uncertainties and dynamics due to the conflict situation, then there is a need for a more flexible and resilient method of field hospital planning in conflict

situations. This need can be addressed by a spatial decision support system (Sugumaran & Degroote, 2010).

Multi-criteria decisions such as the selection of field hospital locations are frequently assisted by multi-criteria decision models (MCDM). These vary greatly in complexity and mathematical knowledge needed to be utilized, but the average decision-maker does not possess the required skills and knowledge to utilize such models to solve decision-making problems. This is where spatial decision support systems (SDSS) come in. An SDSS is an interactive computer system with the purpose of assisting in spatial decision-making by attempting to solve semi-structured spatial problems. It could be used in an operator/decision-making room to be run on a decision (touch) table to increase ease of use and allow for convenient decision-making in groups. This is of great importance as often multiple stakeholders are involved in choosing facility locations. This is also relevant to the Mosul case, as parties with different perspectives on the problem are involved in the decision-making, such as armies, aid organisations, government officials and possibly other groups. A multi-criteria decision model can be implemented in the SDSS to assist the decision-makers.

A way of coping with the uncertainty of data on conflict areas lies in the interactivity of an SDSS. Although data can be out-dated or incomplete, decision-makers often have hands-on experience with the facts on the ground and with similar previous situations, as opposed to just access to abstract data. Such experience is valuable in decision-making as it leads to knowledge that can complement the uncertain data. An SDSS based on flexible and interactive algorithms could allow decision-makers to make on the fly contributions to data that is being considered in the MCDM. For example, if a decision-maker knows that a particular area is risky or a road is inaccessible, the SDSS could allow the decision-maker to adjust the location of the field hospital running an MCDM. This also allows for easy experimentation with different scenarios, while providing corresponding performance indicators to certain scenarios to the decision-maker.

There exists an offering of general SDSSs that aim to facilitate multi-criteria decision problems, but these are often too complex, unclear and too general for the average decision-maker. Jacko et al. (2003) prove that customized SDSSs that take into account the requirements and constraints of specific decision problems, are likely to result in more productive SDSSs. Since the selection of field hospital locations in conflict areas is a very specific decision problem, the need for an SDDS designed around this problem, is identified.

While on the fly adjustments to data and an MCDM could prove very helpful to select locations for field hospitals in a conflict situation, the computation of adjustments and performance indicators should not take too long in order to keep the SDSS user-friendly. Not making the SDSS too computationally requiring possibly forces us to reside on coarse, less complex but often also less accurate MCDMs. These considerations will be further discussed in the theoretical framework and methodology of this research.

1.2 Research Objectives

1.2.1 General objective

The main objective of this research is to design and develop a spatial decision support system for the selection of field hospital locations in conflict areas that is data extensive and is based on fast algorithms. The SDSS executes a set of models that will be selected by a literature review on different models for calculating performance indicators in the planning of field hospital locations. Performance indicators are generated which will visualize output scenario impacts. The developed performance indicators will be integrated in an user-friendly and interactive SDSS which allows for on the fly adjustments of map objects and quick decision-making. The aim is to identify locations with a high connectivity that are also resilient to conflict dynamics while reckoning with other relevant performance indicators. Three sub-objectives with their own sub-questions can be recognized.

1.2.2 Sub-objectives

A successful design and implementation of an SDSS requires knowledge of approaches that exist for performance indicators in the planning of field hospitals which cope best with the dynamics and requirements of a conflict situation. The project framework of section 1.1 showed the importance of a couple of the factors that are to be considered when planning field hospitals in a conflict situation. But

besides these factors being obviously important, little is written about factors for siting field hospitals in previous literature. Moradian et al. states that at least up until February 2014, no literature has addressed site selection for field hospitals (2016). Knowledge on these performance indicators might be found in literature on site selection of similar utilities like storm or fire shelters or custom indicators have to be designed. Further, the problem context has shown that a field hospital location desires some level of connectivity and resiliency due to the dynamics of a conflict situation. Different measurements for the connectivity and resiliency of a location should be compared to see which are most fit to be implemented in performance indicators. Review also has to be done on the components of SDSSs to create a careful design for an SDSS and on how to test an SDSS. Therefore the first sub-objective and its respective sub-questions are:

(1) To identify suitable performance indicators and SDSS components to include in an SDSS for the planning of field hospitals locations in a conflict situation.

- What can different SDSS components contribute to a successful SDSS?
- What methods are to be used to measure the 'resiliency' and 'connectivity' of a location?
- What other performance indicators matter in the context of field hospital location selection and how are they to be calculated?
- What functionality is required of the SDSS?

Next, the performance indicators and SDSS components can be implemented in the SDSS. It will be made interactive in order for field hospital location decision-makers to implement their knowledge and hands-on experience with real life conflict developments and similar experiences into the SDSS and make easy, on the fly adjustments to the input data of the SDSS. The SDSS should not only respect connectivity and resiliency of a location as performance indicators in the planning of field hospitals, but also effectively communicate this to decision-makers through visualizations in the SDSS interface. Decision makers often have great local knowledge from previous field experiences which is hard to translate to data and they make the final decision on field hospital locations, but an SDSS which easily lets them adjust and reflect on this data could greatly support them in their decisions (Rydén, 2011). Other performance indicators than location connectivity and resiliency should be visualized as well to support decision-makers, such as the served population by single field hospitals or that of all field hospitals in a certain configuration combined. At last, it is also important that decision-makers are guided through the process of planning field hospital locations by the interface of the SDSS. Therefore the second sub-objective and its respective sub-questions are:

(2) To develop a flexible SDSS which allows for on the fly adjustments and support for finding connected and resilient field hospital locations.

- Which tools should be provided to decision-makers to make the SDSS flexible and interactive?
- In what ways are performance indicators implemented into the SDSS?
- Which dialogs are needed to guide the decision-maker through the process of planning field hospital locations?
- In what way(s) are performance indicators communicated to the decision-makers?

In order for getting to know whether the designed SDSS is suitable, interactive and able to support in selecting connected and resilient locations for field hospitals, the different implementations of the SDSS have to be tested. Tests will reveal the extent to which the SDSS is fit for this task and what aspects of the SDSS possibly need to be altered. Some alterations can be made between tests, which will lead to various implementation iterations and more to be tested. Different tests will be needed and a setup will have to be designed for these tests. Also test participants have to be selected and a test script will have to be made that stresses all relevant elements of the SDSS. The tests will have to simulate real life field hospital location decision-making as much as possible. Therefore the third sub-objective and its respective sub-questions are:

(3) To test whether different SDSS implementations are a suitable support for the decision-making process of planning field hospital locations.

- Is the SDSS intuitive?
- Is it easy to adjust data in the SDSS?

- Does the SDSS produce realistic connectivity and resilience performance indicators?
- Is the SDSS fast enough?
- In what context is the SDSS tested?
- What test criteria are tested?
- Do the tests follow a certain methodology?

1.3 Research limitations

As shown in the problem context in chapter 1, an SDSS for the selection of field hospital locations should be treated as a mere support tool for decision-makers. Even more so, when assumptions have to be made, data quality proves inadequate and situations are subject to the unpredictable dynamics of a conflict. Also, SDSS will only be tested for data regarding the Mosul crisis. In order for the SDSS to be externally valid, its performances should first be tested in other environments as well. This SDSS is not tested with real decision-makers due to the inaccessibility to these people. The real value of the SDSS in supporting decision-making on the location of field hospitals can only be determined by appliance in a real and on-going conflict situation, to then be compared to conventional methods of field hospital location planning. This SDSS does only require software that is currently freely available to actual decision-makers and is therefore easy to adopt.

1.4 Research Framework

The research objectives and their corresponding questions have led to the research model in *figure 1.1* below. The research framework consists of three main phases: Literature review and methodological decisions (a), design of and implementation in the SDSS (b) and testing and evaluation (c). These phases roughly reflect sub-objectives (1), (2) and (3) of this research. For overview purposes, not all steps are chronologically presented within phases. Their order of appearance is indicated with a little section number in the corner of each step as guidance.

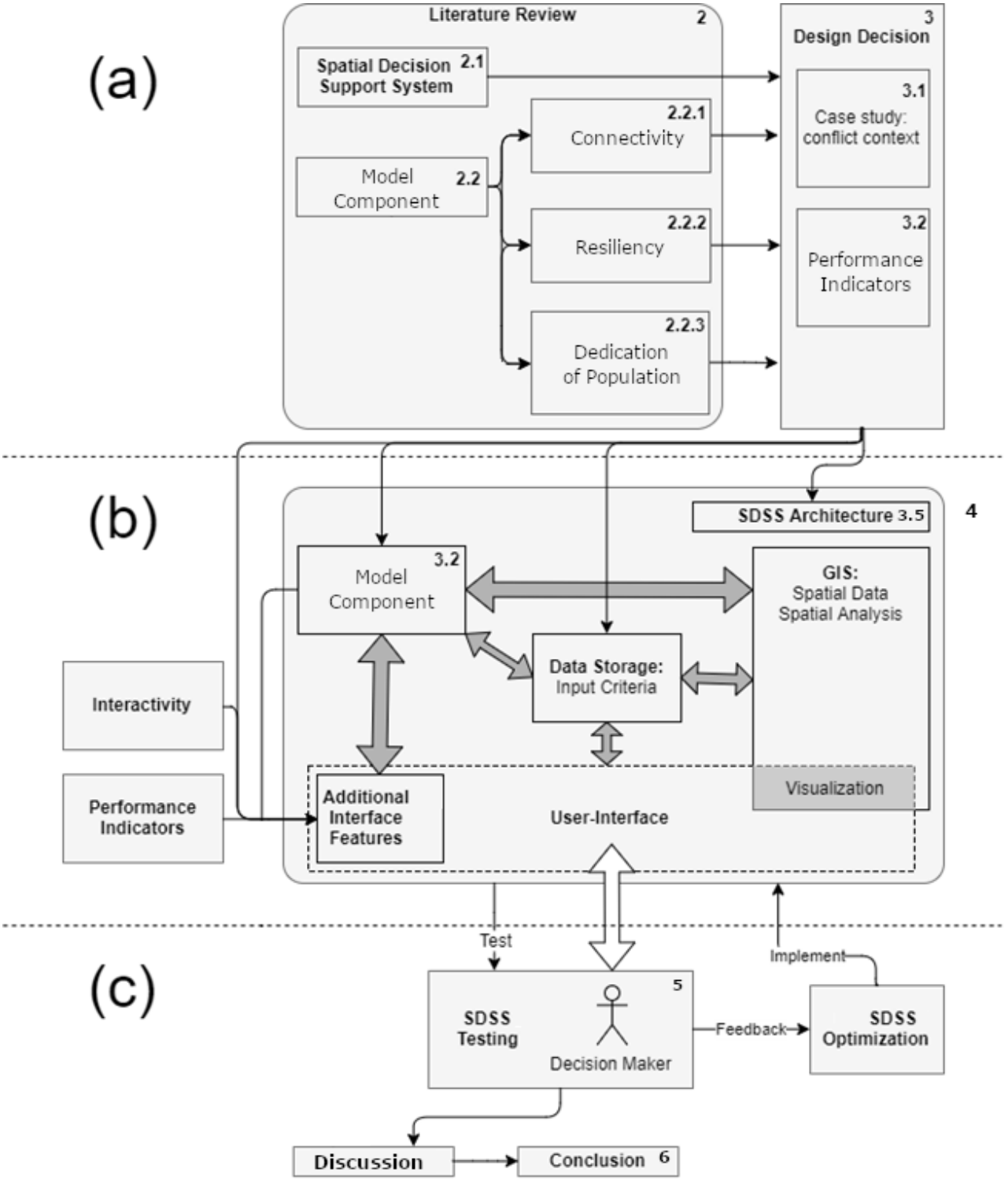


Figure 1.1: Research model consisting of phase; (a): Literature review and methodological decisions, (b): design of and implementation in the SDSS and (c): testing and evaluation.

Each of the above phases, consist of several steps. These are summarized in table 1.1 below and are further elaborated next.

Phase (a): Literature review & methodological decisions	**Phase (b): Design of and implementation in SDSS**	**Phase (c): Testing and optimization**
(a)1 SDSS exploration	*(b)1 Indicator preparation*	*(c)1 SDSS testing*
(a)2 Performance indicator evaluation	*(b)2 Development and visualization of the performance indicators*	*(c)2 SDSS optimization*
(a)3 SDSS design decision	*(b)3 Implementation of interactivity*	*(c)3 Discussion*
	(b)4 Development of the interface	*(c)4 Conclusion*

Table 1.1: Overview research phases

Phase (a): Literature review
The start of the literature review is purely explorative, just to get more knowledge about aspects that are important for the planning of field hospitals in a conflict situation. This has led to *step (a)1: SDSS exploration.* Here recent SDSS literature is explored and selected for relevance to the project framework of this research. Next comes *step (a)2 Performance indicator evaluation.* Here the most important performance indicators in the planning of field hospitals in a conflict are determined. Special attention is given to connectivity and resiliency indicators, looking at various measures e.g. one to evaluate the connectivity and resilience at different locations. Attention is also paid to how other measures can function as other performance indicators to see how certain field hospital distribution scenarios perform. The method for SDSS testing is also considered here. The literature review phase is ended with *step (a)3 SDSS design decision.* This step is the bridge to phase (b). Here choices are made on how the SDSS will be structured, what models are included in its model component, what performance indicators are to be provided to the SDSS taking into account the conflict context and the Mosul data, what interactivity exactly entails and what the interface will require.

Phase (b): Design and implementation of the SDSS
In phase (b) the MC-SDSS will be designed and implemented. It starts with *step (b)1 Indicator preparation.* Here a part of the interface is made which makes sure the right spatial layers for the indicators is being put in, also providing proper information on the relevance of the data. Next is *step (b)2 Development and visualization of the performance indicators.* Here the indicators that represent certain field hospital distribution scenarios are calculated and implemented. They are then included and clearly visualized into the interface of the SDSS to inform the decision-makers as good as possible. After that comes *step (b)3 Implementation of the interactivity,* in which adjustability of the (spatial) data is accommodated. Lastly in this phase is *step (b)4 Development of the interface.* Here a user-friendly interface with just the right number of tools and options is implemented. This is also where guidance is provided.

Phase (c): Testing and optimization
At phase (c), the first version of the SDSS will be finished. This enables various steps, to start with *Step (c)1 SDSS testing.* At this step, the support delivered by the SDSS to decision-makers will be tested by myself and independent test participants. Relevance, completeness, user friendliness and performance are among the elements to look at. *Step (c)2 SDSS optimization* uses the feedback from step (c)2 and is about implementing it in a next SDSS iteration. If necessary, more testing-optimizing iterations can be made. In *step (c)3 Discussion* test results will be discussed and possible future implementation of feedback is considered. Then lastly, *step (c)4 Conclusion* is used to reflect on the objectives of this research and their corresponding questions in an attempt to answer these. Limitations of this research and recommendations for future research are also given here.

1.5 Research Guide

This chapter provides a context to the planning of field hospitals in conflict areas and identifies research objectives and a framework to complete these objectives. The next chapter will review literature on SDSS design, methods for calculation of performance indicators and some literature on the testing of an SDSS. Chapter 3 then provides the Mosul case study, the actual used performance indicators, data used, requirements to the SDSS and different considerations for the design of the SDSS. Chapter 4 will guide the reader through the SDSS design process and different implementation iterations of the SDSS. In chapter 5 the whole process of testing different SDSS implementations is documented, including test set-ups, different testing roles, test script, the selection of test participants, the test outcomes and a discussion of these results. At last, the research objectives will be reflected upon in a concluding chapter 6, limitations to this research are acknowledged and recommendations will be made for future research.

2 THEORETICAL FRAMEWORK

In this chapter the theories regarding the proposed SDSS are considered. First the concept of an SDSS is elaborated further, also discussing its components and architecture. Then different models for performance indicators are reviewed. At last a short consideration is made on the testing methodology.

2.1 Spatial Decision Support System (SDSS)

Regular GIS software packages help address structured problems with a solidly defined solution process, like the running of a sequence of simple queries. But many issues are not so well structured or defined. A prescriptive process is especially absent in problem-solving when various players have a stake in the outcome as these players may fail or have trouble agreeing on the formulation of the problem. Such problems can still benefit from advanced analytical tools to explore the problem, run analyses on it and use the gained information to come to most optimal solutions. An SDSS is designed and developed to address these not so structured problems.

An SDSS is an interactive, computer-based system which can assist in the solving of semi-structured spatial problems (Sprague & Carlson, 1982). An SDSS has the possibility of bridging the gap between decision-makers and complex models, as spatially referenced information is combined with a decision-making environment in order to positively affect the performance of decision makers (Maniezzo, Mendes, & Paruccini, 1998). All the characteristics of an SDSS, according to Sugumaran & Degroote (2010) are summarized in figure 2.1. Carver (1991) argues that only the combination of advanced spatial analysis, the expertise of the decision-maker and a suitable user interface makes an SDSS different from a regular GIS. Carver specifically indicates that an SDSS in a decision committee room could create significant improvements in the way locational decisions are made. Jacko et al. (2003) prove that customized SDSSs that take into account the requirements and constraints of specific decision problems, are likely to result in more productive SDSS than generic GIS platforms. Stakeholder participation and consultation in the locational decision allow for more carrying capacity of decisions made and increases stakeholders' ability to provide useful feedback if it is done through the interaction with customized SDSSs (Carver, 1991).

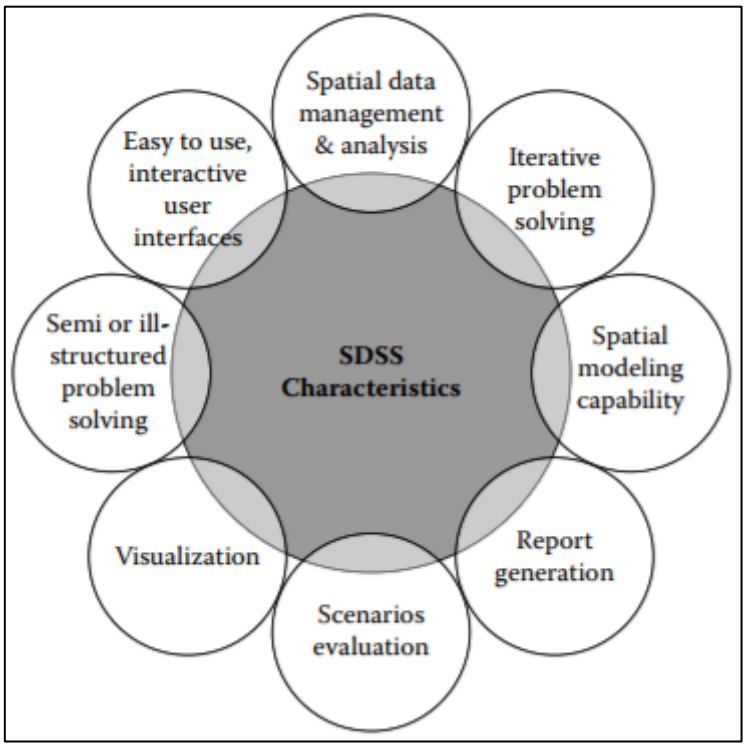


Figure 2.1: SDSS characteristics (Sugumaran & Degroote, 2010).

2.1.1 Components

Because an SDSS has to provide support for decision-making in semi-structured spatial problems, it should be easy to use, provide potential solutions through presentation of different scenarios, be flexible in use and possible to adapt and support analytical models. While an SDSS only has a database, model and user interface at its most basic level, more components are often required to achieve these characteristics. The number of components and level of detail vary in literature, but Sugumaran & Degroote (2010) identified a database management-, model-, dialog management- and stakeholder component crucial while they considered an application domain knowledge component optional (figure 2.2). What follows are short considerations of these different components. The model component is considered in more depth in section 2.3.

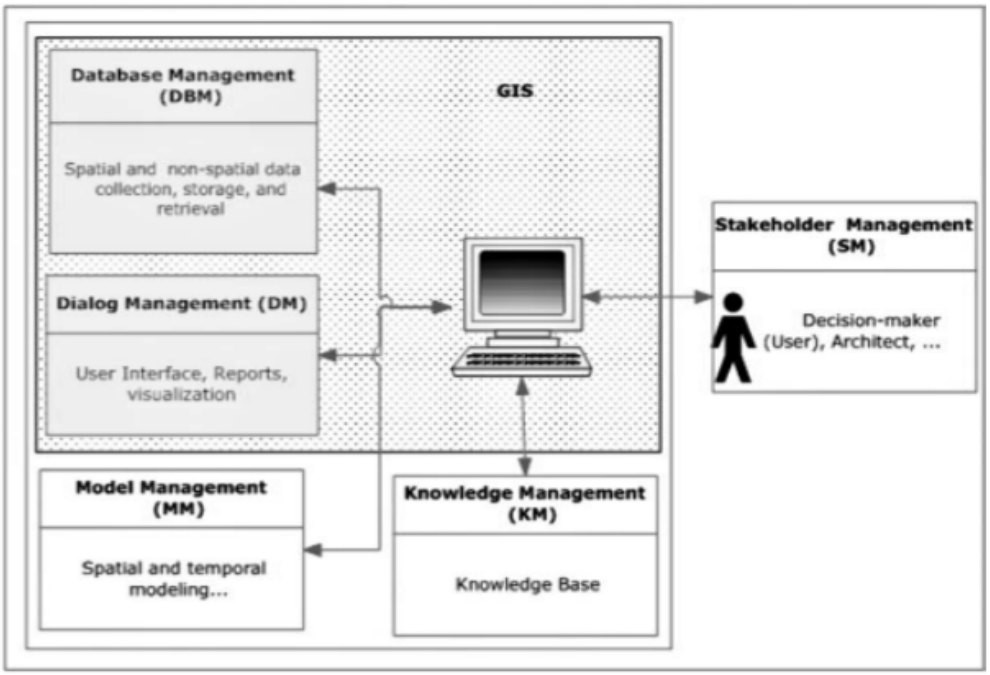


Figure 2.2: SDSS components identified by Sugumaran & Degroote (2010).

2.1.2 GIS and Database Management Systems

Most SDSSs are built around a GIS and use among others the database management and dialog management component of the GIS. The ability to store large amounts of (non)spatial data that is (in)directly linked to spatial features is one of the main strengths of a GIS. Besides the collection, storage and management of data, a GIS also provides functionality for the cartographic visualization of data. The database management component feeds data to the other components in an SDSS.

2.1.3 Stakeholder Management

In any spatial decision-making process, there are a number of stakeholders that have a stake in the potential outcomes of the process. The successful application of an SDSS depends on the involvement of various roles. Sugumaran & Degroote (2010) recognized various roles with different functions in the design, development, implementation and usage of an SDSS. These roles are that of the decision-maker, the analyst, developer and expert as can be seen in figure 2.3 below.

The expert is often a proponent for the SDSS, who acknowledges its potential value. The expert has detailed knowledge on necessary aspects of the spatial decision problem, being familiar with the possibilities that technology provides for the SDSS. Experts can also play a more specific role as they have knowledge on one or more aspects of the spatial decision problem. Either way, the expert provides unique knowledge and insight into the decision problem, which can be incorporated into the SDSS. The expert can work together with the other stakeholder roles to develop the necessary knowledge to and advice on how to make the best use of the SDSS. The expert could however not be aware of the precise context for a location decision in a given area. Strager and Rosenberger (2006) demonstrated such knowledge differences between groups of outside experts and local stakeholders in a spatial multi-criteria analysis.

The developer role includes tasks like collecting requirements for the SDSS from end users, designing the SDSS architecture, developing the user interfaces and functionality programming. Jankowski et al. (2006) stated that knowing user requirements for the SDSS leads to a system that can support decisions well. Therefore these requirements of stakeholders should be investigated before designing and developing the SDSS. According to Sahota and Jeffrey (2006), little employment of an SDSS can be caused by too limited involvement of users in the development of the software. Specific pitfalls that commonly occur are that tools are too time consuming to use, too complex and that there is too much uncertainty in outputs.

The analyst is usually involved in selecting models, running these and interpreting results, to inform the decision-maker. Because GIS-knowledge is often needed in an SDSS, the analyst might be a GIS analyst or someone familiar with the models. Usually, the larger the semi-structured spatial decision problem is, the more analysts from different disciplines are involved (Ascough et al. 2002).

Decision makers are the stakeholders who need meaningful information regarding the potential solutions to the spatial problem at hand. This information is usually provided through the SDSS by experts and analysts to aid their decision-making.

Although these four stakeholder roles in an SDSS are described separately, situations often occur in which the stakeholder has more than one role. This depends on the nature and size of the decision problem and stakeholders' levels of expertise.

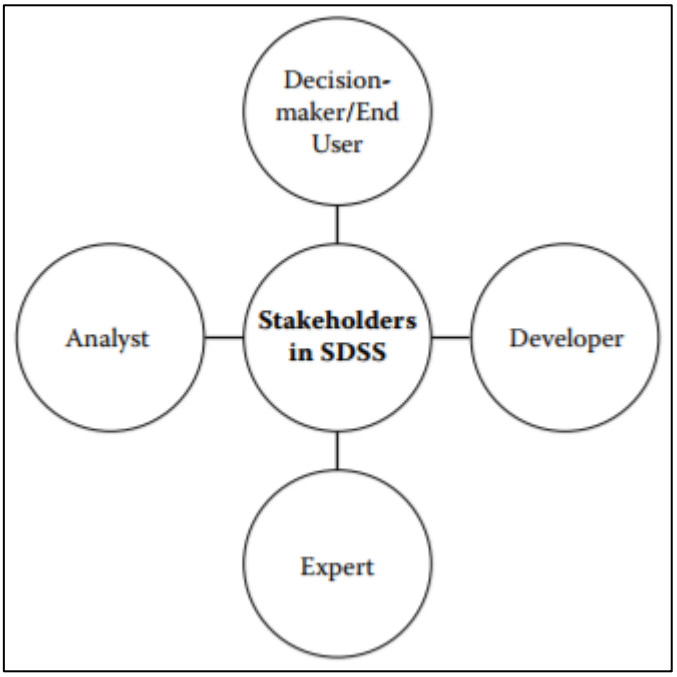


Figure 2.3: Stakeholders involved in SDSS (Sugumaran & Degroote, 2010).

2.1.4 Dialog Management

One of the most important aspects of a successful SDSS is the way users interact with the software in the system. This is called the dialog management component and it provides the user with an interface to operate, give input to and to get output from. The component offers the ability to represent outputs as meaningful maps and it gives the option to produce effective reports, tables and charts. Spatial decision-making processes involve iterative, interactive, and participative involvement of end users (Sugumaran & Degroote, 2010). This involvement is supported in the user interface of an SDSS, and it allows the user to compare potential solutions and their effects on a decision problem.

Malczewski (1999) summarized accessibility, flexibility, interactivity, ergonomic layout and processing-driven functionality to be considered for an effective user interface. Accessibility can be achieved by intuitive user interfaces that facilitate all the functionality for users. The possibility of changing the input on the fly and to undo incorrect actions makes the system flexible. A system will be interactive if

information flows efficiently between the user and the system. An ergonomic layout implies efficient communication between the user and the system. At last a processing-driven interface keeps the user up to date on what is going on and what has to happen next. A careful design of the system by user input and software testing can help in meeting these characteristics.

In the past decades there has been an increased realization that ease of use leads to more employment of an SDSS. As advanced as the system may be, if its interface is too complex, there is a good chance that few users are able to operate it. This stresses the need to meet the characteristics mentioned above.

2.1.5 Knowledge Management

The knowledge management component is often present in an SDSS, but it is not always essential to its functioning. Its purpose is to offer expert knowledge that can help in finding the best solution for a problem or to guide users through the decision-making process and the models included. According to Armstrong et al. (1990), there should be a library with expert knowledge as a guide for users, limiting their need of asking experts.

Zhu et al. (1996) divided the knowledge component into five categories: domain knowledge, model knowledge, utility program knowledge, metadata, and process knowledge. Domain knowledge is knowledge on the specific problem domain the SDSS is used for. The model knowledge describes models, helps to select the right models and their relevant criteria. Information about tangential tools is provided by utility program knowledge. The metadata gives information about the data used in the SDSS and the process knowledge guides through the decision-making process using the SDSS.

Elmes and Cai (1992) made an SDSS for improving by moth affected ecosystems including a visual tutorial in its knowledge component to guide in selecting inputs, operations and data characteristics required for spatial analysis. Witlox (2005) states that only specialized knowledge which is based on true expert input should be included and that the knowledge should not be trivial or overly complicated.

2.1.6 Architecture

It is now clear how an SDSS requires various components that serve multiple purposes. There basically are two ways to provide all the functions in an SDSS. Either by linking software together in a comprehensive system or by developing all the functions in a single software through existing tools or by developing customized tools.

Linking software might be necessary because there are few software applications that provide all required functions. Most SDSSs have been developed this way. This is made possible by creating an interface or intermediary program that takes care of the data transfer between various pieces of software. This model saves time as it makes use of work already done, but it often reduces control of the development design and it possibly decreases flexibility of the SDSS.

Developing all the functionality in a single software can be difficult as it might lack functionality in its original form and so a lot might have to be developed. But when this approach is chosen for the development of an SDSS, it is usually done in a GIS as they in general meet the majority of the requirements for an SDSS. Usually, spatial data analysis, modelling, management and other functionality like visualization and customization of interfaces are present in GIS.

The way in which all of the functionality in an SDSS is combined, can be categorized in a spectrum from no coupling, to loosely coupled, to tightly coupled, to full integration within a single software (Chakhar and Mousseau, 2008). Without coupling, different programs have to be accessed by the user to achieve the intended functionality. With loose coupling, there is some automated interaction between the different software components used in the SDSS. Tight coupling is the same as loose coupling except that all functionality can be accessed from a single interface. Figure 2.4 presents the coupling approaches proposed by Chakhar and Mousseau (2008) in regard to GIS and multicriteria analysis software.

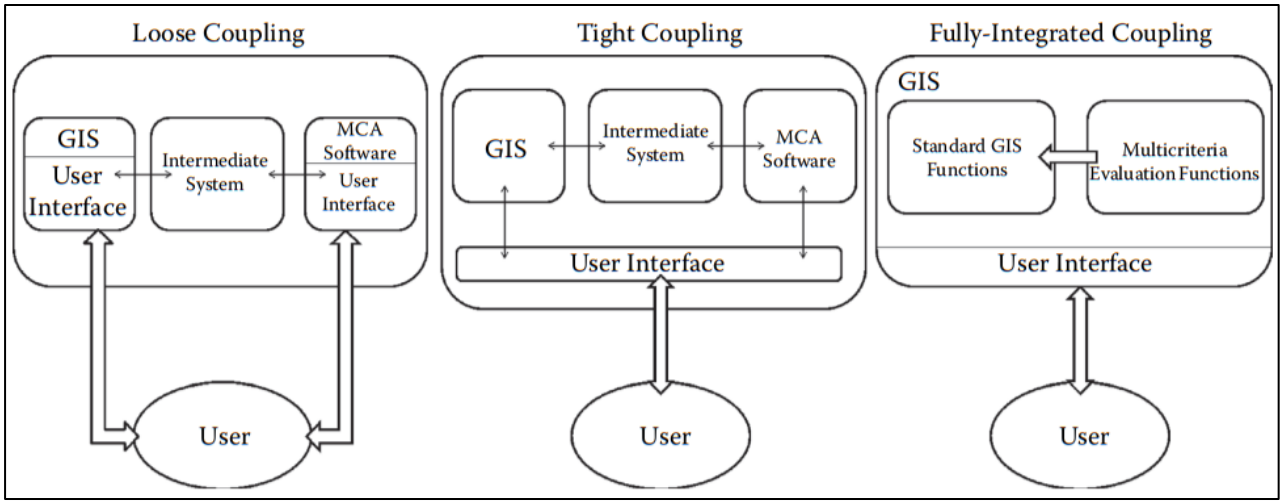


Figure 2.4: Diagrams representing loose, tight, and fully integrated coupling approaches (Chakhar and Mousseau, 2008).

Coutinho-Rodrigues, Simão, & Antunes (2011) have developed an SDSS which integrates MCDMs for the purpose of planning urban infrastructure systems. It is inspired by the fully integrated coupling framework of Chakhar & Mousseau which is shown above (2008). Their SDSS builds on existing GIS functionality, uses a customized user-friendly interface, uses various MCDMS and includes interactive functionality to experiment with these models, in order to assist their aimed for decision-makers as good as possible. *Figure 2.5* below shows the architecture of their SDSS. It serves as an example for the architecture of this research and is further discussed in section 3.5.

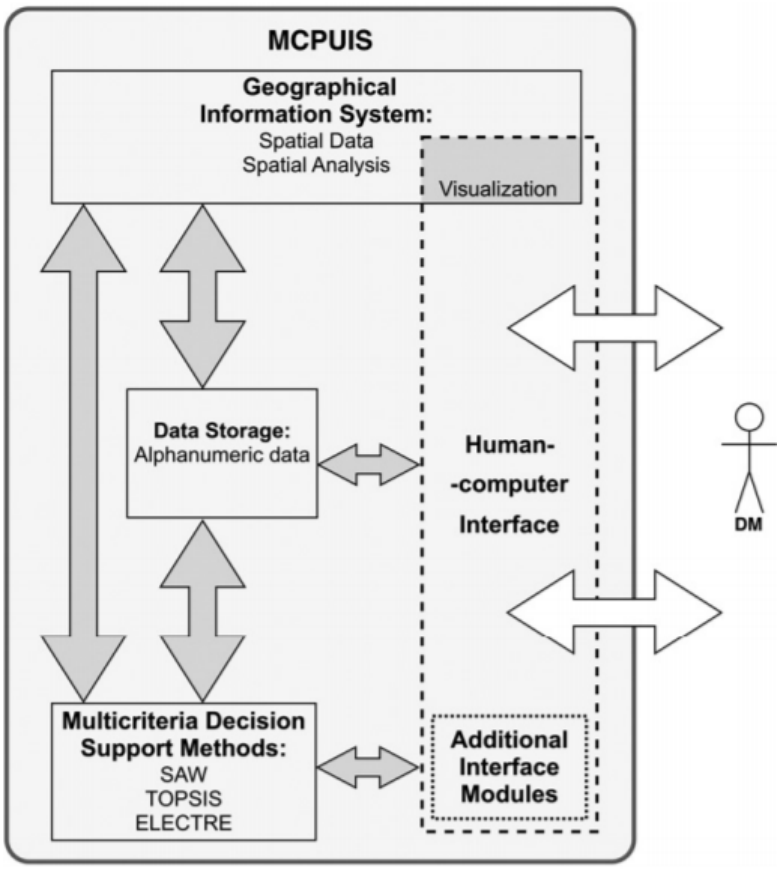


Figure 2.5: Multicriteria Planning of Urban Infrastructure Systems Architecture (Coutinho-Rodrigues et al., 2011).

2.2 Model Management for Performance Indicators

The model management component of an SDSS takes care of all the different models that have to be executed and possibly combined. It contains a certain set of models that provide analytic capabilities to spatial datasets. The spatial models translate the spatial data to more meaningful information for decision-making processes. Next, the need for models in this SDSS is discussed and different models are explored.

The problem of deciding where to place field hospitals is a typical location decision problem which involves various conflicting criteria, making it a multi-criteria decision-making problem (MCDM). Malczewski (2004) states that while there is a wide variety of decision models, there is a lack of rules to decide which to use for what situation. In this research, various models are needed to calculate the various performance indicators. But no existing MCDM is used to combine these performance indicators and propose ideal locations to the stakeholders. Because picking locations for field hospitals in conflict areas is a too complex process to just prescribe locations based on criteria and weights. Instead, a custom MCDM is proposed in which stakeholders are presented with clear visualizations of different performance criteria and combine this information with their expert knowledge to explore the solution space and come to the most optimal locations by exchanging their knowledge between themselves and the SDSS. The stakeholders will be allowed to move candidate locations around on a map to evaluate the impact a location change has on the provided performance indicators. Therefore some theoretical underpinning is provided next for the most important performance indicators in the SDSS. These are the connectivity, resilience and served population by a hospital location. The other performance indicators do not require theoretical underpinning because they are much less complex. Decisions upon the use of certain models are made in chapter 3 on methodology.

2.2.1 Connectivity

The connectivity of a location can be of great importance in deciding where to place field hospitals. Roads give insight into the connectivity between places, as places that might be near one another in Euclidean distance, could be much further apart through a road network. Connectivity is often expressed in some measurement calculated through a road network. Experience learns however, that road network datasets can be difficult and time consuming to generate for conflict areas. Calculations with road networks can be time-consuming as well. Therefore both measurements through a road network and a less complicated measurement are discussed next for expressing connectivity.

One method of expressing connectivity is through centrality measures. Their algorithms are often already available in GIS platforms and they can identify the most important vertices in a road network and provide the ability to rank them (Borgatti, 2005).

Closeness centrality

Closeness centrality of a vertex is the average length of the shortest path between a vertex and all other vertices in a network. The more central the vertex is in the network, the closer it is on average to every other vertex. Sabidussi (1966) defines closeness centrality as:

$$C(x) = \frac{1}{\sum_y d(y,x)} \tag{1}$$

where $d(y,x)$ is the distance between vertices y and x.

Betweenness centrality

Betweenness centrality is how often a node in a network is passed if you look at the shortest routes of any point to any other point in a network. In other words, the higher this centrality, the more likely you are to pass a node when taking a shortest route in the network. Brandes (2001) defines betweenness centrality as:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2}$$

where $\frac{\sigma_{st}(v)}{\sigma_{st}}$ is the fraction of shortest routes between s and t that pass node v.

While closeness and betweenness centrality are quite similar (Brandes, Borgatti & Freeman, 2016), closeness centrality is more susceptible to outliers in the network. One exceptionally single road heading out of the network will shift the closeness centrality heavily towards this road. Betweenness centrality is less susceptible to this because it is more of a 'local' centrality whereas closeness is more 'global' (see figure 2.6 below). Closeness centrality sort of counts a node in the middle of the network as central, while betweenness centrality often classifies major roads as central because they are passed much.

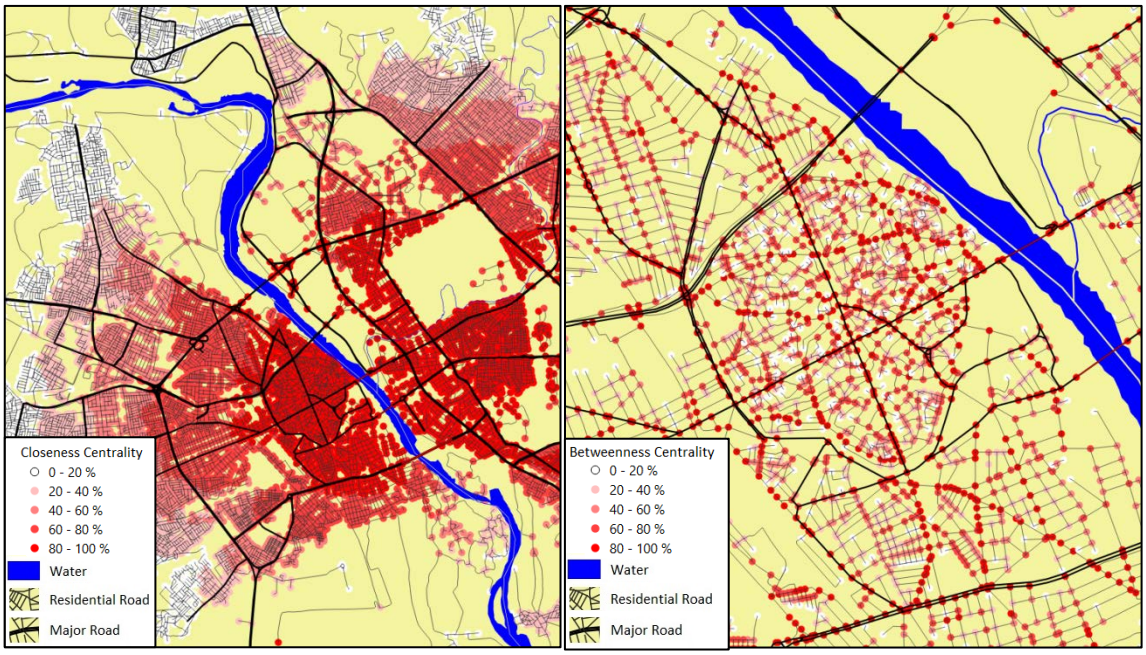


Figure 2.6: closeness centrality (left) and betweenness centrality (right).

Custom connectivity

While these centrality measures could tell us something about the connectivity of a vertex in a network, a similar method might be needed to measure connectivity without a network. The ratio of major- to residential road length reached within a small search window from the given vertex can substitute when there is no network to be traversed through. To gain a sense of the centrality in a wider area of a hospital, a small averaging search window around a hospital could be used as well.

2.2.2 Resiliency

A field hospital location that can easily be cut off from demand sources (population using the hospital), is undesirable in a conflict situation. A demolished road or bridge for example can make a location unreachable. There are measurements for the resiliency of a location. Generally, the most resilient locations are those that will remain available longest to most people, when an increasing number of routes become unavailable. A location in the middle of a segment in a street network for example only has 2 directions to be accessed from; the two directions in the street. A location at an intersection of 8 roads will have a much smaller chance of being disconnected from the rest of the street network, although it should be kept in mind that theoretically 7 out of eight roads could be a dead end, which leaves only one road connecting the location to the remainder of the network. There are different ways for measuring the resiliency of a location, two will be discussed in the next paragraphs. The redundancy index is considered for situations where there is a road network dataset available and when calculation speed is no issue and the degree centrality measurement for when there is just a road dataset or a need for fast calculation.

Redundancy index

The importance of redundancy as a possible measure for resiliency has been stressed in the project framework of this research. Redundancy in networks is vital for providing utility during disruptions, such as when a road segment in a network is blocked or damaged. Redundancy, in this case, provides alternative routes to get to a certain point. The redundancy index, as described by City Form Lab (2016), can approximate the redundancy of paths to get to a certain vertex from any other vertex

in a network. The path redundancy index calculates for a vertex in a network both the shortest path and the alternative paths to each of all the other vertices, given a certain redundancy coefficient. Then the ratio between the summed length of the alternative paths and the shortest path is calculated. Doing so for a certain vertex to each of all the other vertices in a network and taking the average of these ratios provides the redundancy index of a vertex. The redundancy coefficient indicates how much longer the alternative paths are allowed to be compared to the shortest path.

Given a pair of vertices O and D in an undirected graph G, the redundancy ratio for the pair with a redundancy coefficient ρ ≥ 1 is defined by (City Form Lab, 2016):

$$\mathrm{R}^{\rho}[\mathrm{O},\mathrm{D}] = \frac{\sum_{\mathrm{e}\in\mathrm{E}}\mathrm{w}[\mathrm{e}]\cdot\zeta^{\mathrm{O,D}}[\mathrm{e},\rho\cdot\mathrm{d}[\mathrm{O},\mathrm{D}]]}{\sum_{\mathrm{e}\in\mathrm{E}}\mathrm{w}[\mathrm{e}]\cdot\zeta^{\mathrm{O,D}}[\mathrm{e}\cdot\mathrm{d}[\mathrm{O},\mathrm{D}]]} \tag{3}$$

where w[e] is the weight of edge e in G; d[O,D] is the shortest path distance from O to D in G; and ζ O,D[e, x] is 1 if there is any path (not necessarily simple) from O to D in G that goes through edge e and whose weight is at most x, and 0 otherwise. A path is simple when it does not cross a vertex more than once. The enumerator of the ratio is the summed lengths of the alternative paths and the denominator that for the shortest path between two vertices.

The redundancy index is calculated by taking the average of the ratios between the vertex to all other vertices. Given one origin O and a non-empty set of destinations Δ, the redundancy index for O with a redundancy ratio ρ ≥ 1 is defined as (City Form Lab, 2016):

$$R^{\rho}[O,\Delta] = \frac{\sum_{\mathrm{D}\in\Delta}\mathrm{R}^{\rho}[\mathrm{O},\mathrm{D}]}{|\Delta|} \tag{4}$$

where |Δ| is the size of the set Δ, i.e. the number of destinations. The redundancy index is relatively fast and can be calculated for all the vertices in a network at once. The redundancy index is an approximation of the true amount of redundant path length and it usually gets quite close to this true amount. Although it is relatively fast because of its approximation, processing time can quickly add up for large networks and higher redundancy coefficients. The higher the redundancy coefficient, the more computational power is needed to calculate the index. When a redundancy coefficient of 1.1 is chosen, then this means that the redundancy index is calculated for routes 10% longer than the shortest path. If this, for example, gives an index of 5.5, this means that for this coefficient on average 5.5 times more alternative street length becomes available to reach another vertex.

Degree centrality

Besides the earlier discussed closeness and betweenness centralities, another centrality measurement is degree centrality. It is the oldest and conceptually simplest centrality measure. Degree centrality is measured by the number of streets that a vertex or intersection is directly connected to. The more degrees that are connected to the vertex, the less likely it is that a vertex will be cut-off from the rest of the network. The degree centrality of a vertex v, for a give graph $G := (V, E)$ with $|V|$ vertices and $|E|$ edges, is defined as (Freeman, 1978):

$$C_D(v) = \deg(v) \tag{5}$$

The degree centrality calculation is much less requiring than the redundancy index, but a lot less accurate because it only takes directly connected road segments into consideration.

Custom resiliency

While degree centrality could tell us something about the resiliency of a vertex in a network, a similar way is needed to measure resiliency without calculations through a road network. Because as previously mentioned, proper road network data is not always available in a conflict area. The number of roads reached within a very small search window from the given vertex can substitute when there is no network to be traversed through. To gain a sense of the degree centrality in a wider area of a hospital, a small search window around a hospital could be used as well.

2.2.3 Dedication of Population

Location decisions aim to optimize at least one objective function. This could for example be the minimization of facilities, maximizing spatial coverage or minimizing the average distance traveled

from demand to facility (also known as P-median) (Farahani et al., 2010). Daskin & Dean (2004) have reviewed these objective functions with regard to health care facilities and concluded that the maximization of coverage and the minimization of the average distance are the most efficient objectives in the context of planning public facilities. Of these objectives, the minimization of distance leads to the most equitable hospital location pattern (Morrill & Symons, 1977) as it tends to equalize the distance travelled among the target population. This minimization of total distance between population and hospitals is further addressed in the performance indicator definition in section 3.2 and more in-depth discussion of objective functions is performed by Rahman & Smith (2000).

In the case of field hospital location decisions, it often is the case that not enough hospital capacity is available to serve all of the demand from the population. Therefore it is assumed that the first and foremost objective function in the context of this SDSS is to make sure that the available hospital capacity is used to full extent. The stakeholders using this SDSS can evaluate this objective function through a served population performance indicator. When multiple hospitals are present on a map in the SDSS, it has to be determined which hospital serves what area and corresponding population to calculate this performance indicator. There are different methods to determine this and next, we will discuss three methods. These methods differ in whether it is possible to calculate areas that are proportional to some weight given to the hospitals. As it might, for example, be desirable to model that more people travel to a big existing hospital rather than equally towards field hospitals. The methods also range from complex and more accurate to simple and less accurate, which translates into a trade-off between accuracy and computation time.

Voronoi diagram

Probably the simplest method for the division of space between hospitals is the use of a Voronoi diagram. A Voronoi diagram partitions the space around the hospitals into Voronoi polygons.
A Voronoi polygon consists of the area that is closer to the corresponding hospital than to any to any other hospital. Where distance to two or more hospital is equal, the boundaries of the Voronoi polygons will be located. The Voronoi diagram is calculated in Euclidean space without taking roads and obstacles into consideration (see figure 2.7 below). In exchange of this simplicity, the calculation of a Voronoi diagram is rather fast and of course it does not require a road network for its calculations.

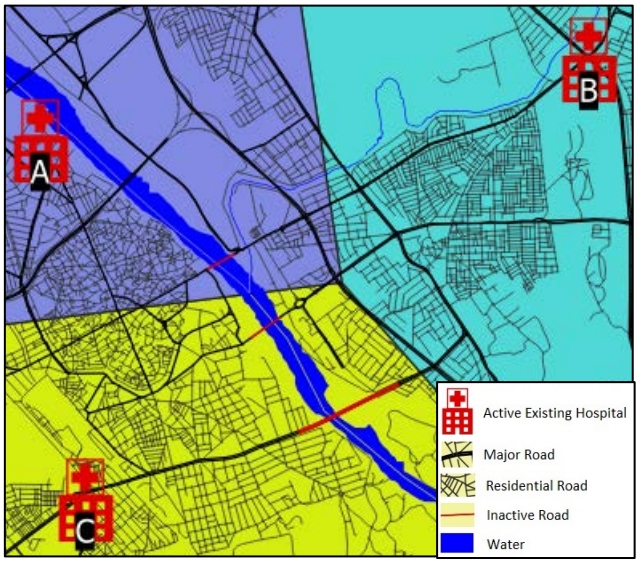


Figure 2.7: The hospital Voronoi polygons of the Voronoi diagram do not take (in)active roads and obstacles into account.

Custom raster areas

Instead of dedicating population to hospitals through Euclidean vector space, a raster-based approach might be computationally similarly requiring and more flexible. For each hospital, a raster distance map could be calculated, such as in figure 2.8 a below. Then the distance raster of each hospital is compared, and for each cell, the hospital distance map with the smallest distance in that cell will be the output cell (GRASS Development Team, 2019). This results in a division of area like in figure 2.8 b below, and is almost the same as the Voronoi diagram. Except with this method, you could set weights to individual hospitals by applying this weight to every cell of a hospital's distance

raster. It, for example, could be desirable to assign hospital capacity as weights because bigger hospitals attract more population. In figure 2.8 c below, the hospital in the yellow area has had a reduced weight applied to its distance raster. This has resulting in the hospitals of the green and purple areas getting more area.

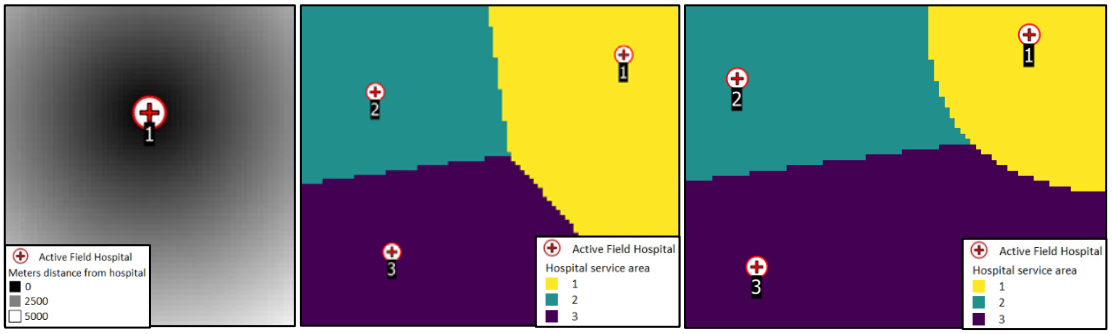


Figure 2.8: (a) individual hospital distance raster, (b) custom raster area division (equal weights) and (c) weighted raster area division (hospital 1 has a weight of 0.75 instead of 1).

This custom raster area method could be extended by using raster path distance allocation algorithms. Such an algorithm calculates for each cell in a raster which of the target cells (hospitals) has the shortest path, taking the cost of each traversed raster cell into account. With a fine enough raster, a vector roads file could be rasterized to act as a cost surface. The resulting raster would hold cells on locations of roads which have no traversing cost and all roadless areas would be impassable. With this method, residential roads could be given a small traversing cost to reflect upon the difference between major and residential roads. Unlike the previously discussed Voronoi diagram, this method accounts for the real-life restriction of having streets to travel through and obstacles to avoid. Without such an implementation of a cost surface, inactive bridges in a road file will not have any effect on the reach of a hospital (figure 2.9 below) and the raster approach would only differ from Voronoi diagram in the ability to set weights. This method also would not require a road network file.

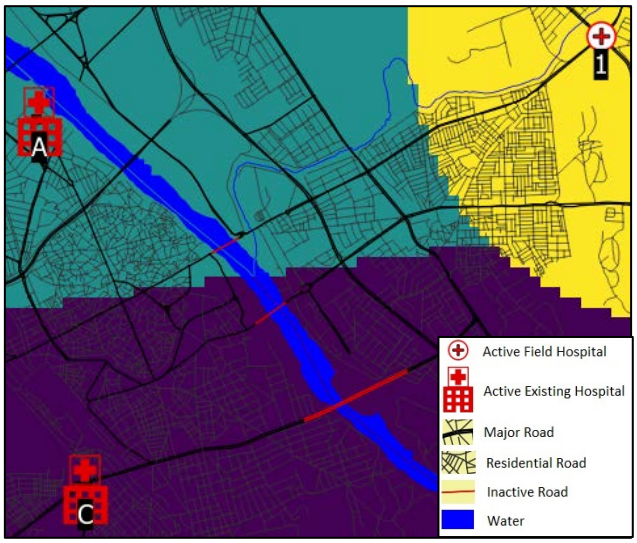


Figure 2.9: Custom raster areas without raster path distance allocation.

Service area allocation

In (vector) service area allocation, for each hospital, a road network is traversed to calculate how far each road segment is from each hospital. Then each road segment gets assigned the hospital identifier of the hospital that is least far away from road segment (GRASS Development Team, 2019). Polygons can then be drawn around subsections of the network to indicate which hospital serves what area. This computation can be somewhat time-consuming and requires a road network. In exchange, it provides the ability to add traversing costs to road segments and junctions. Also it is possible to weight the reach of each hospital. Figure 2.10 below shows how hospital service areas reach over water through bridges, but do not anymore if bridges are toggled inactive.

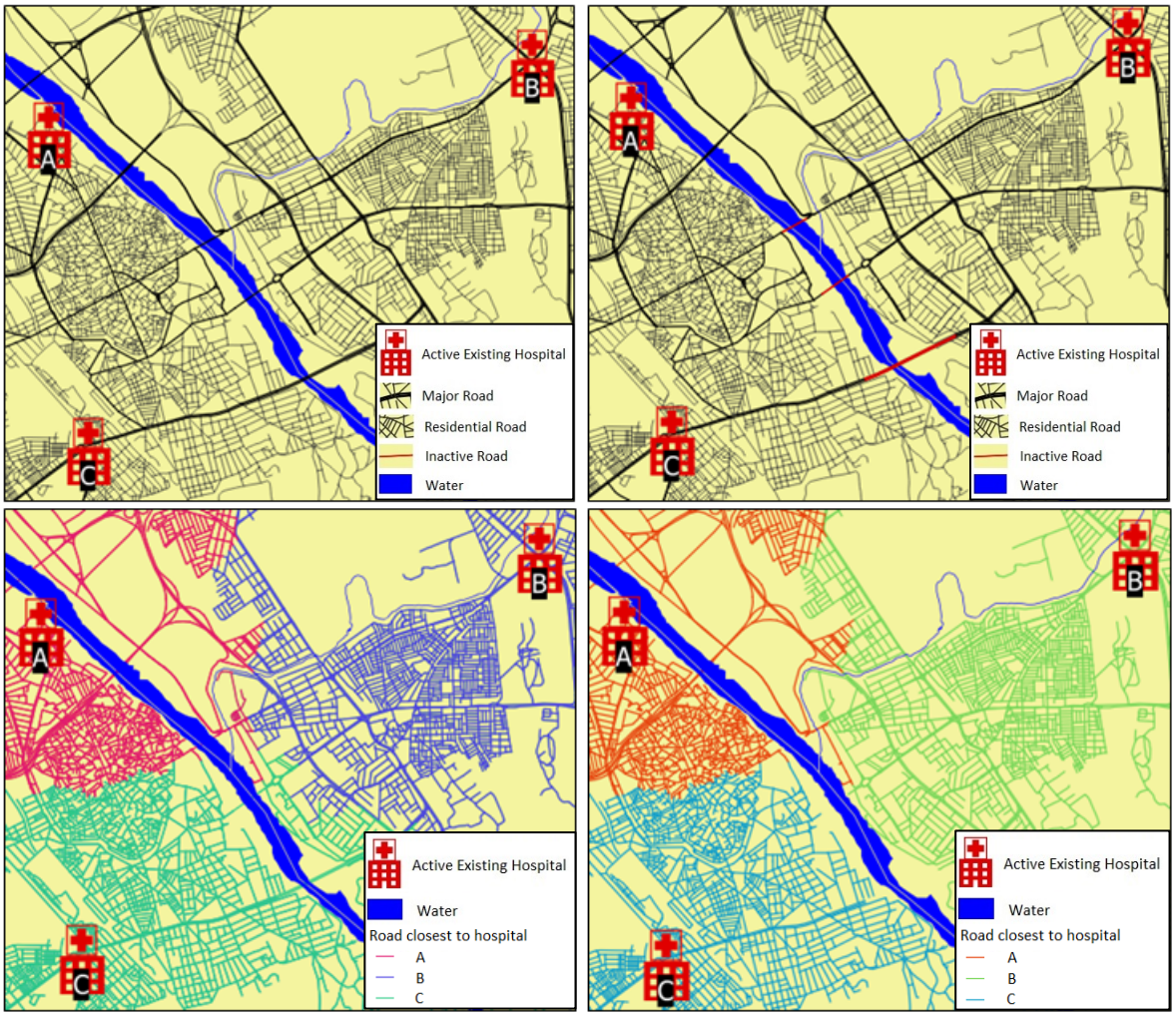


Figure 2.10: (top left) active bridges and (top right) inactive bridges affect the allocation of hospital service areas through a road network (bottom left shows service area with active bridges, bottom right with inactive bridges).

3 METHODOLOGY

3.1 Case Study Area

Mosul is a major city in northern Iraq with an estimated population of between 750.000 and 1.500.000 in its urban area after approximately half a million fled the city in the second half of 2014 because of fighting between ISIL and government forces (UNAMI, 2014). It is located some 400km north of Baghdad, and its old city stands on the west bank of the river Tigris. At the time of the liberation operation, the five bridges over the Tigris have reportedly been destroyed by ISIL forces (Tawfeeq, 2017), and pontoon bridges were the only connections between the west and east part of the city (MacSwan, 2017). In 2017, many of the refugees resided in refugee camps surrounding the city. Besides bridges, many roads and buildings are damaged as well (see figure 3.1) (BBC, 2017).

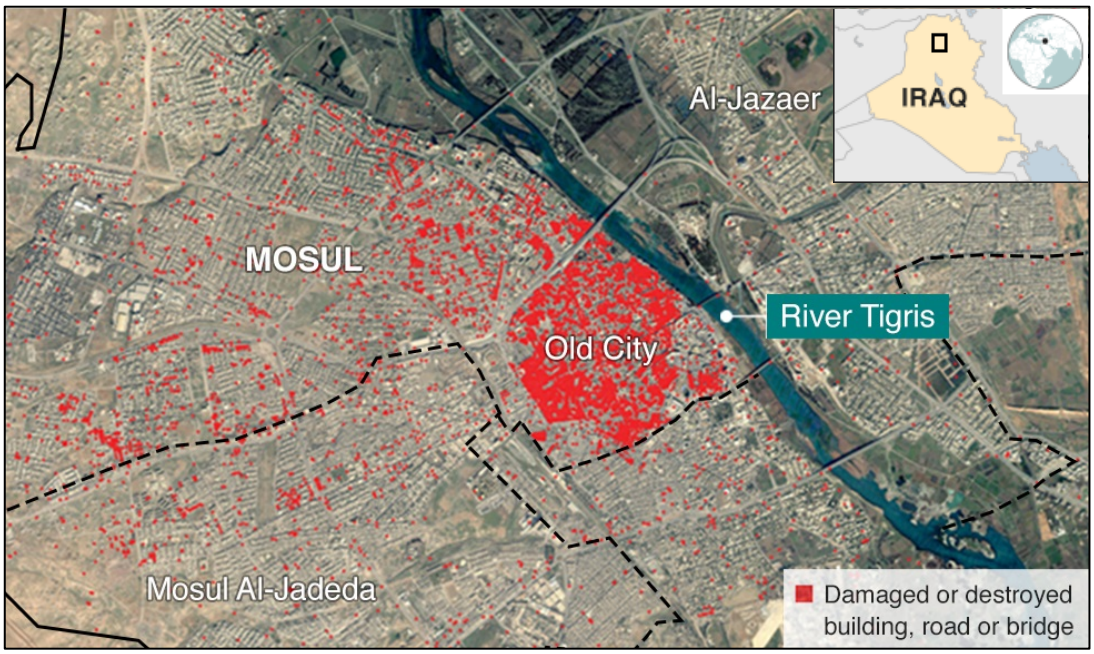


Figure 3.1: Mosul damage assessment on 8 July 2017 (BBC, 2017).

3.2 Performance Indicators in Model Component

What follows is a consideration of the various performance indicators that need to be implemented into the model management of the SDSS, to allow stakeholders to come to the best locations for field hospitals in a conflict situation. We can distinguish between two types of performance indicators, more static ones and quite dynamic ones. The more static ones are calculated only once per session, unless the road and dangerous situations happen during the decision-making process itself. It is expected that at the beginning of a planning session, or a bit prior to, updates are made by the stakeholders to the map objects in order to reflect upon any changes that happened in the conflict dynamics. These static indicators, therefore, are allowed to take longer to calculate. The dynamic indicators will have to be calculated after every shift in objects on the SDSS map. The move of a hospital changes every performance indicator as all of the indicator values are determined by the locations of the hospitals. The difference between the connectivity- and resilience- versus the other indicators is that most of the connectivity and resilience calculations do not have to be repeated at each change, unless roads are toggled (in)active or if an adjustment has been made to the danger(s) on the map. An overview of the performance indicators can be found below in table 3.1. The table contains the indicator name, the main method that is used for the calculation of the indicator, the data that is used in this calculation and the number of methods that were used subsequently in the calculation of an indicator. What follows is a consideration of each of the performance indicators. A fully implemented schematic of the SDSS model component can be found in section 4.2.2 on the first implementation of the SDSS.

Indicator name	Main method	Data used	Nr. of methods
Population	Voronoi diagram	Population Existing hospitals Field Hospitals Danger	3-5
Mean Population Distance	Distance	Population Existing hospitals Field Hospitals Danger	2
Danger Distance	Distance	Existing hospitals Field hospitals Danger	1
Water Distance	Distance	Existing hospitals Field hospitals Water	1
Area Covered	Voronoi diagram	Existing hospitals Field hospitals	3-5
Connectivity	Betweenness centrality	Existing hospitals Field hospitals Roads Danger	3-4
Resilience	Degree centrality	Existing hospitals Field hospitals Roads Danger	3-4

Table 3.1: Overview of field hospital planning performance indicators.

3.2.1 Static Indicators

The accessibility of a location is by many considered the most important factor in the supply of health care (Moradian et al., 2017). The same is assumed for field hospitals. As previously discussed, accessibility is measured in this research as connectivity and resilience.

Connectivity

A comparison between different connectivity measures has been made in section 2.2.1. For the connectivity performance indicator of the SDSS, a custom betweenness centrality measure will be used. First of all betweenness is being used instead of closeness centrality because of the local/global difference that has been showcased. It is desired that the connectivity measure can also locally, outside of the middle of a network, express centrality. By using the betweenness centrality, major roads (also outside of the middle of the road network) will have high centrality values. In real life, this would likely also be the case. Betweenness centrality is calculated for every junction in the road network, and our connectivity indicator will be the average betweenness centrality in a 200 meter window around each hospital.

Resilience

A redundancy index, degree centrality and a custom resiliency measure have been discussed in section 2.2.2. It has been decided to implement a custom degree centrality measure. While the redundancy seemed a more accurate measure of resilience because it took the whole network into consideration, the calculation of this measure quickly takes too long. Therefore the much faster degree centrality is chosen. And to mitigate the loss of accuracy because it only looks at direct resilience, the resilience indicator represents the mean of all degree centrality in a 200 meter window around each hospital.

3.2.2 Dynamic Indicators

Population

Besides permanent population there are two other groups that are probably most relevant for field hospitals; Internally Displaced Persons (IDPs) and non-camp IDPs. IDPs are better known as refugees who reside in camps and non-camp IDPs are persons who reside somewhere else than their own home or a camp, such as an improvised home or at a host.

Of the three discussed methods to dedicate demand to certain hospitals in section 2.2.3, the Voronoi diagram is chosen because of its simplicity. These dynamic indicators need to be calculated very

often in a planning process and their updating needs to be near immediate. The vector and raster based service areas are promising because of their increased accuracy, but take up to a minute to be calculated in this case study.

The demand that needs to be covered is assumed to be the population data that is loaded into the SDSS. If this demand is not yet expressed in data points, the centroids will be taken from the features that do represent the demand. The sum of demand of these points that fall within a single hospital Voronoi polygon is considered to be the served population by a hospital.

Mean population distance
In section 2.2.3 it was concluded that the minimization of the average distance to health facilities leads to the most equitable hospital distribution. Since health equity is important to the WHO, it is decided to provide a mean population distance performance indicator. This is the average distance of the population points in a hospital's Voronoi polygon to the hospital.

Danger distance
Of course a field hospital should not be placed in enemy territory or areas which involve great risk. Areas of great risk could be locations that are likely to be attacked. For example, government buildings, military bases or universities. Or the military could already have plans to attack somewhere, making the area more dangerous. The danger distance performance indicator is the distance of a hospital to the nearest danger.

Water distance
Rydén argues that in the strategic placing of field hospitals in a disaster hit country, water accesibility plays a role (Rydén, 2011). Water could be scarce and therefore it would be somewhat beneficial to place field hospitals near water sources. Even dirty water sources can be beneficial because field hospitals are provided with water filters if necessary. Also, areas of water can be dismissed as candidate locations, taking into account that bridges theoretically could be a location for a field hospital. The water distance performance indicator is the distance of a hospital to the nearest danger.

Area covered
Besides the minimization of average distance to health facilities, the maximization of coverage was another objective function that was considered important in section 2.2.3. Therefore the covered area is also provided as a performance indicator. It is derived from the size of a hospital's Voronoi polygon.

3.3 Data

Data for the Mosul case study is provided by the WHO and is on Northern Iraq.. These are several vector- and raster data layers. Of this provided data, extractions have been made for testing purposes of the SDSS. The extractions are based on a custom extent which can roughly be seen in figure 3.2 below. Both the provided and testing data is listed in table 3.2. Next, the different data is discussed.

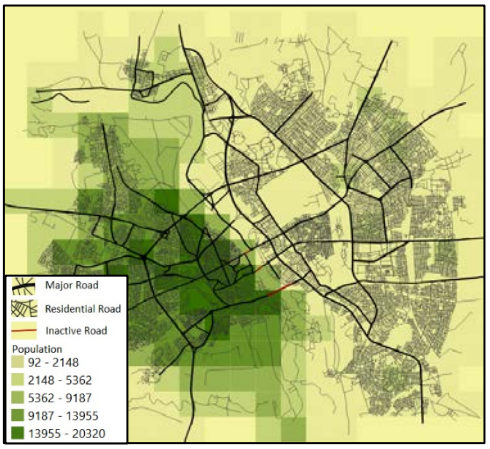


Figure 3.2: custom Mosul data extent (the frame of the figure).

Population
The demand for field hospitals is determined by a polygon distribution of total population in Northern Iraq. It consists of all the population, so IDP's and non-camp IDP's are included.

Roads
The road data was derived from OpenStreetMap. The roads initially were discontinued in small settlements, due to incorrect geometries or missing data. Therefore the roads have been connected in those places as it is assumed that they can be traveled. Also, a network has been made of the road data by fixing connectivity issues.

Existing hospitals
Existing hospitals are an important factor in the determination of field hospital locations as they already serve a certain demand (Moradian et al., 2017). In the SDSS, any provided field and existing hospitals are merged into a single file for processing speed, but they are still differently identified (field hospitals by numbers and existing hospitals by letters).

Field hospitals
Field hospitals can optionally be loaded into the SDSS if any are already operating. If none is loaded, a new file will be created in which field hospitals can be added.

Enemy Territory
The enemy territory is demarcated by danger polygons which is last updated November 24 2017. The enemy/dangerous territory was among others in the Northwest of Mosul at the time of creation. Only this part has been used as danger for the test case.

Water
Only the main water barriers in Northern Iraq have been included in the data on water. These are big rivers and lakes, including the Tigris river which streams through Mosul. The data consists of polygons representing these water bodies.

OUTPUT NAME	GEOMETRY	INPUT NAME	GEOMETRY
Test population	Polygon	TOTAL_POP	Polygon
Test roads	Network	Road_infra_ND	Network
Test existing hospitals	Point	Health_Facilities_24_Nov	Point
Test field hospitals	Point	-	-
Test water	Polygon	Water barriers	Polygon
Test danger	Polygon	Situation_Nov24	Polygon

Table 3.2: Mosul data summary, test data (left) and original Mosul data (right).

3.4 Stakeholder Requirements

As was mentioned in the theoretical background, a single stakeholder can have various roles towards the SDSS. In this SDSS, I am the expert and developer whereas the end user is the analyst and decision-maker. The decision-maker could however not possess the skills of the analyst, but the knowledge base, the combination of different algorithms into the model component and the ease of the SDSS will guide the decision-maker through the process of decision-making using the SDSS. Some decision-makers could also possess expert knowledge, which other decision-makers do not possess. This knowledge could also be added to the knowledge base in future iterations to aid the decision-maker.

Sugumaran & Degroote (2010) and other authors stressed the importance of involving all the stakeholders in the development of the SDSS. Ideally, a user needs analysis was performed among potential users of the SDSS, but it proved to be difficult to access real potential users. A general issue in user needs analysis is that users limitedly share their real needs, because they do not know what is technically possible (Belkin, 2000). Therefore an approach in which expected user requirements are used to design an SDSS could prove an academically interesting case.

Between testing of the SDSS, feedback is considered to be new requirements to the SDSS and is therefore implemented in the design of the SDSS. What follows are the expected requirements for the SDSS in table 3.3, categorized by the SDSS components which were introduced in section 2.1.

Dialog Component	Database (GIS) Component	Model Component	Knowledge Component
Welcome dialog	Adding and (re)moving tools for hospitals, roads and danger	Calculation of served population	Provide explanation of indicators
Load layers dialog	Ability to undo 1 step or undo all steps	Calculation of served area	Inform on relevant parts of performance indicator calculation model
Adjust map objects dialog	Ability to load layers	Calculation of distance to danger	Provide explanation on tools
Allow fast and on the fly adjustments	Keep GIS platform stable	Calculation of distance to water	Provide guidance through steps of SDSS
Ability to switch between different service levels for population served		Calculation of betweenness	Demo which shows how the SDSS works
Ability to set max. hospital reach and capacity		Calculation of degree	
Ability to save and load setup scenario, compare scenarios		Calculation of a suitability map based on indicator weights	
Show performance of hospitals in an overview		Make population and roads covered by danger inactive/inaccessible	
Reflect upon whether capacity is reached			
Set suitability weights dialog			
Provide suitable map symbology			

Table 3.3: Expected stakeholder requirements.

Based on these requirements, an example of typical use of the SDSS will be described next. Once the SDSS is opened, a welcome dialog appears, and then a dialog appears which allows for the loading of the data which will be used for the different performance indicators considered for the most optimal placing of the field hospitals.

Once this data is loaded in the SDSS, the different layers for which actions are possible and their actions are displayed in the interface. For example selecting an action and tapping or dragging leading to among others the following actions:

- If the toggle road action is selected and a road segment is tapped, the possibility is given to toggle the road segment (in)active.
- If the move danger action is selected, its borders can be dragged to reduce or extend the reach of this area.
- If the add hospital action is selected, the map can be pressed to add a field hospital.

After any of these actions, a table can be updated which shows the values for all the performance indicators for each hospital. The ability to do these fast on the fly adjustments of map objects allow a group of stakeholders/decision-makers to experiment with potential solutions and provide them with an effective way of negotiating their stakes into the model outcomes. Also, it provides decision-makers with the ability to use their hands-on experience and update the map in front of them by the developments in the conflict area. This way, the strength of the computer to solve structured problems based on hard data and decision-makers' experience and knowledge can effectively be combined. An example would be the possibility to quickly toggle a road segment as inactive to render it out of the analysis, when intelligence has just been obtained that the road has become inaccessible. Also,

should the decision-maker recognize an area that is likely to become dangerous in the near future, and then a relatively simple add danger action could mark the area as unsuitable for field hospitals to be placed in.

The performance of certain single field hospital locations and the combined performance of all field hospital locations in a certain setup are then visualized by certain the performance indicators, which contribute to the comparison of different setup scenarios. There is always the ability of adding, deleting or dragging field hospitals to other locations. Also after the calculation of performance indicators. Extensive help is provided in this process through a tutorial and by information on how to use different parts of the SDSS.

3.5 SDSS-Architecture

The SDSS in this research is basically developed in one GIS platform (QGIS) for which different third-party software is already supported. GRASS GIS algorithms for example are already integrated into the QGIS platform and the platform already uses PyQt5 for its interface. Looking at the different ways of coupling the SDSS in section 2.1.6, this SDSS will be tightly coupled as all of the functionality will be accessible from the interface of the GIS platform.

The architecture of the SDSS in this research is inspired by the one used in the research of Coutinho-Rodrigues, Simão, & Antunes (2011). Their SDSS is structured to serve functionality similar to that of the SDSS of this research. It for example also builds on existing GIS functionality, uses a customized user-friendly interface, uses an MCDM and includes interactive functionality to experiment with the performance criteria, in order to assist the decision-maker as good as possible. A schematic of the SDSS is provided below in figure 3.3.

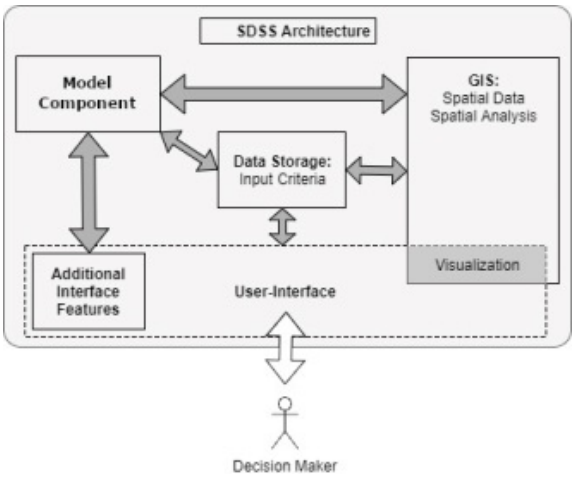


Figure 3.3: SDSS architecture.

3.6 Soft- and hardware

The SDSS will be tightly coupled with the QGIS platform. Interfaces for the SDSS are designed using Qt Designer, which is software for the design of interfaces. The interfaces and are coupled with the QGIS API through Python 3 scripts. GRASS GIS and native QGIS processing algorithms are used in the calculation of performance indicators.

Software	Purpose
QGIS 3.6.3	GIS platform to build SDSS into
Qt Designer	Software to design interfaces
Python 3.7.3	Used to link interfaces with GIS platform and processing algorithms
PyQT 5.13	Used to access and control interfaces
GRASS GIS 7.6.1	Used for various processing algorithms

Table 3.4: software used in this research.

Most decision (touch) tables operate on the visual input of another device. This allows for easy touch implementation in the SDSS: A device with the required software and the SDSS on it can be connected to a touch display which next can be used interactively to test the SDSS. The QGIS platform also supports touch input. Chapter 5 is dedicated to the testing of the SDSS. For the analysis of the testing, screen capture software and an external camera was used to capture the actions and reactions of the test participants.

3.7 Dialog Component

The dialog component of the SDSS includes interface functionality and their layouts. They are carefully designed to be as easily used as possible to stimulate the employment of the SDSS. In section 2.1.4, it is concluded that the accessibility, flexibility, interactivity, ergonomic layout and processing-driven functionality of an interface are essential. Interface accessibility in this SDSS will be pursued by providing all of the functionality in an easily accessible interface. The possibility of changing the input on the fly and undo incorrect actions make the system flexible. The interactivity of the SDSS is also ensured by efficient information flows between user and SDSS. At last the interface is processing-driven as it keeps the user up to date on what is going on and what has to happen next. The testing of this SDSS will reveal whether the interface indeed is easily used.

3.8 Knowledge Component

Although section 2.1.5 concluded that a knowledge component is optional, it is decided to integrate it into the SDSS because it can greatly increase the user-friendliness of the SDSS. We identified domain knowledge, model knowledge, utility program knowledge, metadata, and process knowledge. The included domain knowledge is not very comprehensive as not a lot is documented about real field hospital location decision-making, but could be implemented in future iterations if more experience of stakeholders is gathered. Model knowledge will be provided in the form of explanations of what the performance indicators mean. Utility program knowledge is provided in a visual explanation of how the different actions for map objects and the different visibility toggles work. Metadata is not required as it provides unnecessary distracting information for users. If they really want to, they can view metadata through the GIS platform itself. The process knowledge is provided directly in the interfaces of the SDSS to guide the user through the process of using the SDSS in their field hospital location decision-making process.

3.9 SDSS Testing

In order to complete the third sub-objective of this research on testing of the SDSS, an appropriate testing method has to be used. Since this is the first time that an SDSS is proposed to assist in the planning of field hospital locations, this is a bit of explorative research. It therefore is appropriate to also use a more qualitative testing methodology rather than quantitative. In this stage of SDSS development, qualitative research will surface most of the problems that come with an SDSS prototype and no quantitative testing is needed yet.

The ‘think aloud’ method is used in the testing of this research. The method involves the analysis of recorded voice and actions (both on screen and with their bodies) of test participants while participants are asked to voice their thoughts during decision-making (Van Someren, Barnard & Sandberg, 1994). The method is being used in a variety of fields, including the testing of user interfaces in software. The method is very natural to execute for test participants as most people already speak their thoughts when they are discussion decisions. The recordings during tests allow to analyse the decision-making as accurately as possible. The specific testing with the ‘think-aloud’ method is further discussed in chapter 5.

4 DESIGN AND IMPLEMENTATION OF THE SDSS

In this chapter the design and implementation of the SDSS are discussed. First, an initial design, which resulted from the expected stakeholder requirements of section 3.4, is discussed. Then the first implementation up to the pre-test is presented, which had resulted from self-testing. Next, a second design is presented, which is an implementation that resulted from the updated requirements after the pre-test. Last, a final implementation is presented which resulted from the feedback and updated requirements after test 1. Feedback from test 2 and other requirements that have not yet been implemented in the final design will be reflected upon in recommendations. The tests that have been executed with these implementations, are fully considered in chapter 5.

4.1 Initial SDSS Design

The initial SDSS design contained 6 screens which interacted with a map. These 6 screens guide and support the decision-makers through the process of choosing the most optimal locations for field hospitals. When you open the SDSS, a welcome screen in shown which provides a short description about what the SDSS can be used for and which steps can be taken to get to most optimal locations. Proceed can be pressed to go to the next step and cancel to exit.

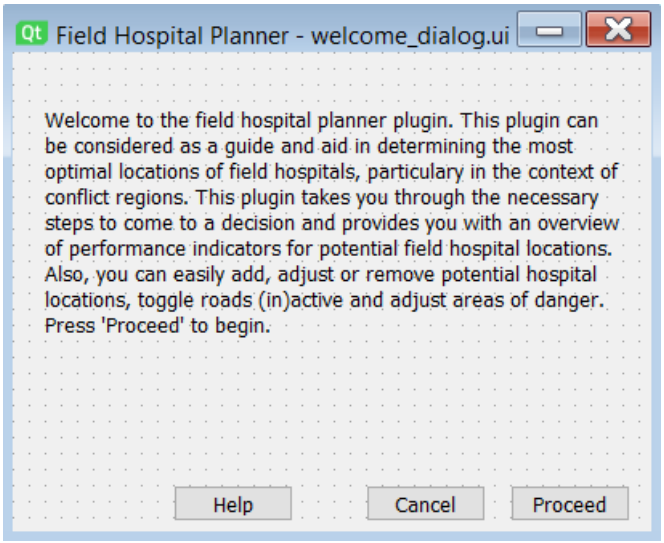


Figure 4.1: Field Hospital Planner welcome dialog design.

In the next screen, paths can be selected to load the correct input layers. Only population and roads are essential for the SDSS to function. Also the help button can be pressed from this screen onwards to get additional info about a screen.

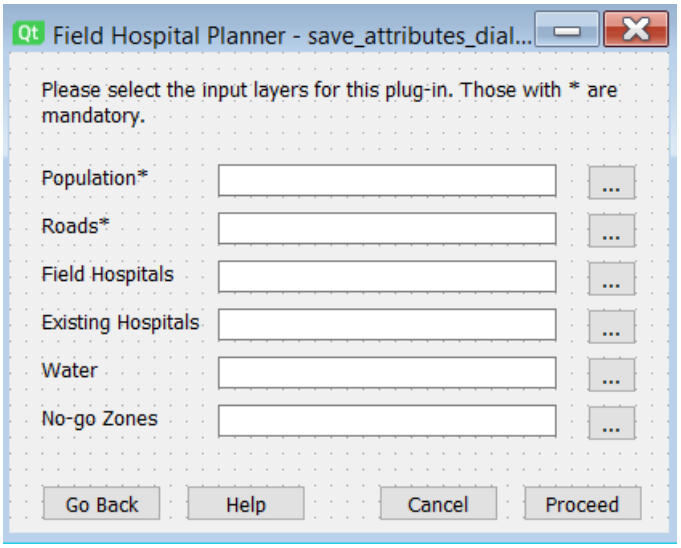


Figure 4.2: Field Hospital Planner load layers dialog design.

The 3rd screen contains several input settings which characterize the to be planned field hospitals: number of hospitals, hospital reach and capacity. Capacity might be divided and set for different levels of medical service. Pressing the 'proceed' button will load all the layers and settings onto the map and the progress bar indicates when it is done loading.

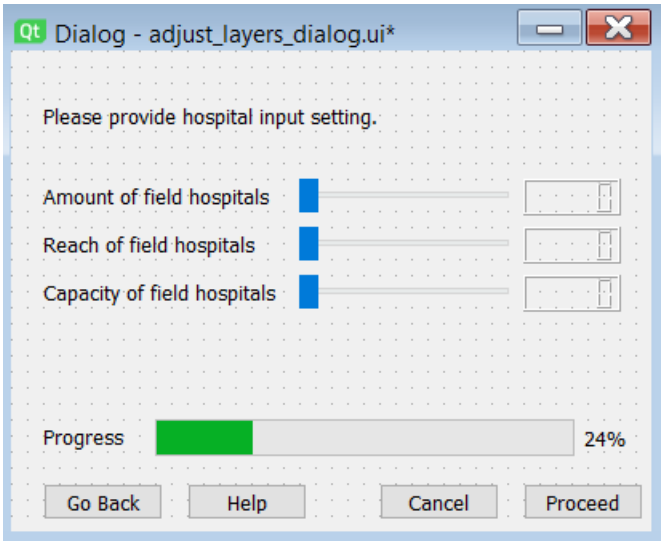


Figure 4.3: Field Hospital Planner hospital settings dialog design.

A 4th screen will show a table with all of the hospitals and performance indicators. It will appear as soon as the input layers are loaded, hospital settings have been set and the indicators have been calculated. If there are many hospitals, the table can be scrolled through and an average row will always stay visible at the bottom row. A certain hospital distribution's table can be saved. This screen will stay visible throughout the remaining screens.

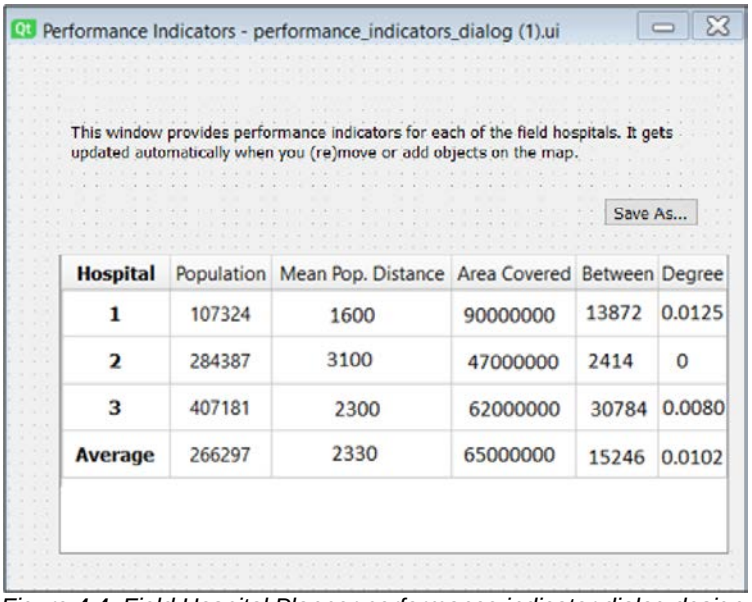


Figure 4.4: Field Hospital Planner performance indicator dialog design.

Screen 5 will provide advice on completion. This screen can be skipped if no advice is required. It requires weights to be set to each of the performance indicators. These weights are then used to calculate a total score suitability raster map. Locations of water or danger are excluded as possible hospital locations on this map. The total suitability by the different weights will be shown on the map, but suitability for each of the individual performance indicators can be viewed as well. This map provides quick insight to which locations are most suitable for field hospitals. A progress bar indicates when the suitability map is finished.

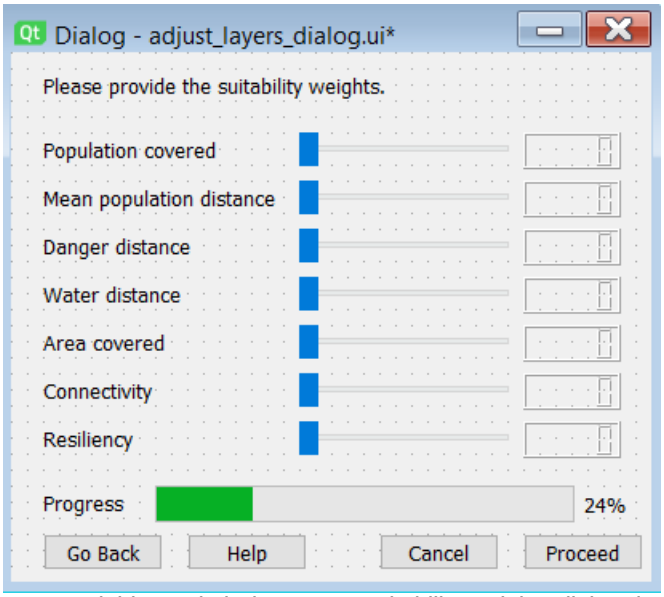


Figure 4.5: Field Hospital Planner set suitability weights dialog design.

Screen 6 is the final screen. This screen allows the decision-makers to easily and in a user-friendly manner adjust relevant objects on the map. If the move hospital radio button is pressed, it for example provides the action to move hospitals on the map by dragging them to another location. The performance indicators for this new location will refresh almost directly to inform the decision-makers about this location. Adjustments can be undone by 1 action or by all actions since adjusting began. Different compositions of field hospitals can also be saved and loaded in order to compare the performance of different scenarios.

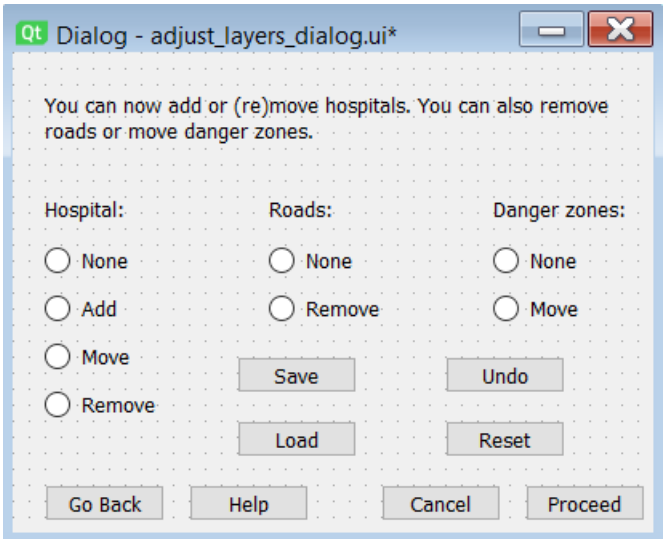


Figure 4.6: Field Hospital Planner adjust objects dialog design.

4.2 First SDSS Implementation (pre-test)

For the first development run it was decided to mainly create proofs of concepts. Three main concepts had to be explored: tool development, calculation of performance indicators and interface design and control (which includes the presentation/visualization of the indicators). If these three concepts were successfully implemented, then an expansion of the software into a coherent functioning SDSS was possible. This implementation has resulted from self-testing and has been tested at the pre-test to see whether the three concepts were sufficiently functional.

4.2.1 Tool Development

The tool development covered the possibility to easily select an action for a certain map object (hospital, road or danger) and then perform this action by pressing and possibly dragging these map objects on the touchscreen. For some actions, existing tools in the QGIS platform were used and for others new tools were developed. The toggling of roads (figure 4.7 a) for example is a newly developed tool. When the user presses a road, the identified road map object has its activity attribute switched between yes/no and it turns the road red if it is toggled inactive. Although adding and removal tools already exist in the platform, custom tools were also developed for these actions because they take unnecessary many steps by the user for the application of this SDSS. For moving map objects, existing functionality has been incorporated because it works as desired (figure 4.7 b & c). The script for the developed tools can be found in appendix D.

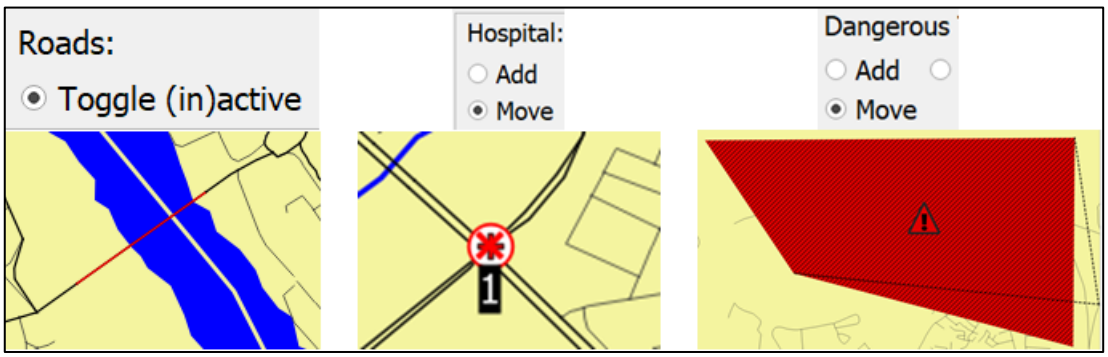


Figure 4.7 left to right: (a) toggle roads, (b) move hospital, (c) move danger tools.

4.2.2 Performance Indicator Calculation

The definitions of the performance indicators have been provided in section 3.2. The calculation of these performance indicators involves linking various native QGIS and GRASS GIS algorithms. Whereas distance to water and danger require just one algorithm, calculation of hospital service area and the population in this area can take up to five. The calculation of the degree and betweenness indicators takes three to four algorithms. Figure 4.8 below shows the model for performance indicator

calculation. It starts with two to six input layers on which various algorithms are executed to get the values for two to six performance indicators for each hospital.

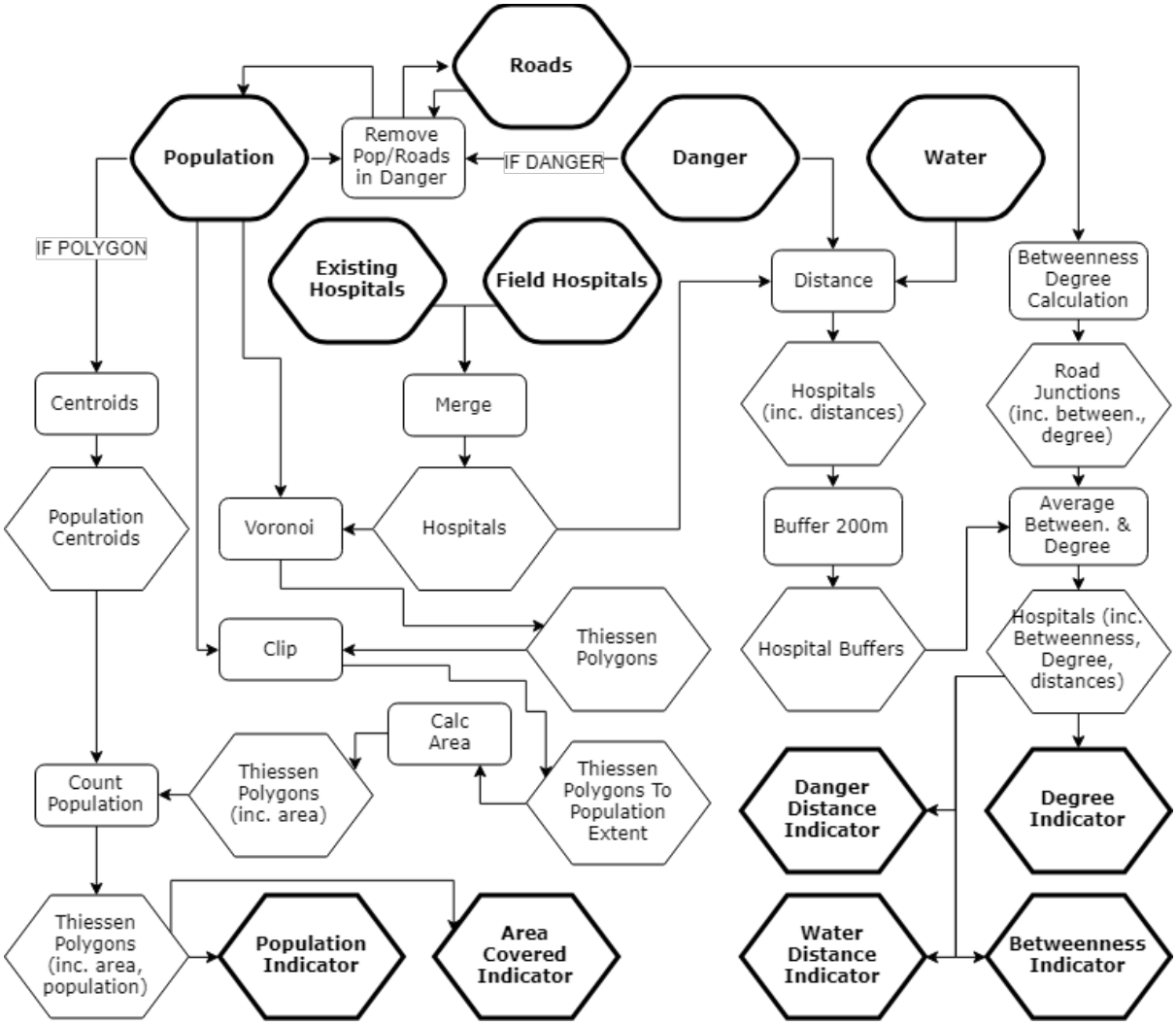


Figure 4.8: Performance indicator calculation model. 2-6 Input layers and 4-6 output indicators.

4.2.3 Interface Design and Control

Another main concept which required exploration was the design of the SDSS interface and the capability of controlling it. Dialogs have been designed in Qt Designer software. Objects in the interface dialogs have been connected with QGIS functionality through Python code. In figure 4.9 below, the four dialogs are shown which resulted from the first exploration of interface development. They are linked by their ‘proceed’ and ‘go back’ buttons and dialog (d) would show up once data layers have been loaded. The ‘…’ buttons in dialog (b) are linked to file selection screens and the radio buttons in dialog (c) to a couple of algorithms which together manage the tools section.

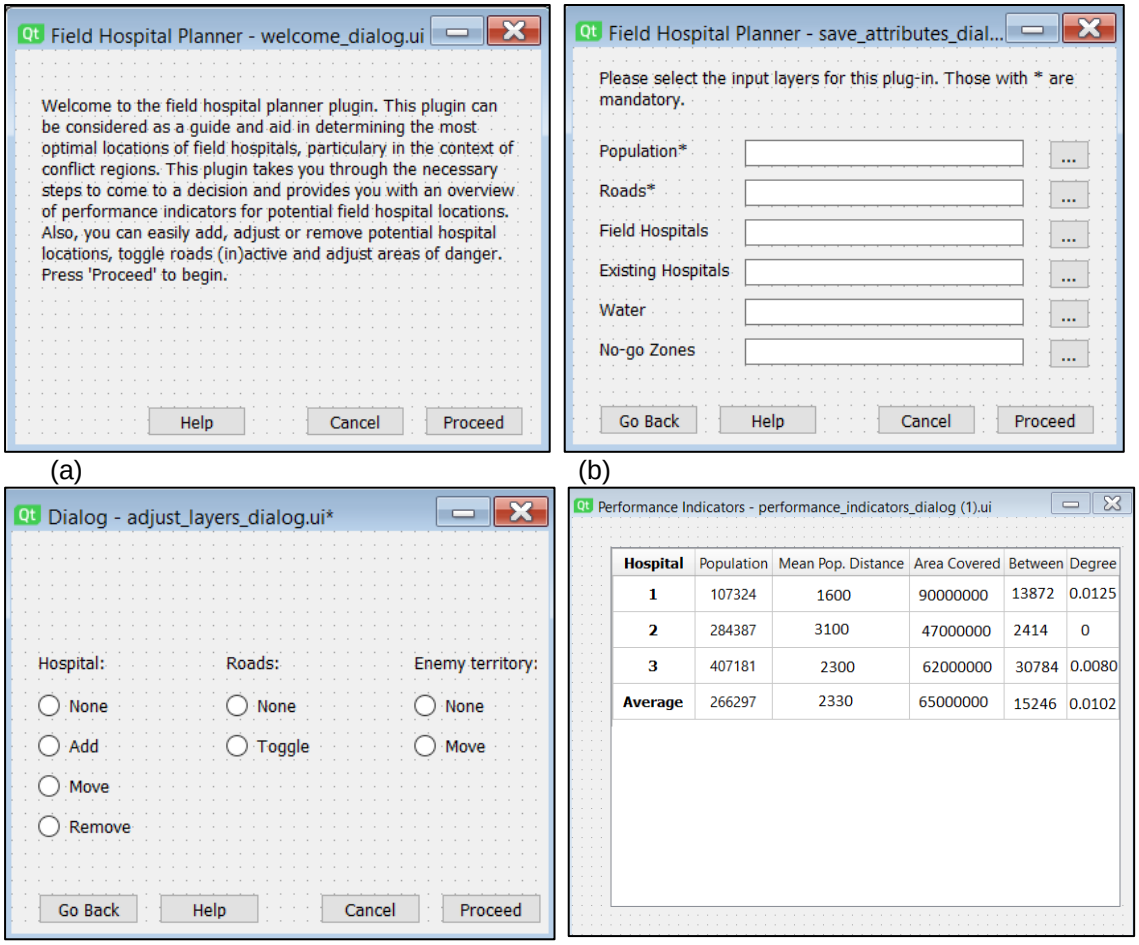


Figure 4.9 Interface dialogs: (a) welcome dialog, (b) load layers dialog, (c) adjust layers dialog and (d) performance indicators dialog.

Performance indicator dialog (d) uses a pre-existing Qt Designer table widget. If any adjustments have been made to the loaded map layers and the table is updated, then the performance indicator calculation model from figure 4.8 will be run again to show new values in the table. The bottom row will always display an average of the above rows and the number of columns can vary between five and seven dependent on whether danger and water data have been provided. In this implementation, indicator values were presented in a hard to interpret manner. Area covered for example shows square meters and between and degree have not been normalized.

4.2.4 Pre-test

As mentioned in section 4.2, the pre-test was conducted mainly to see if the main concepts of the SDSS were viable. If these three concepts were successfully implemented, then an expansion of the software into a coherent functioning SDSS was possible. These concepts were identified from the stakeholder requirements in section 3.4. They are the required tools, the calculation of performance indicators and successful interface design and control. The tested implementation had resulted from self-testing. This first SDSS implementation was showcased to several academics, of which some have interacted with and advised on the Mosul case study (figure 4.10 below). Shaheen Abdulkareem from the Mosul area and someone with SDSS testing experience were also among the academics.

Figure 4.10: SDSS pre-test. Showcasing the three main concepts.

The pre-test showcased the basic functionality of the SDSS. In this version it almost did not have any guidance yet. Looking at the first main concept; the required tools, the pre-test proved that they were functioning. The functionality was there, but switching between tools was still very buggy and could cause crashes. Also after updating the performance table the tools would cause crashes. So the ability to switch between different actions and buttons should be handled better in the next implementation. Then the second concept, the calculation of performance indicators, was also functional. It did take a couple of minutes however and the GIS platform would not be responsive during this calculation, causing the impression that the software had crashed. The communication of the resulting performance indicator values was also vague and unclear, especially because of unrelatable numbers and a lack of hospital identification. Then the third main concept, the design and control of the interface, was showcased in some bare functionality. The performance table was working (until it stopped due to a bug) and different toggles and buttons were successfully linked to desired events. Control of the interface was no issue, but a lot more issues needed to be fixed and functionality implemented. Plans for future implementations of the SDSS were also shared with the invited academics and feedback was given on the showcase in return. The full resulting feedback can be found ordered by test criteria (the test criteria is further discussed in chapter 5) in table 4.1 below.

Intuitiveness	**Performance**	**Reasonability**	**Usability**	**Stability**	**Completeness**	**Guidance**	**Success**
Replace 'none' with save/undo buttons	Indicators take long to load		Streets or districts should be referenced	Editing actions not working after each other	Make SDSS dockable to sides of GIS platform	Guidance largely missing	Yes, but a lot of improvements needed.
Make performance indicators interpretable				Non-responding when computing	Show hospital IDs	* means something is necessary	
Visualize results as well				Enable edits after table update	Visualize hospitals by serving population size	Explain indicators	
					Provide an overall score by weights		

Table 4.1: Pre-test feedback, divided by test criteria from the test design.

4.3 Second SDSS Implementation (first test)

In the second implementation, the three main concepts from the first design have really been integrated into a cohesive SDSS. Most feedback from the pre-test has been implemented in this version. Table 4.2 below shows all the implementations between the pre-test and test 1. They have been categorized in the same way as the stakeholder requirements in section 3.4.

Database (GIS) Component	Model Component	Dialog Component	Knowledge Component
Toggle population density	Add average distance to field hospital indicator	Replace none buttons by a save/undo button	Provide explanation of indicators
Deactivate or remove most of the default toolbars, panels and menus of QGIS	Bug fix: Exclude roads and population from algorithms if covered by danger	Make indicator values better interpretable (normalize road indicators)	Inform on relevant parts of performance indicator calculation model
Add toggle hospitals (in)active tool		Make plugin dockable	Provide explanation on tools
Provide ID to new hospitals		Make pinch zoom function less sensitive	Provide guidance through steps of SDSS
Add an add danger tool		Visualize hospital IDs	
		Bug fix: Switching between tools/actions not fully working	
		Bug fix: Edit/update table not working after each other	

Table 4.2: SDSS implementations between pre-test and test 1.

A first and important update in this implementation is that the Field Hospital Planner SDSS is now docked either on the left or right side of the window. Also most menus, toolbars and panels have been removed from the QGIS interface (figure 4.11). The load layers dialog stayed mostly the same (figure 4.12). It now has switchable extra guidance text and a warning is given if no (correct) population or roads data is selected.

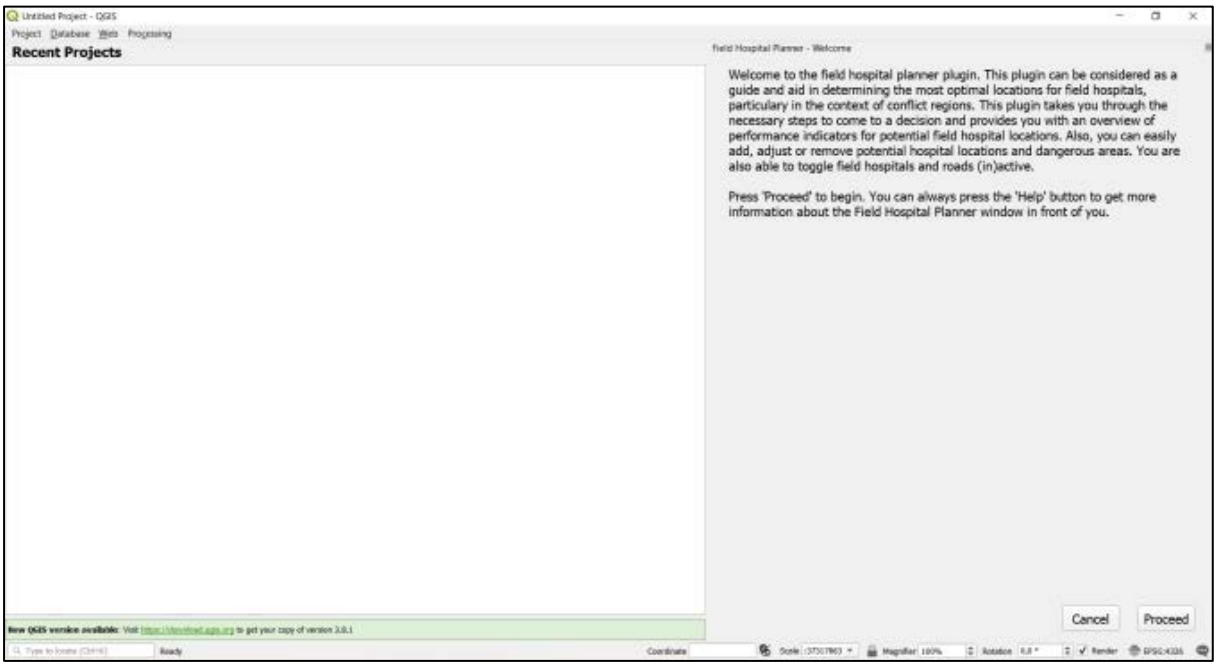


Figure 4.11: Second implementation welcome dialog.

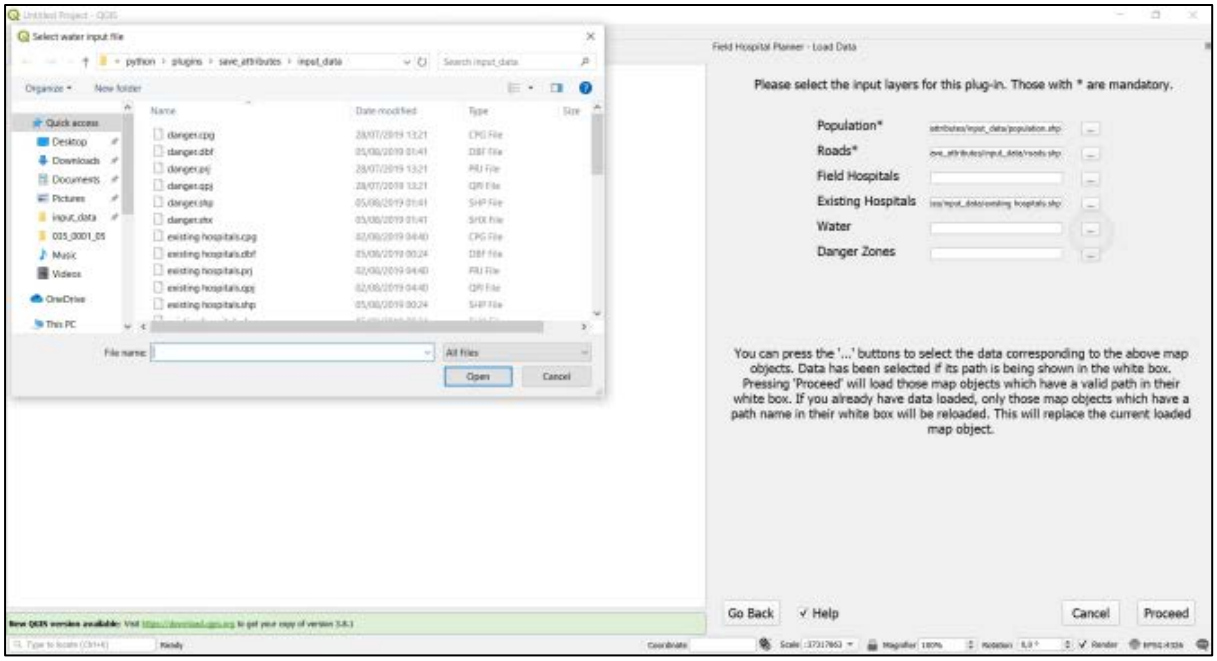


Figure 4.12: Second implementation load layers dialog.

In the second implementation, the adjust layers and performance indicator dialogs have been merged to create more order with less screens (figure 4.13 below). Guidance has been added to the dialog (figure 4.13 below), which informs on all the actions and buttons. It is now possible to toggle the activity of hospital and to add danger areas. Also a population density visualization can be toggled, which are the green blocks in the map area. The none buttons in the previous implementation have been replaced by a single save/undo button, which has to be pressed after each adjustment to either save these adjustments or undo them. Only after clicking this ‘save/undo’ button you are enabled to use another action or to update the table. Also a ‘whole view’ button has been added which zooms the map back to an overview. The table does not auto-update after adjustments to the objects on the map, but the ‘update table’ button has to be pressed to prevent unnecessary updating. Performance indicators now have more readable and meaningful presentations (including units and some normalized values) and a mean served population distance to hospital indicator has been added. Last, a help button toggles a dialog in which the meaning of performance indicators is further explained and a legend is provided (figure 4.14).

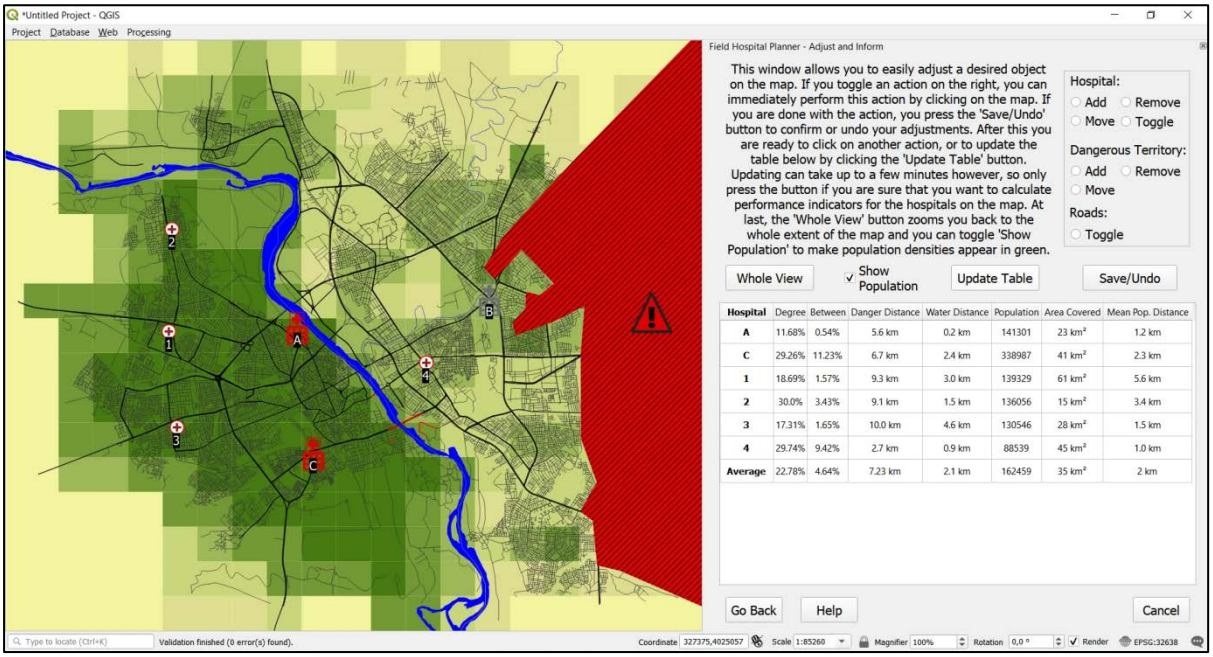


Figure 4.13: Second implementation adjust and inform dialog.

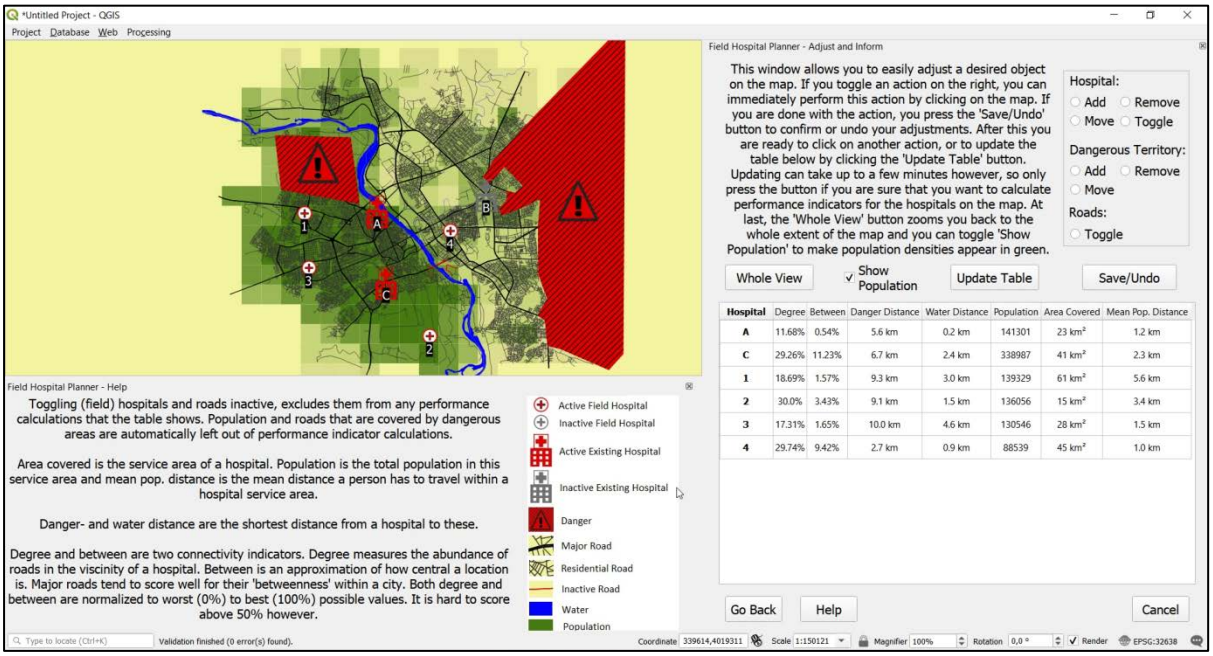


Figure 4.14: Help dialog (bottom left) with legend.

4.4 Third SDSS Implementation (second test)

In the third implementation any remaining stability problems of the SDSS have been fixed. It also has feedback from test 1 implemented. Feedback that has not been implemented at this point, can be found in appendix E. Table 4.3 below shows all the implementations between the test 1 and test 2. They have been categorized in the same way as the stakeholder requirements in section 3.4.

Database (GIS) Component	Model Component	Dialog Component	Knowledge Component
Toggle hospital service areas	Upon update, only fully rerun degree and between. algorithm if roads or danger are adjusted	Warnings added for: reloading layers, trying to update while adjusting, trying to (re)move existing hospitals	Small update for changed functions
	Bug fix: crash if no junctions within 200m of hospital	Provide togglable legend with including population values	
		Change indicator order	
		Add scalebar	
		Bug fix: hospitals sometimes mix up in table	

Table 4.3: SDSS implementations between test 1 and test 2.

Figure 4.15 below shows the third and final implementation of the Field Hospital Planner SDSS. Less has changed between the first and second implementations, but some valuable adjustments were made nonetheless. A floating legend dialog which includes population intensity values, is now togglable and can be placed wherever the users desire. Also a self-adjusting scalebar has been added to the bottom-right of the map area. There now is a ‘show hospital service area’ button as well. Besides these mainly visual adjustments, a performance change has been made as well. Previously, the update table would execute the whole model for performance indicator calculation. Because of this, updating the table would take up to several minutes. Probably at least 95% of this waiting time is caused by calculations made on the road network. Therefore, road network calculations are only repeated whenever roads or danger have been adjusted. This reduces the update time to a maximum of ten seconds if only the hospitals have been adjusted.

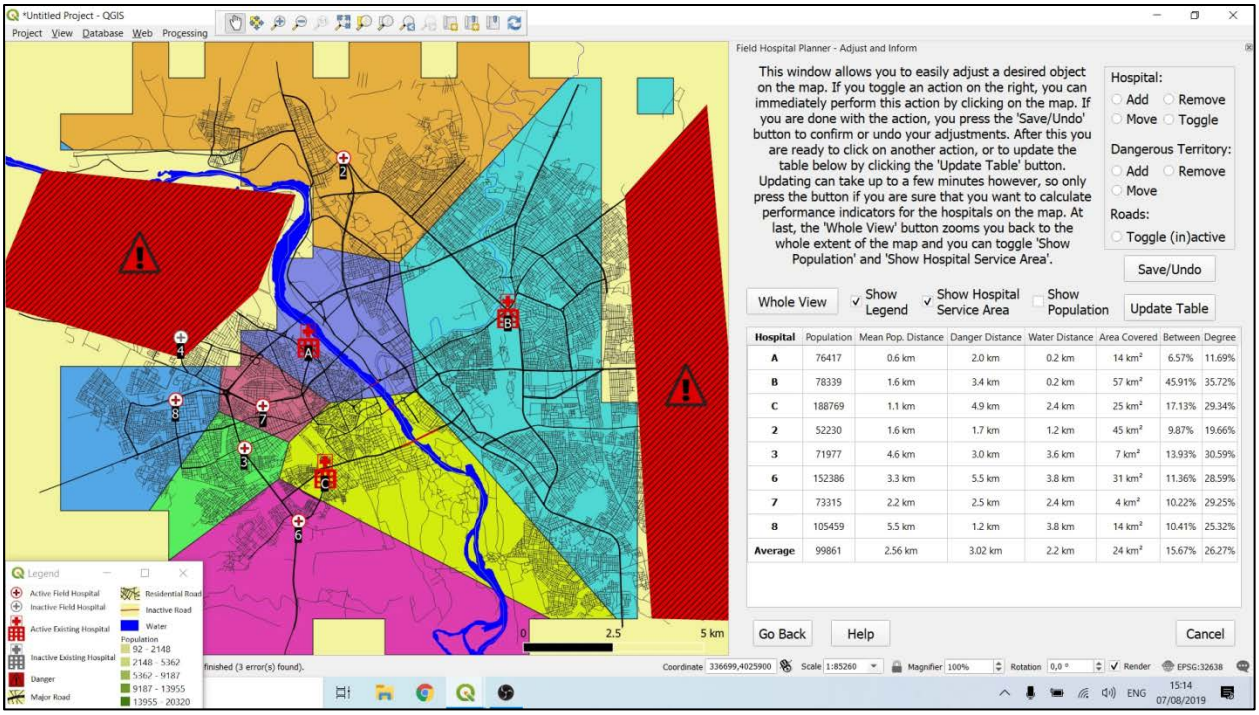


Figure 4.15: Third implementation adjust and inform dialog.

This third implementation clearly contains some elements of the initial design of section 4.1. On the other hand, screens have been merged, options added and left out, functionality added and the final implementation turned out more complex than the initial design. What is missing from the initial design are some hospital characteristics settings and the ability to produce a suitability map.

5 TESTING OF THE SDSS

This chapter first provides information on the nature of the SDSS user tests, as well as the specific settings and set-ups that these tests were conducted in. Also the test- roles, script and selection of participants are extensively discussed. Next, the proceedings of the different tests and their outcomes are presented.

5.1 Test Design

The tests are designed after certain goals and according to criteria. In short, the goal of the SDSS is to be able to easily adjust hospitals, roads and danger on the fly, quickly inform on the resulting performance change which among other relevant indicators includes good connectivity and resilience measures. Seven testing criteria have been identified to cover all aspects of the SDSS. These are intuitiveness, performance (speed), reasonability of outcomes, usability of existing functionality, stability (crashes, bugs), completeness of functionality and guidance. Last, it is important to mimic a real field hospital location group decision process as much as possible, assuring scientific validity.

5.1.1 Test Set-up

Because the SDSS has to be used by a group who exhange information live and all have to physically access the SDSS in their group decision-making, a decision room is required. The decision room requires a big decision (touch) table for all the decision-makers to stand around. Because the 'think aloud' method is used, the decision table and the decision-makers have to be filmed and their voices recorded. The display of the decision table is filmed as well, as this enables to later watch back both the actions taken on the screen as well as the gestures made and the talking done during these actions. Some notes were taken during the tests, but it would be impossible to write down as much information as recording captures. More information means that the tests can be better analyzed. After each of the tests, the earlier mentioned testing criteria are assessed in a group feedback moment and participants are allowed to provide additional information on their cognitive processes during the test. While it is natural for most people, especially in group decision-making, test participants are reminded to please speak their thoughts when performing actions. Also, a general idea of the testing criteria is shared with the participants before the test and that they are encouraged to share their thoughts on these criteria during the test. Last, they are reminded that the experiment is not about rating their actions and solutions, but about what helped and did not help in coming to a solution.

Throughout the development of the SDSS, it of course has been self tested a lot. As explained in section 4.2, a pre-test has also been performed in which three main concepts were tested for functionality: tool development, calculation of performance indicators and interface design and control (which includes the presentation/visualization of the indicators). After this pre-test, the software has been expanded to be a coherent functioning SDSS which could be tested properly like a real world decision-making process. Two tests were then performed with three participants each. The participants got either the role of WHO, allied military forces or GIS specialist assigned. Their profiles are discussed in the next section of this chapter. I moderated the tests by providing instructions for the tests and reading the test script to the participants, which is discussed in 5.1.3. I only intervened if a problem would arise which prevented the participants from progressing the test in order to not unnecessarily influence the tests.

5.1.2 Stakeholder Profiles and Instructions

Because no real field hospital location decision-makers have participated in the testing of the SDSS, it was of extra importance that clear and extensive stakeholder profiles were provided. The test instructions and role-specific profiles can be found in appendix A. The instructions start with a rough planning, to inform the participants what they are up to. It starts with a short introduction of the SDSS and some context of the Mosul case study. Then the participants have time to study their stakeholder profiles and they are allowed to ask questions about these if things are unclear. If they are ready the test begins and after the test there is a group feedback moment and the possibility for a small discussion.

The context about the Mosul case study introduces the positions of the different stakeholders and that they have conflicting interests. It also introduces the task of planning field hospitals with the SDSS as

a supporting tool. The different performance indicators and tools are explained as well. Then the stakeholder profiles provide a backstory for each stakeholder, ending with a small summary of their goals. This summary makes it easier for a stakeholder to lookup what goals have to be kept in mind during the planning (see table 5.1 below).

WHO	Allied armed forces	GIS specialist
Primary goal: Utilize full (field) hospital capacity.	Primary goal: No (field) hospitals are allowed within 1 km of danger.	Primary task: You lead the decision-making process, are a sort of moderator and you operate the touch screen if the others struggle to understand the software.
Secondary goal: Minimize pop. travel distance.	Secondary goal: max. 2 hospitals within 3-1 km distance of danger.	Primary goal: Ensure that hospitals are well connected (near main roads, reflected by the 'between' indicator). Above 15% is very good, under 5% is bad.
Tertiary goal: Minimize water distance.	Task: Toggle existing hospitals inactive if they are within 1 km of danger.	Secondary goal: Ensure that hospitals have a resilient location (many escape routes, reflected by 'degree'). Above 30% is very good, under 15% is bad.
Task: Ensure that existing hospitals are not added, moved or deleted. But they can be toggled (in)active.	Task: Add danger areas if needed.	Secondary task: You could possibly help recognize areas that will easily be 'cut off' from the street network if danger were to expand

Table 5.1: Stakeholder goals and tasks summaries.

Sometimes the goals or tasks are ranked, as utilizing the full hospital capacity for example is much more important to the WHO than being close to water. The goals have been experimented with in self-testing to see what were suitable and realistic values, slightly conflicting interests and relevant tasks. With these stakeholder profiles an attempt is made to make the participants act like their roles as much as possible. After the specific stakeholder profiles, the instructions visually explained some buttons and their workings. The instructions end with some important notes like practical tips for the use of the SDSS, the touch table and a reminder on only asking for my help if it is really needed to proceed with the test.

5.1.3 Test Script

The script for testing has been carefully designed. It can be found in appendix B. The goal of the tests was to stress the SDSS as much as possible: test all the different functionality, make sure different combinations and orders of functionality are used to see whether the SDSS remains stable. The test script has been designed with three things in mind. First, to make sure that all necessary aspects are covered in the tests, the script has been designed after seven testing criteria. These are intuitiveness, performance (speed), reasonability of outcomes, usability of existing functionality, stability (crashes, bugs), completeness of functionality and guidance. Second, special attention has been given to stressing the main desired characteristics of the SDSS: the ability to adjust map objects on the fly, quickly getting informed on the performance change and reasonable connectivity and resilience measures. Third, it is tried to mimic a real field hospital location decision process as much as possible. To make the test script reflect a real decision process, conflicting scenarios were created which most likely forced the stakeholders in conflicts of interest. At the same time, it was made sure that the scenarios did allow for solutions in which all stakeholders were somewhat satisfied. The challenges of the script also slowly build up, to allow the stakeholders to get used to the software and their roles.

The script starts with instructions to load relevant data into the SDSS. Then the stakeholders are presented with an outdated situation, one of the three active existing hospitals is too close to a new danger zone and three bridges have been destroyed, they will have to toggle the hospital and bridges

inactive using the provided tools. But first, they have to update the performance indicators to see the performance of a single hospital which is right next to one of the bridges that will soon be toggled inactive. The stakeholders are quickly informed on the performance of the field hospital and are told to toggle the bridges and make the existing hospital inactive and then update the indicators, paying special attention to the connectivity and resiliency change when updating the indicators. They see that the connectivity score drops heavily and that the resiliency score drops somewhat. By showing the stakeholders this, they have now become acquainted with how these accessibility values work. The 'between' indicator for connectivity took such a big hit because the bridge provided many routes to pass the field hospital. Its 'degree' indicator dropped slightly, because the amount of escape routes only decreased by one (figure 5.1 and 5.2 below). Also, the performance table does not show existing hospital B anymore because it has been toggled inactive. Figure 5.2 shows that the hospital service areas have been updated accordingly to the change.

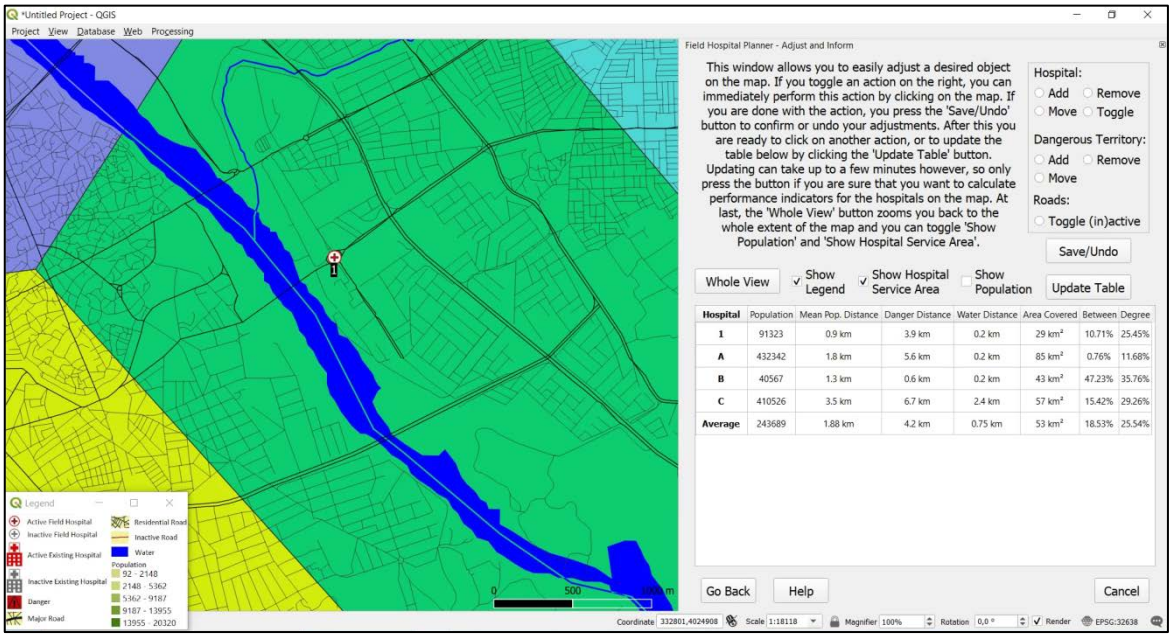


Figure 5.1: Performance indicators are communicated on the right of this screen for a situation in which all roads are active.

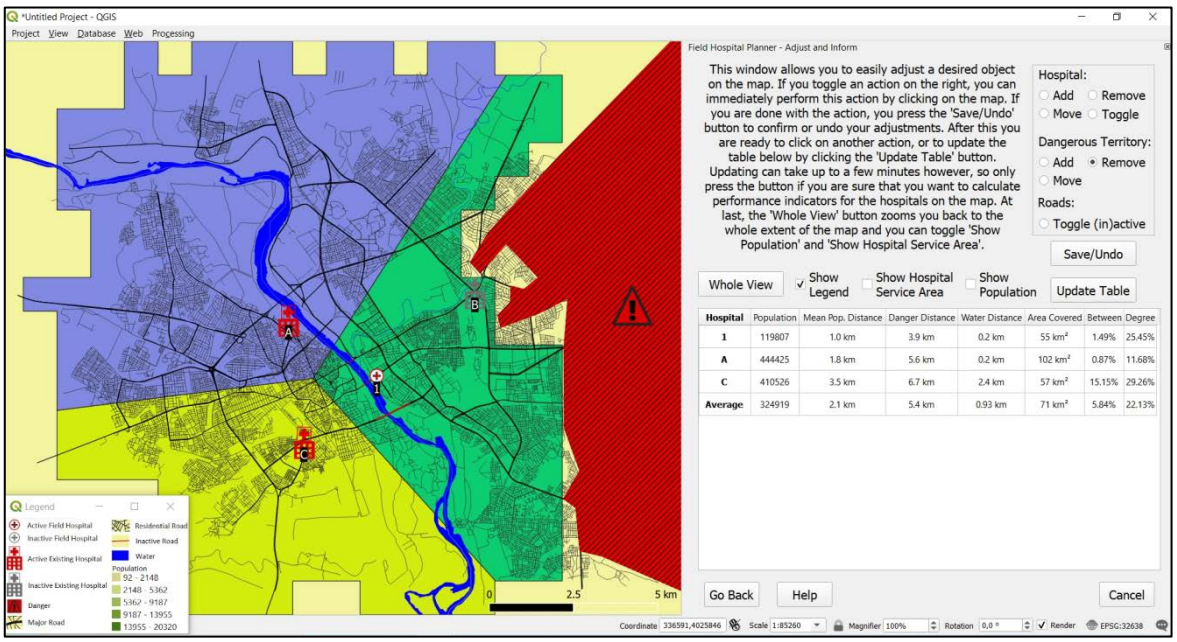


Figure 5.2: Performance indicators are communicated on the right of this screen for a situation in which bridges (roads over water) next to field hospital 1 are toggled inactive.

The stakeholders are told that the field hospital has little use with such connectivity and are ordered to delete this field hospital. They get information on the capacity of existing- and field hospitals and the number of field hospitals at their disposal. They are reminded that they are able to toggle useful map layers. After this the stakeholders are instructed to select field hospital locations by consideration of their interests. They also have to take the effects of the two active existing hospitals in mind. Figure 5.3 below shows a possible solution.

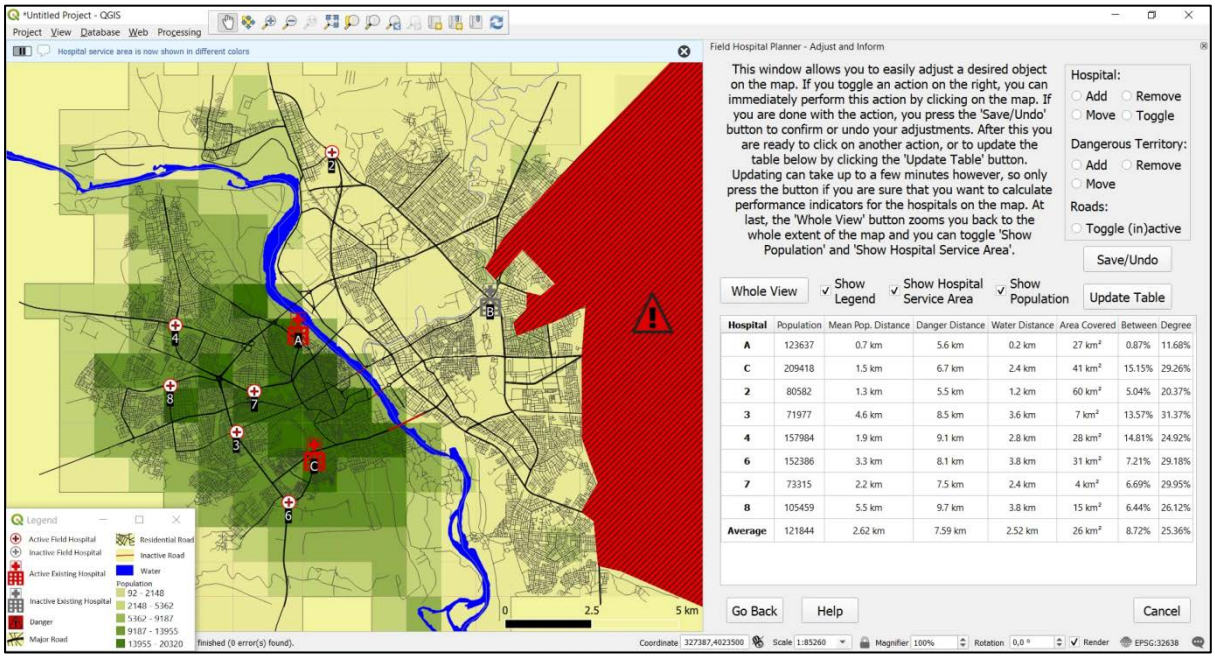


Figure 5.3: Possible solution for day 1 of the test script. Hospital A does not meet its capacity of 150.000 people.

The stakeholders are allowed to experiment for a while with different compositions of field hospitals and if they think that the locations suffice or if it takes too long, the stakeholders are told that we proceed to the next part of the simulation. The next part aims at more interaction with danger polygons and an increase in conflicts of interest. It is seven days later, and their field hospitals performed well. The danger in the east retreats 1km, and the stakeholders have to update this on the map. They are reminded that if hospital B is further than 1km away from danger, that it can be toggled active again. Then the stakeholders are presented with a new danger close to the densely populated west side of the Tigris river. They have to add this danger and toggle any hospitals inside it inactive. The danger deliberately spans an area in which probably at least one field hospital was located. The stakeholders are also told to toggle some nearby roads inactive and update the performance table in order to reflect the new situation. Their composition does not make sense anymore and they are instructed to move their maximum of five remaining field hospitals to suitable locations. All stakeholders are under more pressure now. The danger is closer to the population, which forces the military to yield their goals a bit. More active existing hospitals require careful planning of the WHO to make sure that most of the available service capacity is used. In order for the WHO and the military to somewhat achieve their goals, the GIS specialist has to abandon the most connected and resilient locations and opt for acceptable locations instead. Figure 5.4 below shows a possible solution after day 7. Only hospitals A and B do not meet their capacity of 150.000. But in no other configuration would B get more population, as the existing hospitals cannot be moved. All field hospitals meet their capacity of 50.000 people.

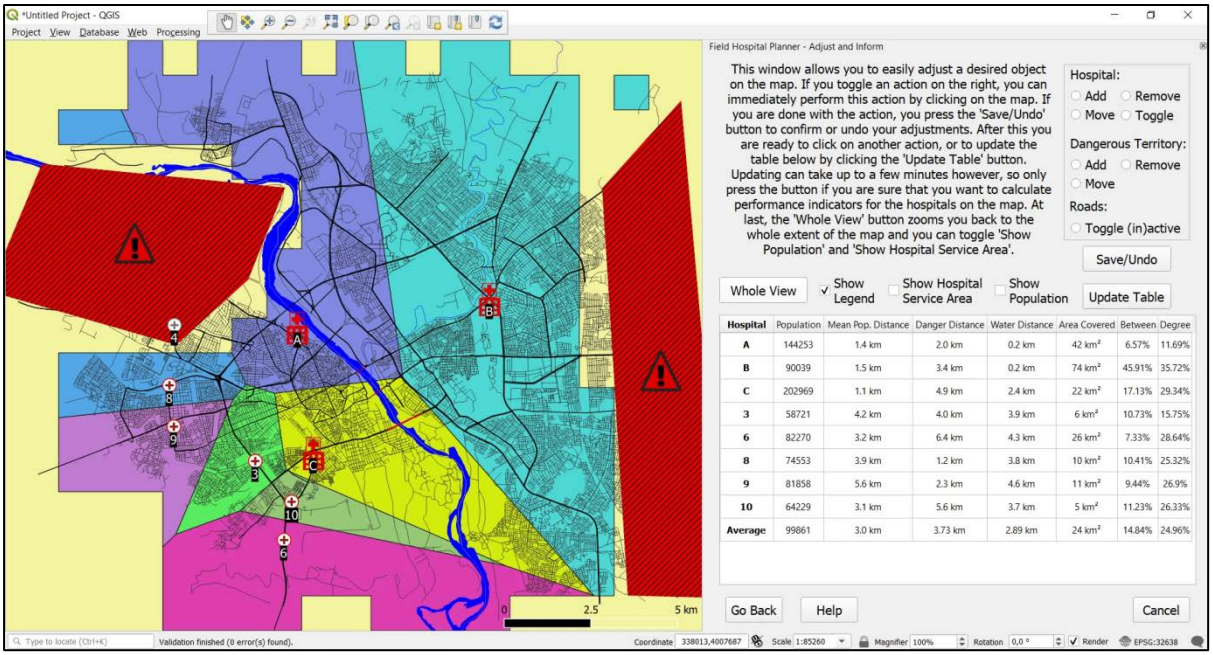


Figure 5.4: Possible solution for day 7 of the test script. Hospital A and B do not meet their capacity of 150.000 people.

5.1.4 Selection of Test Persons

As it was not possible to test the SDSS on real-life users, it was attempted to at least find the most suitable people. Finding the right participants for the role of the GIS specialist was least challenging, but finding participants with a similar background to that of the WHO and the armed forces proved to be too difficult. Instead, it was opted to find participants with diverse backgrounds for these roles and provide the participants with extensive profiles for their roles. The GIS specialists always had at least some reasonable experience with GIS platforms. The WHO and armed forces during test 1 did have some GIS experience, but during test 2, they did not. The selected participants were a diverse group of people with different genders, ages, education levels and fields and occupancies. Participant 4 also has an attention deficit disorder, which especially displays in having trouble in obtaining overviews. The diversity in the characteristics of these groups and the attention deficit disorder increase the chance of different views and experiences in the testing of the SDSS and may increase the chance of obtaining useful feedback. After all, the characteristics of the GIS specialists, WHO employees and armed forces could vary a lot in a real life situation too.

Participant	Test	Gender	Age	Education	GIS experience	Daily occupation
1	1	Male	66	BSc	Experienced	Cartographer
2	1	Female	-	MSc. Geo-Information Management and Appliances	Old experience.	Lecturer geosciences, human geography and planning, social urban transitions
3	1	Female	53	PhD eco-hydrology	17 years GIS for RS	Educational material developer and software tester
4	2	Female	25	BSc.	Reasonably experienced with ArcGIS	Soil geography and earth surface dynamics student
5	2	Male	23	Applied psychology	None	Applied psychologist
6	2	Male	22	Pre-university	None	Waiter

Table 5.2: key characteristics of test participants.

5.2 Testing Outcomes

Next, the course of the different tests and their outcomes will be discussed. The main takeaways are discussed in full and the whole range of feedback is presented in tables that are ordered by the earlier

identified testing criteria. These are intuitiveness, performance (speed), reasonability of outcomes, usability of existing functionality, stability (crashes, bugs), completeness of functionality, guidance and whether the participants overall succeeded to complete the tasks at hand.

5.2.1 Test 1

Section 4.3 shows that the next implementation of the SDSS featured a lot more functionality and was much more of a complete coherent SDSS. This next implementation was tested in test 1. From table 5.2, participant 1 played the role of GIS specialist, participant 2 the role of the WHO and participant 3 the allied armed forces.

The participants did not have many questions about the test instructions. These instructions were provided only on the screen however, which proved inconvenient to consult the instructions again during the test. Therefore in test 2 instructions were provided on paper. The test participants needed a little practice to get used to the tools in the SDSS, but quickly managed to perform the desired adjustments on the map. A general issue with the touchscreen appeared to be that it was difficult to press an object not directly in front of you due to your angle at the screen and the thick glass. Due to the thick glass, you actually press too close to yourself and not on the meant destination. Vertically this could be fixed by turning the table a bit towards the participants and horizontally by only pressing the part of the screen where you are standing, or first walking towards that part of the table. While there was the possibility to toggle a population density visualization on the map, it was a bit of guessing for the participants to determine which hospital served what people. The SDSS did allow the participants to experiment with served populations by moving hospitals around, but the effects of these adjustments were only visible after an update of the performance table. This update took 2-3 minutes, which disabled quick insight in the effects of different hospital distributions. A need for quicker update calculations and a visualization of hospital service areas was identified. The provided population intensity visualization also lacked a corresponding reference to population numbers, which should be added to a legend. Also when the participants had to determine distances, there was no tool or reference to assist them in this. A simple scale bar or a tool for measuring distances through roads should solve this. All of the tools needed to execute day 1 of the test script were functioning correctly and the participants ended this part of the script with the setup in figure 5.5 below.

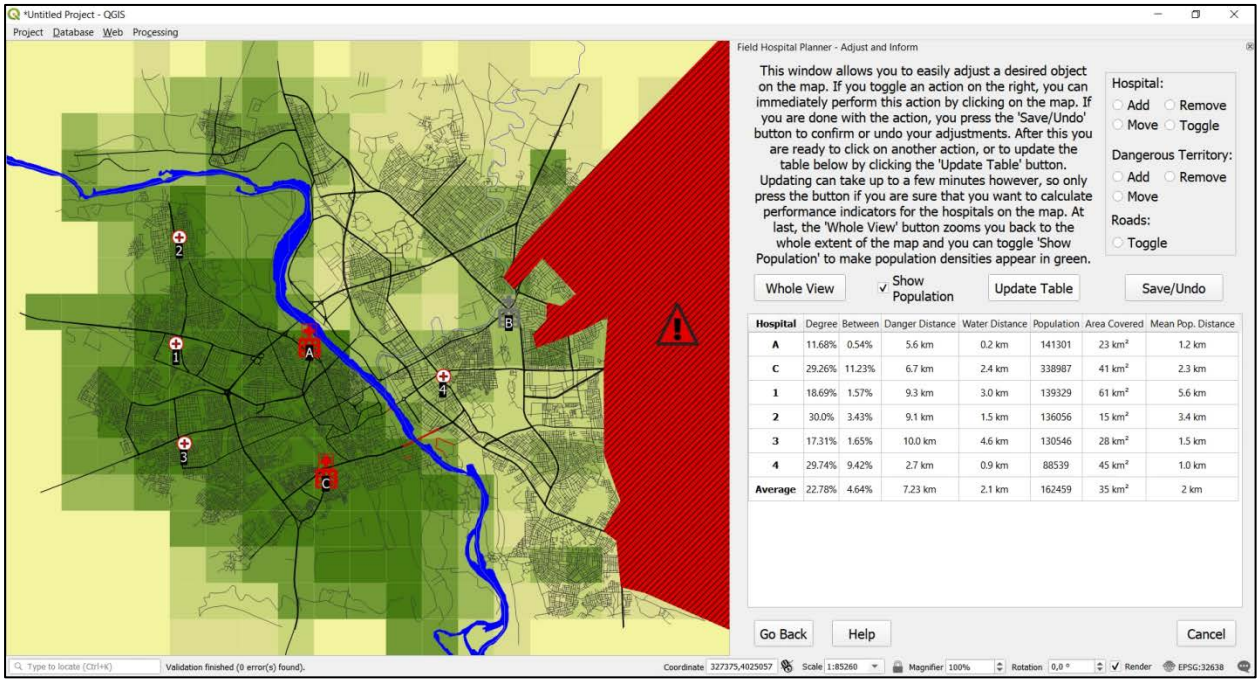


Figure 5.5: SDSS test 1 field hospital setup.

For the next simulated day of decision-making on field hospital locations, the participants had to add a danger zone. This proved cumbersome as one would expect to be able to draw areas on the map, but instead corners of an area would appear wherever the map was pressed. Although this was shared in the instructions, it was not clear enough and a dragging method would be more natural. I had to intervene to make clear what the actual method for adding an area was in order for the test to

continue. At the very end in this second part of test 1, a field hospital was placed in an area with very few roads. This produced an error in the SDSS upon updating the performance table and at the time it was unclear why. We decided to leave it at that and finish with a group feedback moment. Later it was found out that the SDSS gave an error because of how connectivity and resilience indicators are calculated. For hospitals these indicator values are the average of these values for junctions in a 200 meter radius around every hospital. But no junctions were within 200 meters of one of the field hospital, providing an unanticipated absence of values.

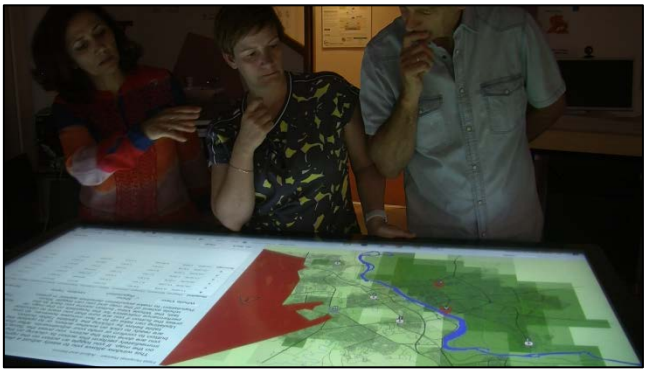


Figure 5.6: SDSS test 1. Allied armed forces, WHO and the GIS specialist evaluating changes in hospital locations.

To conclude, we held a small feedback session in which the participants shared their thoughts on the previous mentioned test criteria. We concluded that some small bug fixes were needed to be able to perform the task at hand in a user friendly manner. The full results of the test are ordered by the earlier mentioned test criteria in table 5.3 below. Looking at the objective of this research, the result of this test was that the SDSS did allow for easy adjustments of the data, but that getting information on the changes took too long. The computed connectivity and resilience values did however appear realistic, so the waiting time for these values was not for nothing. Ideally you could toggle which indicators had to be updated and shown, possibly reducing the time needed to update them.

Intuitiveness	Performance	Reasonability	Usability	Stability	Completeness	Guidance	Success
The intuitiveness is good. It takes short getting used to, but then its fine.	Indicator update is too slow.	Feels reasonable, if the input data is indeed realistic.	Essentials are there.	Difficulty with adding danger zones.	Add floating legend	Add readability points in big numbers	No, but almost
Provide population numbers in legend	Add multithreading to prevent non-response when processing		Auto-update would be nice if it was computed near-immediate.	SDSS got error on last table update due to unexpected input.	Add ruler/scalebar or tool to measure distance through roads		
Change order of indicators	Make road / accessibility indicators optional to decrease waiting time		Move and add functions could be combined.	Hospitals can mix in the performance table	Add road accessibility visualization toggle		
			Toggle which indicators have to be showed	Add danger a bit buggy			

Table 5.3: Test 1 feedback, divided by test criteria from the test design.

5.2.2 Test 2

The third implementation of the SDSS was tested in test 2. Less has changed than between the pre-test and test 1, as is described in detail in section 4.4. From table 5.2, participant 4 played the role of GIS specialist, participant 5 the role of the WHO and participant 6 the allied armed forces. This time, the participants were given instructions on paper, providing them with an easy accessible goals summary.

Again it took a little practice for the participants to get used to the SDSS, but actions took no more than two attempts before they were clear. These participants especially had trouble with not touching the screen when they meant not to. Upon pressing the map, the map would center on the pressed location, causing the view to shift upon accidental touch. The whole view button proved very useful to get back to the overview of the map. The participants found the buildup in script difficulty really useful. Immediately after the bridge hospital example, it was clear to them what the connectivity and resilience indicators meant. The provided guidance was experienced as a bit unstructured and was preferably all in one callable place or in a tool with which elements could be pressed and more info about that element would then be provided. Ideally, you could call a full visual tutorial for the SDSS. During this test, updating only took 2-3 minutes when changes were made to the roads or danger(s), in all other cases 10 seconds at maximum. It was clear that getting information on the adjustments was much easier like this. The participants could easily perform the first day of the script except for one moment where pressing the map for a longer time at one spot caused the map to not be movable by hand anymore. I had to intervene and manually select this map move tool again for the test to continue. The added hospital service areas, floatable legend and scale bar really helped the participants in their decision-making.

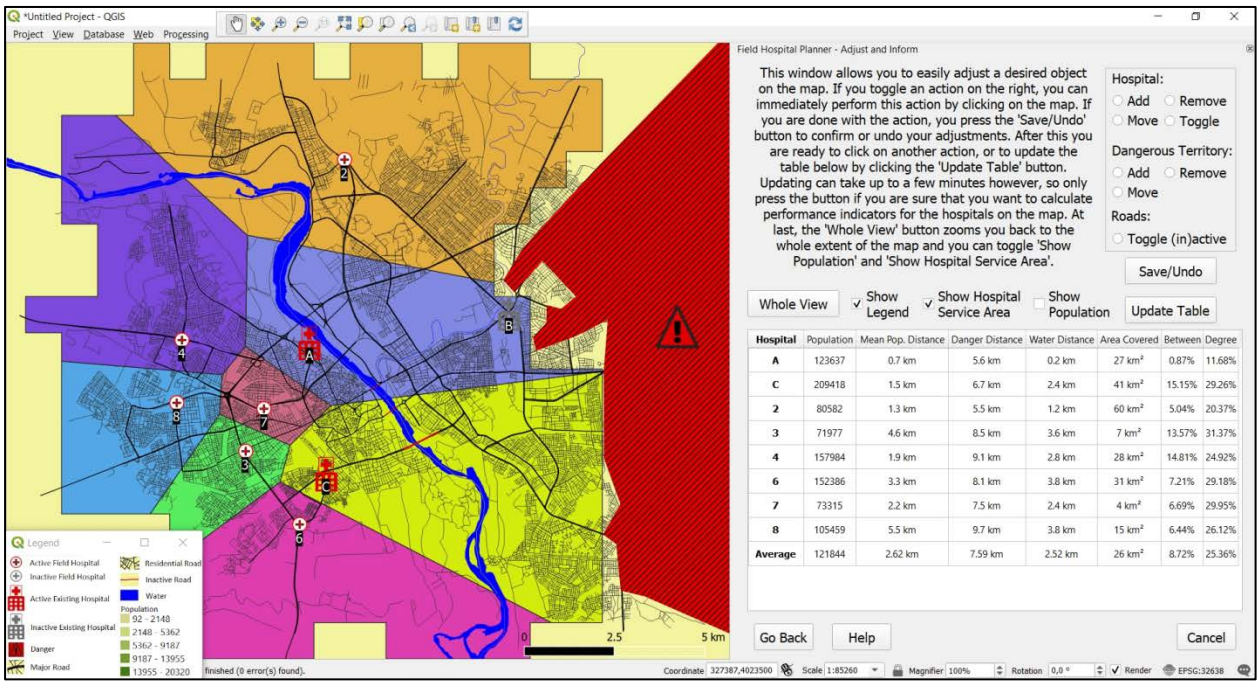


Figure 5.7: SDSS test 2 day 1 result.

For the second part of test 2 again a danger area had to be added. This time, it was no problem to create, but again it was said that drawing a polygon would be more intuitive. Moving a danger area proved more troublesome as every corner of a polygon has to be moved individually. It was suggested to create the possibility of moving a polygon as a whole, making it a lot easier to make a danger ‘retreat’. It was opted to just remove the existing danger and create a new danger as this was way faster than the current moving action. Further the test could be performed without too much problems and in the end very reasonable hospital setups were made, not much more performance seemed possible within the provided context.

Figure 5.8: SDSS test 2. Allied armed forces, GIS specialist and the WHO evaluating changes in hospital locations.

The overall conclusion in the feedback session of test 2 was that the SDSS was very suitable for the tasks at hand and that it was easy to gain quick insight into different hospital setups. Also, this time, updating usually took a lot less than during test 1 due to making road calculations only necessary after road- or danger changes. It was perceived as doable to have to wait on the roads update the few times it had to be done. In reality, such a change is probably performed only once per session, unless the road and danger situation happens to change during the decision-making process. Given the option of faster connectivity and resilience calculations for less accurate values, the participants all said that they would rather have as accurate as possible values. For hospital service area calculation, which currently uses a very quick but inaccurate method, it was suggested to offer at least two calculation methods. This way, the users can decide themselves whether they want a certain update of indicators to be fast, or accurate. The explained calculation methods for hospital service area in section 2.2.3 already discussed models that ranged from fast to accurate, so this could be implemented rather easily. To better communicate connected and resilient areas, a visualization of these values could be toggled for the roads on the map. Other suggestions were zoom in- and out buttons and an undo only one step button. The full results can be found ordered by the earlier mentioned test criteria in table 5.4 below.

Intuitiveness	Performance	Reasonability	Usability	Stability	Completeness	Guidance	Success
Intuitiveness is good. Takes short getting used to, but then fine.	Map refresh at every move unnecessary	Results and indicators seem really reasonable	Pop up save/undo dialog at switch tool attempt	Zoom by pinching not working after a long press	Functionality very complete for task at hand.	Could be even more clear if it was more structured	Yes, SDSS very well enables the tasks at hand
Zooming while pinching would make more sense instead of after taking the fingers off the screen	Update roads takes a bit long, but you that probably only once at the beginning of a session.	Served population is quite simplified, but an indication nonetheless.	Whole view button very useful	Average distance to hospital in served area does not make sense in one area	Zoom in/out buttons.	Make guidance fully optional, so you get new map space once you do not need guidance anymore	
Map centers to point on the map that is pressed. Should be disabled because people will accidently touch the map.	Different options for hospital service area calculation. From fast to accurate.	Connectivity and resilience really make sense	Toggles are very useful, they help you maintain overview	Toggles do toggle, but do not register whether the toggle object is active or not	Instead of action toggling, a menu with possible actions when pressing the map	Create a help tool which gives information about the pressed interface element	
Danger move not intuitive, need to be able to move the polygon as a whole.			Apply connectivity and resilience to roads instead of junctions	Warning for invalid danger geometry not working.	Undo 1 step button	Stress that SDSS is just a support, not necessarily reality.	
Would be nice to have a toggle for whether screen is touchable					Ability to toggle auto table update	Visual demo / tutorial	
Add cancel to save/undo button					Toggle connectivity and resilience map		

Table 5.4: Test 2 feedback, divided by test criteria from the test design.

5.3 Discussion

The different tests nicely reflect how the SDSS has been developed by taking feedback into account. Where the implementation at the pre-test was just a proof of concept, the one at test 2 seems a fully functional and coherent SDSS for the planning of field hospital locations in conflict areas. Here the main takeaways from the tests and the possibilities for improvement are discussed by the different testing criteria. This is done by comparing test 1 and 2 for each of the criteria. An overview of all unimplemented feedback can be found in appendix E.

The intuitiveness of the SDSS is quite good. The participants had little trouble with finding functionality and usually the required actions were clear after a first attempt. There are some things which could add to the intuitiveness even more though. First of all, a more 'modern' touch control for the SDSS. There are conventional expectations of what will happen if you apply certain touches on a

touchscreen. You would expect to be able to drag objects, zoom immediately before releasing a pinch and minimize dialogs by sliding them into the side of a screen for example. The touch handling of the QGIS platform is a bit out-dated to this respect because it does not react this way to many of such touches. You first pinch and release and then you can see how far you have actually zoomed in or out, or you press a hospital and then press another location to place it there instead of dragging the hospital there. Such behaviour could better be 'updated' to contemporary conventions to make the SDSS more intuitive. With a big decision table an option to disable touch input would help as well as people will lean on the table or point to objects and unintentionally move a map.

The performance and the reasonability of the information displayed in the SDSS have quite a big interplay. The art lies in finding a nice balance between the two. More accurate calculation methods often also take more time to process and vice-versa. During test 1, the updating of performance indicators clearly took a little bit too long. In test 2, this was resolved by only performing the heavy calculations when they were necessary: when roads or danger have been altered. If participants had to wait less than 10 seconds on the update, then they were not too bothered. Different adjustments could further increase performance and reasonability of results. Currently, the SDSS will be unresponsive when the performance indicators are being updated and calculated. By writing the indicator update in multi-threaded code, other actions could still be performed during the updating. This allows for updates that take a bit longer and therefore are more accurate. For example, the raster- or vector-based service area allocation methods could be used to determine the dedication of population to different hospitals. These methods are expected to result in more reasonable hospital service areas. Another possibility in this regard is to allow for the selection of different calculation methods, ranging from fast to accurate. This would allow the users to for example only use the accurate method at the end of their planning process to check if the chosen hospital distribution is indeed the most optimal one. A final suggestion to speed up hospital service area calculation with more sophisticated methods is to opt only to use major roads. Also, this reduces the difficulty and time consumption to create correct road networks in a conflict area.

The provided functionality of the SDSS was deemed essential and usable. However, if the test participants could change some functionality, then they would merge add and move actions. Also, it would be fine to add more toggles. A toggle could be added to select an 'auto-update' function, or the ability to toggle more visualizations of indicators. Especially an on map visualization of connectivity and resilience would be very useful. Functionality that could be added is the ability to set weights to performance indicators and then calculate a total performance score which is expressed in a suitability map which visualizes the total suitability of all possible locations on the map. Some basic SDSS functions like more zoom and undo buttons would also be convenient. Then a save, load and report on scenario function would make the SDSS complete.

Regarding the stability of the SDSS, a couple of bugs remain in the current implementation. But no errors or bugs were found in the last test which would make it impossible to proceed the planning process. The main bug is the earlier discussed unresponsiveness of the SDSS during computations and multi-threading could serve as a solution to this inconvenience.

The guidance in the SDSS was perceived as sufficient. Metadata information was not missed, and domain knowledge was provided through the stakeholder's profiles of the test participants. The model knowledge (performance indicators), utility program knowledge (tools) and process knowledge were provided at the right places. A very good remark was that the user should be able to hide all of this guidance to make more room for the actual map. Most information already had the ability to be toggled, but utility program knowledge and some process knowledge are not yet. Solutions that would be even better are a visual tutorial of the SDSS and a 'help tool' with which you can click elements of the SDSS to gain more information about them.

Overall the test participants deem SDSS in its current state already very suitable for the given tasks and the performance and reasonability of the performance indicators are identified as the most important aspects to an even better SDSS for planning field hospital locations in a conflict area.

6 CONCLUSION AND RECOMMENDATION

6.1 Conclusion

The first sub-objective of this research, ***'To identify suitable performance indicators and SDSS components to include in a SDSS for the planning of field hospitals locations in a conflict situation'***, has been achieved by identifying suitable performance indicators and SDSS components that had to be included in an SDSS for the planning of field hospitals locations in a conflict situation. Knowledge has been gathered of algorithms that exist for performance indicator calculation in the planning of field hospitals. Most important, the Voronoi diagram, betweenness centrality and degree centrality have been identified as algorithms suitable for the context of a field hospital planner SDSS. Different measurements for the connectivity and resiliency of a location especially have been compared to see which were most fit to be implemented in performance indicators. The components of SDSSs have been reviewed to create a careful design for the SDSS and a method has been explored for testing the SDSS. Other performance indicators for the context of field hospital location selection have also been identified and defined. With the knowledge of the review on SDSS components in mind, SDSS functionality requirements have been identified.

The second sub-objective of this research, ***'To develop a flexible SDSS which allows for on the fly adjustments and support for finding connected and resilient field hospital locations'***, has been achieved by developing a flexible SDSS which allows for on the fly adjustments and support for finding connected and resilient field hospital locations. In this SDSS, the greater insight into the successful planning of field hospitals in a conflict situation from sub-objective 1 has been implemented in the SDSS. It has been made interactive in order for field hospital location decision-makers to implement their knowledge and hands-on experience with real life conflict developments and similar experiences into the SDSS and make easy, on the fly adjustments to the input data of the SDSS. The SDSS effectively communicates performance indicators to decision-makers through visualizations in the SDSS interface. The SDSS easily lets decision-makers adjust and reflect on performance indicators, which greatly supports them in their decisions. Performance indicators other than location connectivity and resiliency have been visualized as well, showing a total of seven indicators per hospital in a clear performance table. It is also made sure that decision-makers are guided through the process of planning field hospital locations by the interface of the SDSS. Add, (re)move and toggle activity actions have been made available for relevant map objects in the SDSS to provide flexibility and interactivity. Decision-makers using the SDSS are guided through their decision-making process by a welcome, load layers and adjust map and show performance indicators screen.

The third sub-objective of this research, '***To test whether different SDSS implementations are a suitable support for the decision-making process of planning field hospital locations'***, has been achieved by testing whether different SDSS implementation iterations were suitable supports for the decision-making process of planning field hospital locations. The tests revealed the extent to which the SDSS is suitable for the given tasks and feedback has been gathered. After each of the tests new implementations have been made in order for various iterations to be tested. Different tests were needed and a setup has been designed for these tests. Also test participants had to be selected and a test script has been made which stresses all relevant elements of the SDSS. Use was made of the 'think aloud' method during these tests. The tests simulated real life field hospital location decision-making as much as possible. Special attention was given to the connectivity and resiliency of locations and the ease of interactivity with the SDSS. Also, testing criteria were identified which had to be stressed during the tests. The main conclusions on these testing criteria are:

- The intuitiveness of the SDSS is quite good, but a more 'modern' and intuitive would improve this even further.

- The performance and the reasonability have quite a big interplay and are currently acceptable for the SDSS. Multi-threading would improve the performance and allow for more sophisticated methods to be used. The raster- or vector-based service area allocation methods could be used to determine the dedication of population to different hospital in a more reasonable manner. Providing the selection of different calculation methods, ranging from fast to accurate and only using major roads in calculations could also improve

performance.

- Regarding the provided functionality of the SDSS, this was deemed essential and usable. Participants would merge add and move actions, add more toggles, add an 'auto-update' function and add the ability to toggle more visualizations of indicators (especially connectivity and resiliency).

- Functionality that could be added is the ability to set weights to performance indicators and then calculate a total performance score which is expressed in a suitability map which visualizes the total suitability of all possible locations on the map.

- Regarding the stability of the SDSS, a couple of bugs remain in the current implementation. But no errors or bugs were found in the last test which would make it impossible to proceed the planning process.

- The guidance in the SDSS was perceived as sufficient at provided at the right places in the interface. Guidance could be improved by a visual tutorial of the SDSS, a 'help tool' with which you can click elements of the SDSS to gain more information about them and the ability and the ability to toggle guidance.

- Overall the test participants deem SDSS in its current state already very suitable for the given tasks. The performance and reasonability of the performance indicators are identified as the most important aspects to an even better SDSS for planning field hospital locations in a conflict area.

The main objective of this research, ***'to design and develop a spatial decision support system for the selection of field hospital locations in conflict areas that is data extensive and is based on fast algorithms. The SDSS executes a set of models that will be selected by a literature review on different models for calculating performance indicators in the planning of field hospital locations. Performance indicators are generated which will visualize output scenario impacts. The developed performance indicators will be integrated in an user-friendly and interactive SDSS which allows for on the fly adjustments of map objects and quick decision-making. The aim is to identify locations with a high connectivity that are also resilient to conflict dynamics while reckoning with other relevant performance indicators.'***, has been achieved as the SDSS executes a set of models that were selected by a literature review on different models for calculating performance indicators in the planning of field hospital locations. The execution of these models provides performance indicators that are communicated to the user through a clear performance table. The SDSS overall is user-friendly and interactive and it allows for on the fly adjustments of map objects and quick decision-making. The aim to identify locations with a high connectivity that are also resilient to conflict dynamics seems to be reached as these performance indicators reflected realistic values to me and test participants in testing. They also reflected conflict dynamics in a realistic fashion: destroyed bridges for example would realistically change connectivity and resiliency of locations.

This research is a good start in the exploration of SDSS suitability for field hospital location decision-making in conflict areas. It addresses gaps in both the scientific areas of SDSSs and field hospital location decision criteria. This research suggests that an SDSS is very promising for this purpose and next, recommendations are given for future research in these scientific areas.

6.2 Limitations and Recommendations for Future Research

Besides possible implementation of unimplemented feedback which is discussed in section 5.3, various other recommendations could be made for future research and implementations of a field hospital planner SDSS. These recommendations expose the limitations of this research and indicate how these could be resolved.

The scientific justification for the creation of a field hospital planning SDSS could be strengthened by a research which involves actual field hospital planners in conflict areas. SDSS design decisions are rationally justified in this research with expected requirements for an SDSS that serves this purpose, but the consult of real decision-makers is the only way to obtaining real SDSS requirements. Tests

which involve real decision-makers for sure will manifest new requirements and missing functionality. This would also allow for the development of location criteria theories for field hospitals in conflict areas, which currently are largely absent due to their niche subject. To strengthen the external validity of this research even more, the SDSS should be employed in different real conflict scenarios. Also, only a limited number of tests have been executed with a limited number of participants which performed a limited number of tasks. This was sufficient for explorative research on this topic, but letting real life decision-makers use the SDSS more often, will address these limitations and expose those requirements which can only be discovered by empirical quantitative research. Although test participants for this research were selected by careful consideration, selection bias remains, because these were no real field hospital planners. Regarding the 'think aloud' methodology, it has only been adopted to a limited extent. No expected protocols have been designed and compared to the resulting protocols from the tests due to time constraint.

Limitations related to GIS platform (QGIS), in which the SDSS is implemented, are that it is time-consuming to develop all the necessary functionality and to prevent or change undesired behaviour. Especially the software's build-in response to touch input differs from contemporary expectations, as dragging map objects, for example, is by default not supported and objects are moved by pressing on them and then pressing on a new location.

Limitations related to the decision (touch) table, are possible limited availability of such a table in real life cases and impracticalities of using such a table. Users are inclined to lean on the table, causing undesired windows to pop up or close. Or something is pointed at and accidently clicked. Also, a combination of the glass thickness and the angle of the view on the table can cause unintended positions on the screen to be pressed. If it is the case that decision tables are only limited available in practice, then there is a need for a field hospital planning approach that does not require a decision table. One solution would be a setup in which an SDSS is synchronized among different computers. Laptops with touchscreens are commonly available and decision-makers could still negotiate the decision-making process in a single room. The big difference would be that instead of one large decision (touch) table, each decision-maker looks at an own laptop screen. If the decision-makers are indeed physically together in one room, then this would not require an internet connection either, something that could be lacking in a conflict area. Such an approach would of course have its own limitations, such as having less of an overview due to the smaller screen.

REFERENCES

Aarts, E. & Korst, J. (1989). Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing (Chichester, Sussex: John Wiley)

Abdullahi, S., Mahmud, A. R. bin, & Pradhan, B. (2014). Spatial modelling of site suitability assessment for hospitals using geographical information system-based multicriteria approach at Qazvin city, Iran. Geocarto International, 29(2), 164–184. https://doi.org/10.1080/10106049.2012.752531

Angus MacSwan. (2017). Iraqi bridge is sole link for Mosul residents rebuilding lives. Retrieved November 16, 2017, from https://www.reuters.com/article/us-mideast-crisis-iraq-bridge/iraqi-bridge-is-sole-link-for-mosul-residents-rebuilding-lives-idUSKBN1A706D

BBC. (2017). How the battle for Mosul unfolded. Retrieved November 16, 2017, from http://www.bbc.com/news/world-middle-east-37702442

Belkin, N. J. (2000). Helping people find what they don't know. Communications of the ACM, 43(8), 58-61. Retrieved 11 August, 2019, from https://www.researchgate.net/profile/Nicholas_Belkin/publication/220427131_Helping_people_find_what_they_don't_know/links/540db68d0cf2f2b29a39ff8d/Helping-people-find-what-they-dont-know.pdf

Brandes, U. (2001). A faster algorithm for betweenness centrality. Journal of mathematical sociology, 25(2), 163-177. Retrieved August 2, 2019, from https://www.tandfonline.com/doi/pdf/10.1080/0022250X.2001.9990249

Brandes, U., Borgatti, S. P., & Freeman, L. C. (2016). Maintaining the duality of closeness and betweenness centrality. Social Networks, 44, 153-159. Retrieved 15 August, 2019, from https://www.sciencedirect.com/science/article/pii/S0378873315000738

Bruneau, M., Eeri, M., Chang, S. E., Eguchi, R. T., Lee, G. C., O 'rourke, T. D., … Von Winterfeldt, D. (2003). A Framework to Quantitatively Assess and Enhance the Seismic Resilience of Communities. Earthquake Spectra, 19(4), 733–752. https://doi.org/10.1193/1.1623497

Chakhar, S., & Mousseau, V. (2008). GIS-based multicriteria spatial modeling generic framework. International Journal of Geographical Information Science, 22(11–12), 1159–1196. https://doi.org/10.1080/13658810801949827

City Form Lab. (2016). Urban Network Analysis - Help V1.01. Retrieved from http://media.voog.com/0000/0036/2451/files/20160120_UNA_help_v1_1.pdf

Coutinho-Rodrigues, J., Simão, A., & Antunes, C. H. (2011). A GIS-based multicriteria spatial decision support system for planning urban infrastructures. https://doi.org/10.1016/j.dss.2011.02.010

Daskin, M. S., & Dean, L. K. (2004). LOCATION OF HEALTH CARE FACILITIES. Operations Research and Health Care. Retrieved from http://84.89.132.1/~ramalhin/Referencias/Daskin_2004.pdf

Esri. (2017). Algorithms used by the ArcGIS Network Analyst extension—Help | ArcGIS Desktop. Retrieved December 5, 2017, from http://desktop.arcgis.com/en/arcmap/latest/extensions/network-analyst/algorithms-used-by-network-analyst.htm#ESRI_SECTION1_6FFC9C48F24746E182082F5DEBDBAA92

Farahani, R. Z., SteadieSeifi, M., & Asgari, N. (2010). Multiple criteria facility location problems: A survey. Applied Mathematical Modelling, 34(7), 1689–1709. https://doi.org/10.1016/J.APM.2009.10.005

GRASS Development Team, 2017. Geographic Resources Analysis Support System (GRASS 7) Programmer's Manual. Open Source Geospatial Foundation Project. Electronic document: http://grass.osgeo.org/programming7/

Li, X., & Yeh, A. G. (2005). Integration of genetic algorithms and GIS for optimal location search. International Journal of Geographical Information Science, 19(5), 581–601. https://doi.org/10.1080/13658810500032388

Malczewski, J. (2004). GIS-based land-use suitability analysis: a critical overview. Progress in Planning, 62, 3–65. https://doi.org/10.1016/S0305-9006(03)00079-5

Maniezzo, V., Mendes, I., & Paruccini, M. (1998). Decision support for siting problems. Decision Support Systems, 23(3), 273–284. https://doi.org/10.1016/S0167-9236(98)00042-6

Mohammed Tawfeeq. (2017). Mosul's bridges destroyed by ISIS as troops advance - CNN. CNN. Retrieved from http://edition.cnn.com/2017/01/13/middleeast/iraq-mosul-troops-advance/

Moradian, M. J., Ardalan, A., Nejati, A., Boloorani, A. D., Akbarisari, A., & Rastegarfar, B. (2017). Risk Criteria in Hospital Site Selection: A Systematic Review. PLOS Currents Disasters. https://doi.org/10.1371/CURRENTS.DIS.A6F34643F3CD22C168B8C6F2DEEAE86D

Moradian, M. J., Ardalan, A., Nejati, A., Darvishi Boloorani, A., Akbarisari, A., & Rastegarfar, B. (2016). Importance of Site Selection for Stockpiling Field Hospitals for Upcoming Disasters. Bulletin of Emergency and Trauma, 4(3), 124–5. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/27540545

Morrill, R. L., & Symons, J. (1977). Efficiency and Equity Aspects of Optimum Location. Geographical Analysis, 9(3), 215–225. https://doi.org/10.1111/j.1538-4632.1977.tb00575.x

Nickerson, J. W. (2015). Ensuring the security of health care in conflict settings: an urgent global health concern. CMAJ, 187(11), E347-E348. Retrieved from: http://www.cmaj.ca/content/cmaj/187/11/E347.full.pdf?casa_token=Or1toNcGsJAAAAAA:gfS1ZQDlzNLH40BQOXyoa8zXEIiti2SbX8HU2nMotQrkcWnQKAObOoS7s1KqkakFDLFWjqEg1H8-

Rahman, S.-U., & Smith, D. K. (2000). Use of location-allocation models in health service development planning in developing nations. European Jounal of Operational Research. Retrieved from https://pdfs.semanticscholar.org/5aac/80259a602bd5ca5551ba98c3db4f82b9d851.pdf

Rydén, M. (2011). Strategic Placing of Field Hospitals Using Spatial Analysis. Retrieved from http://kth.diva-portal.org/smash/get/diva2:420334/FULLTEXT02.pdf

Saaty, T. L., & Peniwati, K. (2008). Group decision making : drawing out and reconciling differences. RWS Publications. Retrieved from https://books.google.com.sg/books/about/Group_Decision_Making.html?id=phLWKwAACAAJ

Sprague, R. H., & Carlson, E. D. (1982). Building effective decision support systems. Prentice-Hall. Retrieved from https://books.google.nl/books/about/Building_Effective_Decision_Support_Syst.html?id=89_sPTAmkVsC&redir_esc=y

Sugumaran, R., & Degroote, J. (2010). Spatial decision support systems: principles and practices. Crc Press. Retrieved from: https://www.taylorfrancis.com/books/9780429148095

Tien, J. M., El-Tell, K., & Simons, G. R. (1983). Improved formulations to the hierarchical health facility location-allocation problem. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13(6), 1128–1132. https://doi.org/10.1109/TSMC.1983.6313187

UNAMI. (2014). Report on the Protection of Civilians in Armed Conflict in Iraq. Retrieved from http://www.ohchr.org/Documents/Countries/IQ/UNAMI_OHCHR_POC_Report_FINAL_6July_10September2014.pdf

Van Someren, M. W., Barnard, Y. F., & Sandberg, J. A. C. (1994). The think aloud method: a practical approach to modelling cognitive. London: Academic Press. Retrieved from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.821.4127&rep=rep1&type=pdf

WHO. (2017). WHO Special Situation Report, Mosul Crisis, Iraq. Retrieved from http://www.who.int/hac/crises/irq/sitreps/iraq-mosul-health-situation-report-23july2017.pdf?ua=1

WHO-PAHO. (2003). WHO-PAHO Guidelines for the Use of Foreign Field Hospitals in the Aftermath of Sudden-Impact Disasters. Retrieved November 16, 2017, from http://www.paho.org/disasters/index.php?option=com_content&view=article&id=674:pahowho-guidelines-for-the-use-of-foreign-field-hospitals&Itemid=924&lang=en

Xu, X., Chen, A., Jansuwan, S., Heaslip, K., & Yang, C. (2015). ScienceDirect 21st International Symposium on Transportation and Traffic Theory Modeling Transportation Network Redundancy. Transportation Research Procedia, 9, 283–302. https://doi.org/10.1016/j.trpro.2015.07.016

Zhu, X., R. J. Aspinall, and R. G. Healey. (1996). ILUDSS: A knowledge-based spatial decision support system for strategic land-use planning. Computers and Electronics in Agriculture 15(4):279–301.

Appendix A: Testing Instructions (Stakeholder Profiles)

Planning
Introduction of plug-in (5 minutes)
Study the profiles (10 minutes)
Ask question about the profiles (5 minutes)
First part of the exercise (15 minutes)
Second part of the exercise (15 minutes)
Oral feedback and discussion (10 minutes)

General
The Islamic State of Iraq and the Levant (ISIL) seized the city Mosul, Iraq in June 2014. This led to attempts to retake and free the city from 2015 till 2017 by Iraqi, Peshmerga and international forces. During this conflict, many inhabitants of Mosul have been displaced from their homes. Together with its partners, the World Health Organization (WHO) has been providing medical aid during the Mosul crisis by field hospital deployment. These field hospitals are mobile which allows them to change locations, should an alteration in the ground situation ask for it.
The situation on the ground can change in various ways, as you will experience later during this experiment. A multi-disciplinary group consisting of different stakeholders, such as WHO staff, armed forces and GIS experts together decide at which locations the available field hospitals can be deployed best. Each of these stakeholders has their preferences and opinion when it comes to where the field hospitals are going to be set up. In this experiment, the three of you represent different parties in this decision-making process, namely the WHO, allied armed forces and a GIS expert.
You are aided by a digital decision (touch) table in your decision-making, which will show a map of the Mosul city and potential field hospital sites. The software running on the table is designed to guide you, the decision-makers, through the process of finding the best locations for field hospitals. It does so by allowing you to easily add, move or delete field hospitals or dangerous territory. You also have the option of toggling roads or hospitals inactive, which you would do, for example, when a bridge is destroyed. It could also be the case that regular hospitals are also active in the city.
As decision-makers, each of you values different properties of field hospital locations. Seven different field hospital performance indicators are shown to reflect your different interests. They are the connectivity and resiliency of a field hospital on a scale of 0 to 100, the area (m2) a field hospital is covering with its location, the amount of population in that area, the average distance (meters) of the covered population to their (field) hospital and distance to danger and water. The ‘between’ indicator is a measure that indicates how passable the location is if you want to get from any point A to any point B in the city. Major roads are likely to have a better ‘between’ value. ‘Degree’ is an approximation of how many roads can be blocked in the vicinity of a field hospital without it being cut off from the rest of the city. Any road segments or (field) hospitals that are toggled inactive are excluded from the performance indicator calculations. Roads and population covered by dangerous territory are also considered inactive and neglected.

WHO
You will be representing the World Health Organization in this experiment. You manage the field hospitals, make sure they are occupied by medical personnel and that they are provided with sufficient supplies to daily service a certain amount of people (capacity). In this experiment your most important goal is to fully use the capacity of each hospital (‘population’ performance indicator shows the serviced population). You almost never have enough resources to provide medical service to everybody who is in need. Therefore, it is not acceptable when a field hospital is servicing only a part of its capacity. Second most important to you is minimizing the average distance people have to travel to get to a field hospital (‘mean pop distance’ performance indicator). Your field hospitals also distribute small water filtering systems for when people have no access to drinkable water. These have little use when there is no water nearby to filter, so minimizing the distance to water is your third goal (‘water distance’ performance indicator). But this third goal is less important. Besides field hospitals, some regular hospitals could still be in service. These cannot be moved and have to be taken into account when deciding on where to put field hospitals.

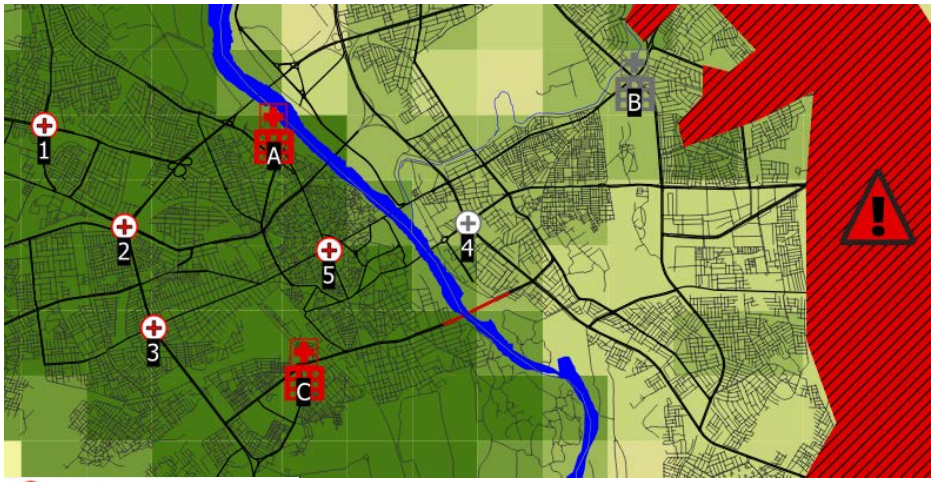
Goals summary
- Primary: Utilize full (field) hospital capacity
- Secondary: Minimize pop. travel distance
- Tertiary: Minimize water distance

- Existing hospitals cannot be added, moved or deleted, but can be toggled (in)active

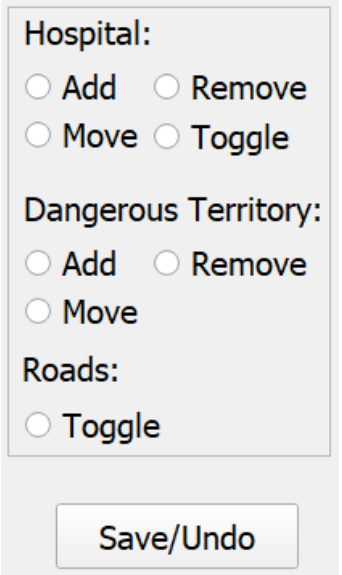
Allied armed forces
You will be representing allied armed forces in this experiment. Your task is to make sure that field hospitals are as safe from conflict as possible. To a certain extent, you can ensure the safety of field hospitals, but you only have a limited amount of defensive forces to do so. Of all the decision-makers, you know best which areas are safe, which are at risk and which are occupied by enemy forces. Unfortunately, the population in dangerous territory cannot be provided with health services. You have to make sure that field hospitals are not too close to dangerous territory and therefore, you try to minimize this distance (look at the 'danger distance' performance indicator). The closer the field hospitals are to danger, the more forces you need to provide safety. Within 3 kilometres from danger is a rule of thumb for when defences are needed. Field hospitals can be placed closer than 3 kilometres to danger, but you only have forces to defend two field hospitals down to 1 kilometre from danger.
Goals summary:
- Primary: No (field) hospitals are allowed within 1 km of danger
- Secondary: max 2 hospitals within 3-1 km distance of danger
- Toggle existing hospitals inactive if they are within 1 km of danger
- Add danger areas if needed

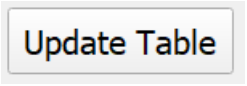
GIS specialist
You will be representing the GIS expert in this experiment. You have been chosen to do so because you are at least familiar with GIS. You will mostly be controlling the touch screen, and you lead the decision-making process. Because of your familiarity with the software, you are a kind of moderator in the experiment. If the other decision-makers struggle to understand the software, you might be able to explain it to them. As a GIS specialist, you are also familiar with street networks and how people flow through them. This gives you logistical insight in the decision-making process. You will also remind the medical and military stakeholders of the important difference between absolute distances, and distances when you have to travel over a street network. To be able to supply medical support and to defend a field hospital it should not be easy to cut it off from the whole street network. You ensure that field hospitals are central ('between' performance indicator) by being near main roads in the network. You also look at the resiliency ('degree' performance indicator) of a location in respect to the street network: are there sufficient routes for surrounding population to get to the field hospital? Or will a lot of people be disconnected when you remove one road? You could also help the military stakeholder in identifying points in the street network which could easily be cut off if dangerous territory were to expand. Keep in mind that a very good location for all field hospitals is not realistic.
Goals summary:
- You lead the decision-making process, are a sort of moderator and you operate the touch screen if the others struggle to understand the software
- Ensure that hospitals are well connected (near main roads, reflected by the 'between' indicator). Above 15% is very good, under 5% is bad.
- Ensure that hospitals have a resilient location (many escape routes, reflected by 'degree'). Above 30% is very good, under 15% is bad.
- You could possibly help recognize areas that will easily be 'cut off' from the street network if danger were to expand

Field Hospital Planner Interface
Below you can see a typical map for this plugin together with symbol explanations on the right. You have (in)active (field) hospitals, danger zones, a network of major and residential roads which can also be toggled inactive, water and population density.



It is possible to easily perform actions on (field) hospitals, dangerous territory and roads by clicking one of the white circles below. After that, you can press the map to apply the action. If you are done with the action, you first press the 'Save/Undo' button before you can do anything else.



Lastly, you can press the 

button to calculate performance indicators (see picture below) for each active (field) hospital on the map. Beware to only press this button if you are done adjusting objects on the map, as updating can take up to a few minutes.

Hospital	Degree	Between	Danger Distance	Water Distance	Population	Area Covered	Mean Pop. Distanc
A	11.66%	1.05%	5.6 km	0.2 km	73374	21 km²	1.0 km
C	29.3%	15.15%	6.7 km	2.4 km	232532	32 km²	2.2 km
1	23.35%	14.86%	9.1 km	2.8 km	127020	67 km²	5.6 km
2	35.7%	12.11%	8.3 km	2.7 km	79117	7 km²	3.4 km
3	29.41%	20.58%	8.6 km	3.6 km	143581	46 km²	1.4 km
4	29.57%	14.4%	3.5 km	0.7 km	101795	4 km²	1.4 km
5	26.94%	8.22%	5.7 km	1.0 km	219026	45 km²	2.3 km
Average	26.56%	12.34%	6.79 km	1.91 km	139492	31 km²	2 km

Further notes

- You can zoom in and out on the map just like you would by pinching on a smartphone.
- Try not to touch the decision table with multiple people at the same time.
- Try not to lean on the edges of the decision table, as it might cause the application to exit essential parts of the software.
- Thijs, the developer of the software in front of you, is not allowed to intervene in the experiment unless it really cannot proceed otherwise.
- You are being recorded to gain some context on why certain actions were performed on the decision table.
- You can press help buttons in the software to gain additional info on what you are seeing in the interface, especially about the performance indicators.
- For adding areas of danger, you can do so by tapping where the corners should be and you can finish an area by pressing and holding your finger for 2 seconds anywhere on the map.

Appendix B: Testing Script

Day 1
Proceed to the second window of the Field Hospital Planner where you can load data.

The data for day one is located at
C:/Users/Minou/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins/save_attributes/input_data/
Please select all the .shp files for population, roads, field- and existing hospitals, water and danger and press ‘proceed’.

It looks like the city of Mosul soon will be entirely freed from local insurgents. International forces have secured most of the city, but fighting remains in the East of the city. This has created a dangerous area. The long lasting fighting has also caused several of the bridges of Mosul to be destroyed. Three major hospitals are still standing in the city. But because hospital ‘B’ is so close to the fighting front, it has been decided to not use this hospital for the time being.

There currently is one field hospital on the ground. Please press ‘Update Table’ to see how this hospital performs. But the map is currently not up to date. Please select the toggle action for the roads and toggle the 2 southernmost bridges inactive. Save the adjustments and now select the toggle action for hospitals t toggle hospital ‘B’ inactive. Write down the ‘Between’ value of field hospital 1 and press ‘Update Table’ again to see how the field hospital performs now that the map has been updated. What has happened to the ‘Between’ value? As you can see the ‘between‘ value took a big hit now that the bridge is gone, it is less of a central place in the street network now. We will therefore remove field hospital 1, press the hospital remove action and remove it.

Although they are unrealistic numbers, for the sake of this simulation we assume that each existing hospital can daily serve 150.000 people and each field hospital 50.000 people. Besides field hospital 1, there are 5 other field hospitals at your disposal today. So 6 in total. Toggle ‘show population’ to see where there is demand for medical care. Toggle ‘show hospital service area’ to show what area of population a hospital serves. As you can see, most demand is located west of the Tigris river as many people have fled the East part of Mosul. Try to place all 6 field hospitals, keeping each of your goals in mind. You of course cannot move the existing hospitals, so try to keep this in mind.

Day 7
We are now a couple of days further. Your 6 field hospitals have been setup and running while fighting continued. All went well (or not? Could anticipate to their choices) and the danger in the east has fully retreated about 1 km eastwards. Select the danger move tool and click on one of the corners of the danger to move that corner. Save, but do not yet update the table.

Unfortunately another insurgency has developed on the Westside of the Tigris. Insurgents have been hiding among the population but have now gathered forces and raised their weapons. The new threat spans a danger area of about 1,5 km northwest of hospital A from the Tigris all the way to the end of the Major road straight West of hospital A. Add this danger with the danger add action. You will also have to toggle any field hospitals in this area inactive, as you cannot reach it anymore. If existing hospital B is further than 1km from danger, you can toggle it active again. Please update the table. The remaining hospitals probably are not in a very ideal position now. (possibly point at roads which have to be toggled inactive) Try to move the hospitals to better positions.

Appendix C: Main SDSS Python Script

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 Field Hospital Planner
                                 A QGIS plugin

Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-Builder/
                             -------------------
        begin                : 2019-05-28
        git sha              : $Format:%H$
        copyright            : (C) 2019 by Thijs
        email                : thijsvandercaaij@gmail.com
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""
import time
import os # This is is needed in the pyqgis console also
from qgis.core import (
   QgsVectorLayer, QgsLineSymbol
)
from PyQt5.QtCore import QSettings, QTranslator, qVersion, QCoreApplication, QUrl
from PyQt5.QtGui import QIcon, QColor, QFont
from PyQt5.QtWidgets import QAction, QFileDialog, QMessageBox, QTableWidgetItem,
QProgressBar, QProgressDialog
from PyQt5.QtMultimediaWidgets import QVideoWidget
from PyQt5.QtMultimedia import QMediaPlayer, QMediaContent
from qgis.core import * #

# Initialize Qt resources from file resources.py
from .resources import *
# Import the code for the dialog
from .save_attributes_dialog import *
import os.path
from .mapTools import *

class SaveAttributes:
   """QGIS Plugin Implementation."""

   def __init__(self, iface):
      """Constructor.

      :param iface: An interface instance that will be passed to this class
         which provides the hook by which you can manipulate the QGIS
         application at run time.
      :type iface: QgsInterface
      """
      # Save reference to the QGIS interface
      self.iface = iface
      # initialize plugin directory
```

```
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'SaveAttributes_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)

            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&Save Attributes')

        # Check if plugin was started the first time in current QGIS session
        # Must be set in initGui() to survive plugin reloads
        self.first_start = None

        # Editing instances
        self.editing = False
        self.modified = False
        #self.territoryLayer = QgsProject.instance().mapLayersByName('testpolygon')
        #self.territoryLayer = self.territoryLayer[0]
        #self.hospitalLayer = QgsProject.instance().mapLayersByName('Mosul')
        #self.hospitalLayer = self.hospitalLayer[0]
        self.roadsLayer = None
        self.hospitalLayer = None
        self.waterLayer = None
        self.dangerLayer = None
        self.populationLayer = None
        self.areaVoronoiLayer = None
        self.own_dir = os.path.dirname(os.path.abspath(__file__))
        self.step3count = 0
        self.step2count = 0
        self.radioButtonClicked = False
        self.saveButtonPressed = False
        self.roadsToggled = False
        self.roadsChanged = False
        self.dangerChanged = False
        #zoomFactor = 6
        #self.iface.mapCanvas().setWheelFactor( zoomFactor )
        #self.iface.mapCanvas().scaleChanged.connect(self.newScale)



    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
```

```
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('SaveAttributes', message)


    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.

        :param icon_path: Path to the icon for this action. Can be a resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
        :type icon_path: str

        :param text: Text that should be shown in menu items for this action.
        :type text: str

        :param callback: Function to be called when the action is triggered.
        :type callback: function

        :param enabled_flag: A flag indicating if the action should be enabled
            by default. Defaults to True.
        :type enabled_flag: bool

        :param add_to_menu: Flag indicating whether the action should also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool

        :param add_to_toolbar: Flag indicating whether the action should also
            be added to the toolbar. Defaults to True.
        :type add_to_toolbar: bool

        :param status_tip: Optional text to show in a popup when mouse pointer
            hovers over the action.
        :type status_tip: str

        :param parent: Parent widget for the new action. Defaults None.
        :type parent: QWidget

        :param whats_this: Optional text to show in the status bar when the
            mouse pointer hovers over the action.

        :returns: The action that was created. Note that the action is also
            added to self.actions list.
        :rtype: QAction
        """

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)
```

```
        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            # Adds plugin icon to Plugins toolbar
            self.iface.addToolBarIcon(action)

        if add_to_menu:
            self.iface.addPluginToVectorMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

        icon_path = ':/plugins/save_attributes/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'Save Attributes as CSV'),
            callback=self.run,
            parent=self.iface.mainWindow())

        # will be set False in run()
        self.first_start = True
        QgsProject.instance().clear()


    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginVectorMenu(
                self.tr(u'&Save Attributes'),
                action)
            self.iface.removeToolBarIcon(action)

#    def select_output_file(self):
#      filename, _filter = QFileDialog.getSaveFileName(
#        self.dlg, "Select output file ","", '*.csv')
#      self.dlg.lineEdit.setText(filename)
#
#    def select_input_file(self):
#      filename2, _filter2 = QFileDialog.getOpenFileName(
#        self.dlg2, "Select input file ","")
#      self.dlg2.pushButton_2.setText(filename2)

    def select_water_file(self):
      filename2, _filter2 = QFileDialog.getOpenFileName(
        self.dlg2.dock, "Select water input file ","")
      self.dlg2.dock.water_path.setText(filename2)

    def select_roads_file(self):
      filename2, _filter2 = QFileDialog.getOpenFileName(
```

```
        self.dlg2.dock, "Select roads input file ","")
      self.dlg2.dock.roads_path.setText(filename2)

    def select_population_file(self):
      filename2, _filter2 = QFileDialog.getOpenFileName(
        self.dlg2.dock, "Select population input file ","")
      self.dlg2.dock.population_path.setText(filename2)

    def select_danger_file(self):
      filename2, _filter2 = QFileDialog.getOpenFileName(
        self.dlg2.dock, "Select danger input file ","")
      self.dlg2.dock.danger_path.setText(filename2)

    def select_hospitals_file(self):
      filename2, _filter2 = QFileDialog.getOpenFileName(
        self.dlg2.dock, "Select field hospitals input file ","")
      self.dlg2.dock.hospitals_path.setText(filename2)

    def select_fieldhospitals_file(self):
      filename2, _filter2 = QFileDialog.getOpenFileName(
        self.dlg2.dock, "Select existing hospitals input file ","")
      self.dlg2.dock.fieldhospitals_path.setText(filename2)

    def about(self):
        """
        Visualize an About window.
        """
        QMessageBox.about(self, "About geoWeightedSum model", """ <p>Performs geographic multi-
criteria decision making using weighted sum model Documents and data are available in <a
href="http://maplab.alwaysdata.net/geomcda.html"> www.maplab.alwaysdata.net</a></p> <p>Author:
Gianluca Massei <a href="mailto:g_massa@libero.it">[g_massa at libero.it]</a></p> """)

    def load_layers(self):
        if self.step2count > 0:
            reply = QMessageBox.question(QMessageBox(), "Warning", "Do you really want to load the
entered layers again?", QMessageBox.Yes | QMessageBox.No, QMessageBox.Yes)
            if reply == QMessageBox.Yes:
                pass
            else:
                return


        self.message("Loading layers. Please wait...")
        population = self.dlg2.dock.population_path.text()
        population =
'C:/Users/Minou/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins/save_attributes/input
_data/population.shp'

        if population:
            try:
                QgsProject.instance().removeMapLayers( [self.populationLayer.id()] )
            except:
                pass
            self.populationLayer = QgsVectorLayer(population, "Population", "ogr")
            QgsProject.instance().addMapLayer(self.populationLayer)
            self.populationLayer.loadNamedStyle(self.own_dir + '/symbols/population_symbol.qml')

QgsProject.instance().layerTreeRoot().findLayer(self.populationLayer.id()).setItemVisibilityChecked(F
alse)
```

```
        hospitals = self.dlg2.dock.hospitals_path.text()
        fieldhospitals = self.dlg2.dock.fieldhospitals_path.text()
        #hospitals =
'C:/Users/Minou/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins/save_attributes/input
_data/mosul_points.shp'

        if hospitals or fieldhospitals or not self.hospitalLayer:
            try:
                QgsProject.instance().removeMapLayers( [self.hospitalLayer.id()] )
            except:
                pass
            # hard coded crs, could and should be soft coded by looking at road crs
            self.hospitalLayer = QgsVectorLayer("Point?crs=epsg:32638&field=id:string", "Hospitals",
"memory")

        #crs = self.hospitalLayer.crs()
        #crs.createFromId(32638)
        #self.hospitalLayer.setCrs(crs)
        if hospitals:
            try:
                QgsProject.instance().removeMapLayers( [self.tempHospitalLayer.id()] )
            except:
                pass
            self.tempHospitalLayer = QgsVectorLayer(hospitals, "Existing Hospitals", "ogr")
            temp_hospital_features = self.tempHospitalLayer.getFeatures()
            self.hospitalLayer.dataProvider().addFeatures(temp_hospital_features)
            self.hospitalLayer.updateFields()

        #add activity field if it does not yet exist
        active_index = self.hospitalLayer.fields().indexFromName('active')
        if active_index == -1:
            activityField = QgsField( 'active', QVariant.String )
            self.hospitalLayer.dataProvider().addAttributes([activityField])
            self.hospitalLayer.updateFields()

        #change activity field values to 'y'
        self.hospitalLayer.startEditing()
        for f in self.hospitalLayer.getFeatures():
            fieldIndex = f.fieldNameIndex( 'active' )
            self.hospitalLayer.changeAttributeValue(f.id(), fieldIndex, 'y')
        self.hospitalLayer.commitChanges()
        self.hospitalLayer.setDefaultValueDefinition(active_index, QgsDefaultValue('y'))

        if fieldhospitals:
            try:
                QgsProject.instance().removeMapLayers( [self.fieldhospitalsLayer.id()] )
            except:
                pass
            self.fieldhospitalsLayer = QgsVectorLayer(fieldhospitals, "Field Hospitals", "ogr")
            #QgsProject.instance().addMapLayer(self.fieldhospitalsLayer)
            existing_hospital_features = self.fieldhospitalsLayer.getFeatures()
            self.hospitalLayer.dataProvider().addFeatures(existing_hospital_features)
            self.hospitalLayer.updateFields()
#       #check for inactive roads
#       road_values = []
#       for feature in self.roadsLayer.getFeatures():
#           activity = feature['active']
#           road_values.append(activity)
#       QgsMessageLog.logMessage(activity, tag="debug" )
#       if 'n' in road_values:
```

```
#           fields = self.roadsLayer.fields()
#           tempLayer = QgsVectorLayer("LineString?crs=epsg:32638", "result", "memory")
#           query = QgsExpression('"active"=\'%s\'' % 'y')
#           features = self.roadsLayer.getFeatures(QgsFeatureRequest(query))
#           #tempLayer.startEditing()
#           tempLayer.dataProvider().addAttributes(fields)
#           tempLayer.dataProvider().addFeatures(features)
#           #tempLayer.commitChanges()
#           QgsProject.instance().addMapLayer(tempLayer)

        """ hospital styling """
        self.hospitalLayer.loadNamedStyle(self.own_dir + '/symbols/hospital_symbols.qml')
        self.hospitalLayer.loadNamedStyle(self.own_dir + '/symbols/hospital_label.qml')
#
#       # initialize the default symbol for this geometry type
#       active_symbol = QgsSymbol.defaultSymbol(self.hospitalLayer.geometryType())
#       inactive_symbol = QgsSymbol.defaultSymbol(self.hospitalLayer.geometryType())
#
#       active_symbol_path = self.own_dir + '/symbols/hospital active.svg'
#       active_hospital_symbol = QgsSvgMarkerSymbolLayer(active_symbol_path)
#       active_hospital_symbol.setSize(10)
#       active_symbol.changeSymbolLayer(0, active_hospital_symbol)
#
#       inactive_symbol_path = self.own_dir + '/symbols/hospital inactive.svg'
#       inactive_hospital_symbol = QgsSvgMarkerSymbolLayer(inactive_symbol_path)
#       inactive_hospital_symbol.setSize(10)
#       inactive_symbol.changeSymbolLayer(0, inactive_hospital_symbol)
#
#       active_category = QgsRendererCategory('y', active_symbol, 'Active')
#       inactive_category = QgsRendererCategory('n', inactive_symbol, 'Inactive')
#       categories = [active_category, inactive_category]
#
#       self.hospitalRenderer = QgsCategorizedSymbolRenderer('active', categories)
#
#       # assign the created renderer to the layer
#       if self.hospitalRenderer is not None:
#           self.hospitalLayer.setRenderer(self.hospitalRenderer)
#
#       # repaint the layer
#       self.hospitalLayer.triggerRepaint()
#       self.iface.layerTreeView().refreshLayerSymbology(self.hospitalLayer.id())

        # Connect the layer to the signal geometryChanged, so when a feature is
        # moved in the layer, self.modified is set to true and the edit mode reacts accordingly
        self.hospitalLayer.geometryChanged.connect(self.onModified)
        self.hospitalLayer.featureDeleted.connect(self.onModified)
        self.hospitalNames = []
        for f in self.hospitalLayer.getFeatures():
            self.hospitalNames.append(f['id'])


        roads = self.dlg2.dock.roads_path.text()
        roads =
'C:/Users/Minou/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins/save_attributes/input
_data/roads.shp'

        if roads:
            try:
                QgsProject.instance().removeMapLayers( [self.roadsLayer.id()] )
            except:
```

```
            pass
        self.roadsLayer = QgsVectorLayer(roads, "Roads", "ogr")
        QgsProject.instance().addMapLayer(self.roadsLayer)

    water = self.dlg2.dock.water_path.text()
    water =
'C:/Users/Minou/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins/save_attributes/input
_data/water.shp'

    if water:
        try:
            QgsProject.instance().removeMapLayers( [self.waterLayer.id()] )
        except:
            pass
        self.waterLayer = QgsVectorLayer(water, "Water", "ogr")
        QgsProject.instance().addMapLayer(self.waterLayer)
        water_symbol = QgsFillSymbol.createSimple({'color': 'blue'})

        # assign the created symbol to the layer
        self.waterLayer.renderer().setSymbol(water_symbol)

        # repaint the layer
        self.waterLayer.triggerRepaint()
        self.iface.layerTreeView().refreshLayerSymbology(self.waterLayer.id())

        #add water_dist field to hospitals if it does not yet exist
        active_index = self.hospitalLayer.fields().indexFromName('water_dist')
        if active_index == -1:
            activityField = QgsField( 'water_dist', QVariant.Double )
            self.hospitalLayer.dataProvider().addAttributes([activityField])
            self.hospitalLayer.updateFields()

        self.waterLayer.loadNamedStyle(self.own_dir + '/symbols/water_symbol.qml')

    danger = self.dlg2.dock.danger_path.text()
    danger =
'C:/Users/Minou/AppData/Roaming/QGIS/QGIS3/profiles/default/python/plugins/save_attributes/input
_data/danger.shp'
    if not danger:
        self.dangerLayer = QgsVectorLayer("Polygon?crs=epsg:32638&field=id:integer", "Danger",
"memory")

    if danger:
        try:
            QgsProject.instance().removeMapLayers( [self.dangerLayer.id()] )
        except:
            pass
        self.dangerLayer = QgsVectorLayer(danger, "Danger", "ogr")
        QgsProject.instance().addMapLayer(self.dangerLayer)
        #add dngr_dist field to hospitals if it does not yet exist
        active_index = self.hospitalLayer.fields().indexFromName('dngr_dist')
        if active_index == -1:
            activityField = QgsField( 'dngr_dist', QVariant.Double )
            self.hospitalLayer.dataProvider().addAttributes([activityField])
            self.hospitalLayer.updateFields()

        self.dangerLayer.geometryChanged.connect(self.onModified)
        self.dangerLayer.featureAdded.connect(self.dangerAdded)
        #self.dangerLayer.beforeAddingExpressionField.connect(self.skipExpression)
```

```
#           danger_symbol_path = self.own_dir + '/symbols/Danger.svg'
#           danger_symbol = QgsSVGFillSymbolLayer(danger_symbol_path)
#           #active_symbol.changeSymbolLayer(0, danger_symbol)
#           # assign the created symbol to the layer
#           self.dangerLayer.renderer().symbol().changeSymbolLayer(0, danger_symbol)
          self.dangerLayer.loadNamedStyle(self.own_dir + '/symbols/danger_symbol.qml')
          active_index = self.dangerLayer.fields().indexFromName('id')
          self.dangerLayer.setDefaultValueDefinition(active_index, QgsDefaultValue('1'))
          # repaint the layer
          #self.dangerLayer.triggerRepaint()
          #self.iface.layerTreeView().refreshLayerSymbology(self.dangerLayer.id())

#       # create a new symbol
#       active_roads_symbol = QgsLineSymbol.createSimple({'line_style': 'topo road', 'color': 'black'})
#       inactive_roads_symbol = QgsLineSymbol.createSimple({'line_style': 'topo road', 'color': 'red'})

       # apply symbol to layer renderer
       if roads:
          #add activity field if it does not yet exist
          active_index = self.roadsLayer.fields().indexFromName('active')
          if active_index == -1:
             activityField = QgsField( 'active', QVariant.String )
             self.roadsLayer.dataProvider().addAttributes([activityField])
             self.roadsLayer.updateFields()

          #change activity field values to 'y'
#           self.roadsLayer.startEditing()
#           for f in self.roadsLayer.getFeatures():
#              fieldIndex = f.fieldNameIndex( 'active' )
#              self.roadsLayer.changeAttributeValue(f.id(), fieldIndex, 'y')
#           self.roadsLayer.commitChanges()


#           active_category = QgsRendererCategory('y', active_roads_symbol, 'Active')
#           inactive_category = QgsRendererCategory('n', inactive_roads_symbol, 'Inactive')
#           categories = [active_category, inactive_category]
#
#           renderer = QgsCategorizedSymbolRenderer('active', categories)
#
#           # assign the created renderer to the layer
#           if renderer is not None:
#              self.roadsLayer.setRenderer(renderer)
#
#           # repaint the layer
#           self.roadsLayer.triggerRepaint()
#           self.iface.layerTreeView().refreshLayerSymbology(self.roadsLayer.id())
          self.roadsLayer.loadNamedStyle(self.own_dir + '/symbols/road_symbols.qml')
       #self.iface.mainWindow().blockSignals(True)
       QgsProject.instance().addMapLayer(self.hospitalLayer)
       #self.iface.mainWindow().blockSignals(False)
       self.iface.mapCanvas().setCanvasColor(QColor.fromHsv(60,0.35*255,0.96*255))

        # set custom render order of layers
       bridge = self.iface.layerTreeCanvasBridge().rootGroup()
       order = [self.hospitalLayer.id(), self.dangerLayer.id(), self.roadsLayer.id(), self.waterLayer.id(),
self.populationLayer.id()]
       bridge.setHasCustomLayerOrder(True)
       try:
          bridge.setCustomLayerOrderByIds( order )
       except:
```

```
            pass
        self.step2count += 1


        self.dlg3.dock.tableWidget.setColumnCount(8)
        bold_font = QFont('Times', weight = QFont.Bold)
        column_headers = ['Hospital', 'Population', 'Mean Pop. Distance', 'Danger Distance', 'Water
Distance', 'Area Covered', 'Between', 'Degree']
        self.dlg3.dock.tableWidget.setHorizontalHeaderLabels(column_headers)
        self.dlg3.dock.tableWidget.horizontalHeaderItem(0).setFont(bold_font)
        self.dlg3.dock.tableWidget.resizeColumnsToContents()

        self.zoomToFull()
        pixmap =
QPixmap(r'C:\Users\Minou\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins\save_attrib
utes\symbols\legend.png')
        self.legend.image.setPixmap(pixmap)
        self.legend.show()

#       if not self.hospitalLayer.isValid():
#           print("Layer failed to load!")
#       self.iface.messageBar().pushMessage(
#           "Success, layers have been added!", duration=3)

    def newScale(self, scale):
        if scale > 110000:
            self.iface.mapCanvas().zoomScale(110000.0)

    def setEditMode(self):
        self.iface.setActiveLayer(self.layer)
        if self.editing:
            if self.modified:
                reply = QMessageBox.question(QMessageBox(), "Confirm", "Save Changes?",
QMessageBox.Yes | QMessageBox.No, QMessageBox.Yes)
                if reply == QMessageBox.Yes:
                    self.layer.commitChanges()
                    if self.layer == self.dangerLayer:
                        self.dangerChanged = True

                else:
                    self.layer.rollBack()
                    if self.layer == self.roadsLayer:
                        #self.message("ja dit is een weg")
                        self.roadsChanged = False
                    if self.layer == self.dangerLayer:
                        self.dangerChanged = False
                self.editing = False
            else:
                self.layer.commitChanges()
                self.layer.triggerRepaint()
                self.editing = False
                #self.iface.actionPan().trigger()
        else:
                self.layer.startEditing()
                self.layer.triggerRepaint()
                self.editing = True
                self.modified = False
                #self.iface.actionPan().trigger()
                #self.adjustActions()
```

```
    def onModified(self):
        self.modified = True

    def onRoadModified(self):
        self.modified = True
        self.roadsChanged = True

    def dangerAdded(self, fid):
        request = QgsFeatureRequest()
        request.setFilterFids([fid])
        features = self.dangerLayer.getFeatures(request)
        for f in features:
            geom = f.geometry()
            if geom.isGeosValid():
                self.modified = True
                #self.message('Danger added')
            else:
                self.dangerLayer.rollBack()
                self.editing = False
                self.saveButtonPressed = True
                self.radioButtonClicked = False
                # enable all radiobuttons
                self.dlg3.dock.hospitalAdd.setCheckable(True)
                self.dlg3.dock.hospitalMove.setCheckable(True)
                self.dlg3.dock.hospitalRemove.setCheckable(True)
                self.dlg3.dock.hospitalToggle.setCheckable(True)
                self.dlg3.dock.territoryAdd.setCheckable(True)
                self.dlg3.dock.territoryMove.setCheckable(True)
                self.dlg3.dock.territoryRemove.setCheckable(True)
                self.dlg3.dock.roadsToggle.setCheckable(True)

                self.iface.actionPan().trigger()

                # uncheck all radiobuttons
                self.dlg3.dock.hospitalAdd.setChecked(False)
                self.dlg3.dock.hospitalMove.setChecked(False)
                self.dlg3.dock.hospitalRemove.setChecked(False)
                self.dlg3.dock.hospitalToggle.setChecked(False)
                self.dlg3.dock.territoryAdd.setChecked(False)
                self.dlg3.dock.territoryMove.setChecked(False)
                self.dlg3.dock.territoryRemove.setChecked(False)
                self.dlg3.dock.roadsToggle.setChecked(False)
                reply = QMessageBox.question(QMessageBox(), "Warning", "Please create non-twisted
danger areas. Your editing is being rolled back.", QMessageBox.Ok, QMessageBox.Ok)
                if reply == QMessageBox.Ok:
                    return
                else:
                    return


    def onHospitalModified(self, featureId, geometryChange=None):
        #self.message(str(featureId))
        request = QgsFeatureRequest()
        request.setFilterFids([featureId])
        features = self.hospitalLayer.getFeatures(request)
        feature = None
        for f in features:
            feature = f
            feature_name = feature['id']
        if not feature:
```

```
            self.modified = True
            self.message('bijzonder geval, crash? modified')
            return
         #   new_hospital_names = []
          #  for f in self.hospitalLayer.getFeatures():
           #     new_hospital_names.append(f['id'])
            #feature_name = list(set(self.hospitalNames) - set(new_hospital_names))
        if not feature_name.isdigit():
            self.hospitalLayer.rollBack()
            self.editing = False
            self.saveButtonPressed = True
            self.radioButtonClicked = False
            # enable all radiobuttons
            self.dlg3.dock.hospitalAdd.setCheckable(True)
            self.dlg3.dock.hospitalMove.setCheckable(True)
            self.dlg3.dock.hospitalRemove.setCheckable(True)
            self.dlg3.dock.hospitalToggle.setCheckable(True)
            self.dlg3.dock.territoryAdd.setCheckable(True)
            self.dlg3.dock.territoryMove.setCheckable(True)
            self.dlg3.dock.territoryRemove.setCheckable(True)
            self.dlg3.dock.roadsToggle.setCheckable(True)

            self.iface.actionPan().trigger()

            # uncheck all radiobuttons
            self.dlg3.dock.hospitalAdd.setChecked(False)
            self.dlg3.dock.hospitalMove.setChecked(False)
            self.dlg3.dock.hospitalRemove.setChecked(False)
            self.dlg3.dock.hospitalToggle.setChecked(False)
            self.dlg3.dock.territoryAdd.setChecked(False)
            self.dlg3.dock.territoryMove.setChecked(False)
            self.dlg3.dock.territoryRemove.setChecked(False)
            self.dlg3.dock.roadsToggle.setChecked(False)
            self.message('You can not (re)move existing hospitals! You can only do this with field
hospitals.', 10)
        else:
            self.modified = True

    def onHospitalDeleted(self, featureId):
        #self.message(str(featureId))
        new_hospital_names = []
        for f in self.hospitalLayer.getFeatures():
            new_hospital_names.append(f['id'])
        feature_name = set(self.hospitalNames) - set(new_hospital_names)
        feature_name = next(iter(feature_name), None)
        if not feature_name:
            self.modified = True
            self.message('bijzonder geval, crash? deleted')
            return
        if not feature_name.isdigit():
            self.hospitalLayer.rollBack()
            self.editing = False
            self.saveButtonPressed = True
            self.radioButtonClicked = False
            # enable all radiobuttons
            self.dlg3.dock.hospitalAdd.setCheckable(True)
            self.dlg3.dock.hospitalMove.setCheckable(True)
            self.dlg3.dock.hospitalRemove.setCheckable(True)
            self.dlg3.dock.hospitalToggle.setCheckable(True)
            self.dlg3.dock.territoryAdd.setCheckable(True)
```

```
            self.dlg3.dock.territoryMove.setCheckable(True)
            self.dlg3.dock.territoryRemove.setCheckable(True)
            self.dlg3.dock.roadsToggle.setCheckable(True)

            self.iface.actionPan().trigger()

            # uncheck all radiobuttons
            self.dlg3.dock.hospitalAdd.setChecked(False)
            self.dlg3.dock.hospitalMove.setChecked(False)
            self.dlg3.dock.hospitalRemove.setChecked(False)
            self.dlg3.dock.hospitalToggle.setChecked(False)
            self.dlg3.dock.territoryAdd.setChecked(False)
            self.dlg3.dock.territoryMove.setChecked(False)
            self.dlg3.dock.territoryRemove.setChecked(False)
            self.dlg3.dock.roadsToggle.setChecked(False)
            self.message('You can not (re)move existing hospitals! You can only do this with field
hospitals.', 10)
        else:
            self.modified = True


    def setAddHospital(self):
        if self.radioButtonClicked == False:
            self.modified = False
            self.editing = False
            self.radioButtonChecked(self.dlg3.dock.hospitalAdd)
            self.layer = self.hospitalLayer
            self.iface.setActiveLayer(self.layer)
            self.setEditMode()
            self.addPoint = AddPointTool(self.iface.mapCanvas(), self.layer, self.onModified)
            self.iface.mapCanvas().setMapTool(self.addPoint)

    def setMoveHospital(self):
        if self.radioButtonClicked == False:
            self.modified = False
            self.editing = False
            self.radioButtonChecked(self.dlg3.dock.hospitalMove)
            self.layer = self.hospitalLayer
            self.iface.setActiveLayer(self.layer)
            self.setEditMode()
            self.moveTool = self.iface.actionVertexToolActiveLayer().trigger()
            #self.iface.actionVertexToolActiveLayer().trigger()

    def setRemoveHospital(self):
        if self.radioButtonClicked == False:
            self.modified = False
            self.editing = False
            self.radioButtonChecked(self.dlg3.dock.hospitalRemove)
            self.layer = self.hospitalLayer
            self.iface.setActiveLayer(self.layer)
            self.hospitalNames = []
            for f in self.layer.getFeatures():
                self.hospitalNames.append(f['id'])
            self.setEditMode()
            self.deletePoint = DeletePointTool(self.iface.mapCanvas(), self.layer, self.onModified)
            self.iface.mapCanvas().setMapTool(self.deletePoint)

    def setAddTerritory(self):
        if self.radioButtonClicked == False:
            self.modified = False
```

```
        self.editing = False
        self.radioButtonChecked(self.dlg3.dock.territoryAdd)
        self.layer = self.dangerLayer
        self.iface.setActiveLayer(self.layer)
        self.setEditMode()
        self.addTool = self.iface.actionAddFeature().trigger()

    def setMoveTerritory(self):
        if self.radioButtonClicked == False:
            self.modified = False
            self.editing = False
            self.radioButtonChecked(self.dlg3.dock.territoryMove)
            self.layer = self.dangerLayer
            self.iface.setActiveLayer(self.layer)
            self.setEditMode()
            self.moveTool = self.iface.actionVertexToolActiveLayer().trigger()

    def setRemoveTerritory(self):
        if self.radioButtonClicked == False:
            self.modified = False
            self.editing = False
            self.radioButtonChecked(self.dlg3.dock.territoryRemove)
            self.layer = self.dangerLayer
            self.iface.setActiveLayer(self.layer)
            self.setEditMode()
            self.deletePoint = DeletePointTool(self.iface.mapCanvas(), self.layer, self.onModified)
            self.iface.mapCanvas().setMapTool(self.deletePoint)

    def saveButtonClicked(self):
        if self.saveButtonPressed == False:
            self.saveButtonPressed = True
            self.radioButtonClicked = False
            # enable all radiobuttons
            self.dlg3.dock.hospitalAdd.setCheckable(True)
            self.dlg3.dock.hospitalMove.setCheckable(True)
            self.dlg3.dock.hospitalRemove.setCheckable(True)
            self.dlg3.dock.hospitalToggle.setCheckable(True)
            self.dlg3.dock.territoryAdd.setCheckable(True)
            self.dlg3.dock.territoryMove.setCheckable(True)
            self.dlg3.dock.territoryRemove.setCheckable(True)
            self.dlg3.dock.roadsToggle.setCheckable(True)

            self.setEditMode()
            self.iface.actionPan().trigger()

            # uncheck all radiobuttons
            self.dlg3.dock.hospitalAdd.setChecked(False)
            self.dlg3.dock.hospitalMove.setChecked(False)
            self.dlg3.dock.hospitalRemove.setChecked(False)
            self.dlg3.dock.hospitalToggle.setChecked(False)
            self.dlg3.dock.territoryAdd.setChecked(False)
            self.dlg3.dock.territoryMove.setChecked(False)
            self.dlg3.dock.territoryRemove.setChecked(False)
            self.dlg3.dock.roadsToggle.setChecked(False)

    def radioButtonChecked(self, checkedbutton):
        self.radioButtonClicked = True
        self.saveButtonPressed = False
        # disable all radiobuttons
        self.dlg3.dock.hospitalAdd.setCheckable(False)
```

```
        self.dlg3.dock.hospitalMove.setCheckable(False)
        self.dlg3.dock.hospitalRemove.setCheckable(False)
        self.dlg3.dock.hospitalToggle.setCheckable(False)
        self.dlg3.dock.territoryAdd.setCheckable(False)
        self.dlg3.dock.territoryMove.setCheckable(False)
        self.dlg3.dock.territoryRemove.setCheckable(False)
        self.dlg3.dock.roadsToggle.setCheckable(False)
        checkedbutton.setCheckable(True)
        checkedbutton.setChecked(True)


    def setToggleRoad(self):
        if self.radioButtonClicked == False:
            self.modified = False
            self.editing = False
            self.radioButtonChecked(self.dlg3.dock.roadsToggle)
            self.layer = self.roadsLayer
            self.iface.setActiveLayer(self.layer)
            self.setEditMode()
            #self.layer.startEditing()
            self.toggleRoad = ToggleTool(self.iface.mapCanvas(), self.layer, self.onRoadModified)
            self.iface.mapCanvas().setMapTool(self.toggleRoad)

    def setToggleHospital(self):
        if self.radioButtonClicked == False:
            self.modified = False
            self.editing = False
            self.radioButtonChecked(self.dlg3.dock.hospitalToggle)
            self.layer = self.hospitalLayer
            self.iface.setActiveLayer(self.layer)
            self.setEditMode()
            #self.layer.startEditing()
            self.toggleHospital = ToggleTool(self.iface.mapCanvas(), self.layer, self.onModified)
            self.iface.mapCanvas().setMapTool(self.toggleHospital)

    #go from dlg to dlg2
    def nextStep1(self):
        self.dlg.dock.close()
        self.dlg2.iface.addDockWidget( Qt.RightDockWidgetArea, self.dlg2.dock )

    #go from dlg2 to dlg
    def backStep1(self):
        self.dlg2.dock.close()
        self.dlg.iface.addDockWidget( Qt.RightDockWidgetArea, self.dlg.dock )

    #go from dlg2 to dlg3
    def nextStep2(self):
        self.dlg2.dock.close()
        self.dlg3.iface.addDockWidget( Qt.RightDockWidgetArea, self.dlg3.dock )
        self.load_layers()

    #go from dlg3 to dlg2
    def backStep2(self):
        if self.radioButtonClicked:
            reply = QMessageBox.question(QMessageBox(), "Warning", "Please press the 'Save/Undo'
button before adding/replacing data", QMessageBox.Ok, QMessageBox.Ok)
            if reply == QMessageBox.Ok:
                return
            else:
                return
```

```
        self.dlg3.dock.close()
        self.dlg2.iface.addDockWidget( Qt.RightDockWidgetArea, self.dlg2.dock )

    def quitdlg(self):
        reply = QMessageBox.question(QMessageBox(), "Confirm", "Do you really want to quit the Field
Hospital Planner?\nYou will lose any loaded layers", QMessageBox.Yes | QMessageBox.No,
QMessageBox.Yes)
        if reply == QMessageBox.Yes:
            self.dlg.dock.close()
            QgsProject.instance().clear()
        else:
            return

    def quitdlg2(self):
        reply = QMessageBox.question(QMessageBox(), "Confirm", "Do you really want to quit the Field
Hospital Planner?\nYou will lose any loaded layers", QMessageBox.Yes | QMessageBox.No,
QMessageBox.Yes)
        if reply == QMessageBox.Yes:
            self.dlg2.dock.close()
            QgsProject.instance().clear()
        else:
            return

    def quitdlg3(self):
        reply = QMessageBox.question(QMessageBox(), "Confirm", "Do you really want to quit the Field
Hospital Planner?\nYou will lose any loaded layers", QMessageBox.Yes | QMessageBox.No,
QMessageBox.Yes)
        if reply == QMessageBox.Yes:
            self.dlg3.dock.close()
            QgsProject.instance().clear()
        else:
            return


    def message(self, text, duration=3):
        self.iface.messageBar().pushMessage(text, duration=duration)

    def tempStep(self):
        """ adds danger_id = 1 to those population features that are covered by danger """
        import processing
        filename = self.own_dir + '/temp_population' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = { 'DISCARD_NONMATCHING' : False,
                    'INPUT' : self.populationLayer,
                    'JOIN' : self.dangerLayer,
                    'JOIN_FIELDS' : [],
                    'METHOD' : 0,
                    'OUTPUT' : filename,
                    'PREDICATE' : [0],
                    'PREFIX' : 'danger_' }

        processing.run('qgis:joinattributesbylocation', parameters)



    def progdialog(self, progress):
```

```
        dialog = QProgressDialog()
        dialog.setWindowTitle("Progress")
        dialog.setLabelText("please wait")
        dialog.setWindowFlags(Qt.WindowStaysOnTopHint)
        bar = QProgressBar(dialog)
        bar.setTextVisible(True)
        bar.setValue(progress)
        dialog.setBar(bar)
        dialog.setMinimumWidth(300)
        dialog.show()
        return dialog, bar

    #go from dlg3 to dlg4
    def nextStep3(self):
        # check if edits are saved
        if self.radioButtonClicked:
            reply = QMessageBox.question(QMessageBox(), "Warning", "Please press the 'Save/Undo'
button before updating the table", QMessageBox.Ok, QMessageBox.Ok)
            if reply == QMessageBox.Ok:
                return
            else:
                return

        dialog, bar = self.progdialog(0)
        bar.setValue(0)
        bar.setMaximum(100)
        if self.dangerChanged or self.roadsChanged or self.step3count == 0:
            self.dangerLocations()
            self.tempPopulationLayer = QgsVectorLayer(self.own_dir + '/temp_population' +
str(self.step3count) + '.shp', "Temp population", "ogr")
            query = QgsExpression('"danger_id"=\'%s\'' % '1')
            request = QgsFeatureRequest(query)
            request.setSubsetOfAttributes([])
            request.setFlags(QgsFeatureRequest.NoGeometry)
            self.tempPopulationLayer.startEditing()
            for f in self.tempPopulationLayer.getFeatures(request):
                self.tempPopulationLayer.deleteFeature(f.id())
            self.tempPopulationLayer.commitChanges()

            self.tempRoadsLayer = QgsVectorLayer(self.own_dir + '/temp_roads' + str(self.step3count) +
'.shp', "Temp roads", "ogr")
            query = QgsExpression('"danger_id"=\'%s\'' % '1')
            request = QgsFeatureRequest(query)
            request.setSubsetOfAttributes([])
            request.setFlags(QgsFeatureRequest.NoGeometry)
            self.tempRoadsLayer.startEditing()
            for f in self.tempRoadsLayer.getFeatures(request):
                self.tempRoadsLayer.deleteFeature(f.id())
            self.tempRoadsLayer.commitChanges()

        self.tempHospitalLayer2 = QgsVectorLayer("Point?crs=epsg:32638", "Temp hospitals",
"memory")
        query = QgsExpression('"active"=\'%s\'' % 'y')
        request = QgsFeatureRequest(query)
        features = self.hospitalLayer.getFeatures(request)
        self.tempHospitalLayer2.dataProvider().addAttributes(self.hospitalLayer.fields())
        self.tempHospitalLayer2.dataProvider().addFeatures(features)
        self.tempHospitalLayer2.updateFields()
```

```
        self.voronoi()
        self.voronoiLayer = QgsVectorLayer(self.own_dir + '/voronoi' + str(self.step3count) + '.shp',
"Voronoi", "ogr")
        #QgsProject.instance().addMapLayer(self.voronoiLayer)


        self.centroids()
        self.popCentroidsLayer = QgsVectorLayer(self.own_dir + '/pop_centroids' + str(self.step3count)
+ '.shp', "Pop_centroids", "ogr")
#         QgsProject.instance().addMapLayer(self.popCentroidsLayer)

        self.population_distance()
        self.popCentroidsLayer = QgsVectorLayer(self.own_dir + '/population_nearest' +
str(self.step3count) + '.shp', "Pop_centroids", "ogr")


        self.count_population()
        self.popVoronoiLayer = QgsVectorLayer(self.own_dir + '/pop_voronoi' + str(self.step3count) +
'.shp', "Pop_voronoi", "ogr")
#         QgsProject.instance().addMapLayer(self.popVoronoiLayer)

        bar.setValue(5)

        self.clip()
        self.trimmedVoronoiLayer = QgsVectorLayer(self.own_dir + '/pop_voronoi_trimmed' +
str(self.step3count) + '.shp', "Pop_voronoi_trimmed", "ogr")
        #QgsProject.instance().addMapLayer(self.trimmedVoronoiLayer)

        self.area()
        try:
            QgsProject.instance().removeMapLayers( [self.areaVoronoiLayer.id()] )
        except:
            pass
        self.areaVoronoiLayer = QgsVectorLayer(self.own_dir + '/pop_voronoi_area' +
str(self.step3count) + '.shp', "Pop_voronoi_area", "ogr")
        self.areaVoronoiLayer.loadNamedStyle(self.own_dir + '/symbols/service_area_symbol.qml')
        QgsProject.instance().addMapLayer(self.areaVoronoiLayer)

QgsProject.instance().layerTreeRoot().findLayer(self.areaVoronoiLayer.id()).setItemVisibilityChecked(
False)

        self.danger_distance()
        #QgsProject.instance().removeMapLayers( [self.hospitalLayer.id()] )

        self.hospitalDangerLayer = QgsVectorLayer(self.own_dir + '/danger_nearest' +
str(self.step3count) + '.shp', "Hospitals", "ogr")
        #QgsProject.instance().addMapLayer(self.hospitalLayer)

        self.water_distance()
        #QgsProject.instance().removeMapLayers( [self.hospitalLayer.id()] )
        self.hospitalWaterLayer = QgsVectorLayer(self.own_dir + '/water_nearest' + str(self.step3count)
+ '.shp', "Hospitals", "ogr")
        #QgsProject.instance().addMapLayer(self.hospitalLayer)

        self.buffer()
        self.hospitalBuffer = QgsVectorLayer(self.own_dir + '/hospital_centrality_buffer' +
str(self.step3count) + '.shp', "Hospital buffer", "ogr")

        #check for inactive roads
        #road_values = []
```

```
        #for feature in self.roadsLayer.getFeatures():
        #    activity = feature['active']
        #    road_values.append(activity)
        #QgsMessageLog.logMessage(activity, tag="debug" )
        # if there are inactive roads, make templayer with active ones only
        #if 'n' in road_values:
        self.roadsToggled = True
        query = QgsExpression('"active"=\'%s\'' % 'n')
        request = QgsFeatureRequest(query)
        request.setSubsetOfAttributes([])
        request.setFlags(QgsFeatureRequest.NoGeometry)
        self.tempRoadsLayer.startEditing()
        for f in self.tempRoadsLayer.getFeatures(request):
            self.tempRoadsLayer.deleteFeature(f.id())
        self.tempRoadsLayer.commitChanges()

        #QgsProject.instance().addMapLayer(self.tempRoadsLayer)
        bar.setValue(10)
        if self.roadsChanged or self.dangerChanged or self.step3count == 0:
            if not self.roadsChanged and not self.dangerChanged:
                #self.message(self.roadsChanged+self.dangerChanged)
                self.centralityLayer = QgsVectorLayer(self.own_dir + '/centrality_points' +
str(self.step3count) + '.shp', "Centrality points", "ogr")
            else:
            #self.message(self.roadsChanged+self.dangerChanged)
                self.centrality()
                self.centralityLayer = QgsVectorLayer(self.own_dir + '/centrality_points' +
str(self.step3count) + '.shp', "Centrality points", "ogr")

                # normalize centrality indicators
                degree_list = []
                between_list = []
                for feature in self.centralityLayer.getFeatures():
                    degree = feature['degree']
                    between = feature['betw']
                    degree_list.append(degree)
                    between_list.append(between)
                degree_min = min(degree_list)
                degree_max = max(degree_list)
                between_min = min(between_list)
                between_max = max(between_list)
                #QgsMessageLog.logMessage(str(between_min), tag="debug" )
                #QgsMessageLog.logMessage(str(between_max), tag="debug" )
                new_between_list = []
                for i in between_list:
                    p = (i-between_min)/(between_max-between_min)*100
                    new_between_list.append(p)
                new_degree_list = []
                for i in degree_list:
                    p = (i-degree_min)/(degree_max-degree_min)*100
                    new_degree_list.append(p)
                #QgsMessageLog.logMessage(str(between_min), tag="debug" )
                #QgsMessageLog.logMessage(str(between_max), tag="debug" )

                self.centralityLayer.startEditing()
                for f, between, degree in zip(self.centralityLayer.getFeatures(), new_between_list,
new_degree_list):
                    fieldIndex1 = f.fieldNameIndex( 'betw' )
                    self.centralityLayer.changeAttributeValue(f.id(), fieldIndex1, between)
                    fieldIndex2 = f.fieldNameIndex( 'degree' )
```

```
                self.centralityLayer.changeAttributeValue(f.id(), fieldIndex2, degree)
            self.centralityLayer.commitChanges()


        self.average_centrality()
        self.averageCentralityLayer = QgsVectorLayer(self.own_dir + '/average_centrality' +
str(self.step3count) + '.shp', "Centrality points", "ogr")
        #gsProject.instance().addMapLayer(self.averageCentralityLayer)
        self.join()
        self.manyIndicatorsLayer = QgsVectorLayer(self.own_dir + '/danger_pop_dist_area' +
str(self.step3count) + '.shp', "Many indicators", "ogr")


        """ table update """
        self.hospitalAmount = self.tempHospitalLayer2.featureCount()
        self.dlg3.dock.tableWidget.setRowCount(self.hospitalAmount)

        id_values = []
        danger_values = []
        water_values = []
        pop_values = []
        popdist_values = []
        area_values = []
        degree_values = []
        between_values = []
        for feature in self.averageCentralityLayer.getFeatures():
            degree = feature['degree_mea']
            between = feature['betw_mean']
            try:
                degree = round(degree, 2)
                between = round(between, 2)
            except:
                degree = 0
                between = 0
            degree_values.append(degree)
            between_values.append(between)

        for feature in self.manyIndicatorsLayer.getFeatures():
            danger = feature["dngr_dist"]
            danger = float(danger)
            danger = round(danger/1000, 1)
            danger_values.append(danger)

            population = feature["TotPop_sum"]
            popdist = feature["Host_Pop_m"]
            population = int(population)
            popdist = round(popdist/1000, 1)
            pop_values.append(population)
            popdist_values.append(popdist)

            fid = feature["id"]
            id_values.append(fid)

            area = feature["area"]
            # m2 to km2
            area = int(area/1000000)
            area_values.append(area)

        #for feature in self.tempHospitalLayer2.getFeatures():
```

```
        for feature in self.hospitalWaterLayer.getFeatures():
            water = feature["water_dist"]
            water = float(water)
            water = round(water/1000, 1)
            water_values.append(water)

#       for feature in self.areaVoronoiLayer.getFeatures():
#           population = feature["TotPop_sum"]
#           popdist = feature["Host_Pop_m"]
#           population = int(population)
#           popdist = round(popdist/1000, 1)
#           pop_values.append(population)
#           popdist_values.append(popdist)
#
#           fid = feature["id"]
#           id_values.append(fid)
#
#           area = feature["area"]
#           # m2 to km2
#           area = int(area/1000000)
#           area_values.append(area)

#       for feature in self.areaVoronoiLayer.getFeatures():
#           area = feature["area"]
#           # m2 to km2
#           area = int(area/1000000)
#           area_values.append(area)

        id_values = map(str, id_values)
        danger_values = map(str, danger_values)
        water_values = map(str, water_values)
        pop_values = map(str, pop_values)
        area_values = map(str, area_values)
        degree_values = map(str, degree_values)
        between_values = map(str, between_values)
        popdist_values = map(str, popdist_values)

        for hospital, value in enumerate(id_values):
            cellItem = QTableWidgetItem()
            cellItem.setText(value)
            cellItem.setFlags(Qt.ItemIsEnabled)
            cellItem.setTextAlignment(Qt.AlignCenter)
            cellItem.setFont(QFont('Times', weight = QFont.Bold))
            self.dlg3.dock.tableWidget.setItem(hospital,0,cellItem)

        for hospital, value in enumerate(danger_values):
            cellItem = QTableWidgetItem()
            cellItem.setText(value)
            cellItem.setFlags(Qt.ItemIsEnabled)
            cellItem.setTextAlignment(Qt.AlignCenter)
            self.dlg3.dock.tableWidget.setItem(hospital,3,cellItem)

        for hospital, value in enumerate(water_values):
            cellItem = QTableWidgetItem()
            cellItem.setText(value)
            cellItem.setFlags(Qt.ItemIsEnabled)
            cellItem.setTextAlignment(Qt.AlignCenter)
            self.dlg3.dock.tableWidget.setItem(hospital,4,cellItem)
```

```
for hospital, value in enumerate(pop_values):
    cellItem = QTableWidgetItem()
    cellItem.setText(value)
    cellItem.setFlags(Qt.ItemIsEnabled)
    cellItem.setTextAlignment(Qt.AlignCenter)
    self.dlg3.dock.tableWidget.setItem(hospital,1,cellItem)

for hospital, value in enumerate(area_values):
    cellItem = QTableWidgetItem()
    cellItem.setText(value)
    cellItem.setFlags(Qt.ItemIsEnabled)
    cellItem.setTextAlignment(Qt.AlignCenter)
    self.dlg3.dock.tableWidget.setItem(hospital,5,cellItem)

for hospital, value in enumerate(popdist_values):
    cellItem = QTableWidgetItem()
    cellItem.setText(value)
    cellItem.setFlags(Qt.ItemIsEnabled)
    cellItem.setTextAlignment(Qt.AlignCenter)
    self.dlg3.dock.tableWidget.setItem(hospital,2,cellItem)

for hospital, value in enumerate(degree_values):
    cellItem = QTableWidgetItem()
    cellItem.setText(value)
    cellItem.setFlags(Qt.ItemIsEnabled)
    cellItem.setTextAlignment(Qt.AlignCenter)
    self.dlg3.dock.tableWidget.setItem(hospital,7,cellItem)

for hospital, value in enumerate(between_values):
    cellItem = QTableWidgetItem()
    cellItem.setText(value)
    cellItem.setFlags(Qt.ItemIsEnabled)
    cellItem.setTextAlignment(Qt.AlignCenter)
    self.dlg3.dock.tableWidget.setItem(hospital,6,cellItem)

# add row to bottom of table with sums
rowcount = self.dlg3.dock.tableWidget.rowCount()
self.dlg3.dock.tableWidget.insertRow(rowcount)
columncount = self.dlg3.dock.tableWidget.columnCount()
for column in range(columncount):
    if column == 0:
        continue
    values_in_column = []
    for row in range(rowcount):
        try:
            value = self.dlg3.dock.tableWidget.item(row, column).text()
            if 5 < column < 8:
                self.dlg3.dock.tableWidget.item(row, column).setText(value+'%')
            if 1 < column < 5:
                self.dlg3.dock.tableWidget.item(row, column).setText(value+' km')
            if column == 5:
                self.dlg3.dock.tableWidget.item(row, column).setText(value+' km\u00b2')


        except:
            pass
        #QgsMessageLog.logMessage(str(value), tag="debug" )
        values_in_column.append(value)
    values_in_column = map(float, values_in_column)
    column_sum = sum(values_in_column)
```

```
            column_average = float(round((column_sum / rowcount), 2))
            # self.iface.messageBar().pushMessage(str(column_sum), duration=3)
            item = QTableWidgetItem(str(int(column_average)))
            if 5 < column < 8:
                item = QTableWidgetItem(str(column_average)+'%')
            if 1 < column < 5:
                item = QTableWidgetItem(str(column_average)+' km')
            if column == 5:
                item = QTableWidgetItem(str(int(column_average))+' km\u00b2')

            item.setTextAlignment(Qt.AlignCenter)
            item.setFlags(Qt.ItemIsEnabled)
            self.dlg3.dock.tableWidget.setItem(rowcount , column, item)
        cellItem = QTableWidgetItem('Average')
        cellItem.setFont(QFont('Times', weight = QFont.Bold))
        cellItem.setTextAlignment(Qt.AlignCenter)
        self.dlg3.dock.tableWidget.setItem(rowcount , 0, cellItem)

        self.dlg3.dock.tableWidget.resizeColumnsToContents()
        self.dlg3.dock.tableWidget.verticalHeader().setVisible(False)

#       """ hospital styling """
#       self.hospitalLayer.loadNamedStyle(self.own_dir + '/symbols/hospital_symbols.qml')
#       self.hospitalLayer.loadNamedStyle(self.own_dir + '/symbols/hospital_label.qml')
#
#       """ reconnect hospital signals """
#       self.hospitalLayer.geometryChanged.connect(self.onHospitalModified)
#       self.hospitalLayer.featureDeleted.connect(self.onHospitalDeleted)

        # set custom render order of layers
        bridge = self.iface.layerTreeCanvasBridge().rootGroup()
        order = [self.hospitalLayer.id(), self.dangerLayer.id(), self.roadsLayer.id(), self.waterLayer.id(),
self.populationLayer.id(), self.areaVoronoiLayer.id()]
        bridge.setHasCustomLayerOrder(True)
        bridge.setCustomLayerOrderByIds( order )
        self.iface.mapCanvas().refresh()

        bar.setValue(100)
        self.step3count += 1
        self.roadsChanged = False
        self.dangerChanged = False

    def danger_distance(self):
        """ Calculates distance to nearest 'to'feature from each 'from' feature.
        Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they did
not yet exist """
        import processing
        filename = self.own_dir + '/danger_nearest' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = {'GRASS_MIN_AREA_PARAMETER' : 0.0001,
                      'GRASS_OUTPUT_TYPE_PARAMETER' : 0,
                      'GRASS_REGION_PARAMETER' : None,
                      'GRASS_SNAP_TOLERANCE_PARAMETER' : -1,
                      'GRASS_VECTOR_DSCO' : '',
                      'GRASS_VECTOR_EXPORT_NOCAT' : False,
                      'GRASS_VECTOR_LCO' : '',
```

```
                'column' : ['dngr_dist'],
                'dmax' : -1,
                'dmin' : -1,
                'from' : self.tempHospitalLayer2,
                'from_output' : filename,
                'from_type' : [0,1,3],
                'output' : 'TEMPORARY_OUTPUT',
                'to' : self.dangerLayer,
                'to_column' : None,
                'to_type' : [0,1,3],
                'upload' : [1] }

        processing.run('grass7:v.distance', parameters)


    def water_distance(self):
        """ Calculates distance to nearest 'to'feature from each 'from' feature.
        Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they did
not yet exist """
        import processing
        filename = self.own_dir + '/water_nearest' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = {'GRASS_MIN_AREA_PARAMETER' : 0.0001,
                'GRASS_OUTPUT_TYPE_PARAMETER' : 0,
                'GRASS_REGION_PARAMETER' : None,
                'GRASS_SNAP_TOLERANCE_PARAMETER' : -1,
                'GRASS_VECTOR_DSCO' : '',
                'GRASS_VECTOR_EXPORT_NOCAT' : False,
                'GRASS_VECTOR_LCO' : '',
                'column' : ['water_dist'],
                'dmax' : -1,
                'dmin' : -1,
                'from' : self.tempHospitalLayer2,
                'from_output' : filename,
                'from_type' : [0,1,3],
                'output' : 'TEMPORARY_OUTPUT',
                'to' : self.waterLayer,
                'to_column' : None,
                'to_type' : [0,1,3],
                'upload' : [1] }

        processing.run('grass7:v.distance', parameters)

    def voronoi(self):
        #""" Calculates distance to nearest 'to'feature from each 'from' feature.
        #Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they
did not yet exist """
        import processing
        filename = self.own_dir + '/voronoi' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = {'-a' : False,
                '-l' : False,
```

```
                '-s' : False,
                '-t' : False,
                'GRASS_MIN_AREA_PARAMETER' : 0.0001,
                'GRASS_OUTPUT_TYPE_PARAMETER' : 0,
                'GRASS_REGION_PARAMETER' : self.tempPopulationLayer,
                'GRASS_SNAP_TOLERANCE_PARAMETER' : -1,
                'GRASS_VECTOR_DSCO' : '',
                'GRASS_VECTOR_EXPORT_NOCAT' : False,
                'GRASS_VECTOR_LCO' : '',
                'input' : self.tempHospitalLayer2,
                'output' : filename,
                'smoothness' : 0.25,
                'thin' : -1 }

    processing.run('grass7:v.voronoi', parameters)

def dangerLocations(self):
    """ adds danger_id = 1 to those population features that are covered by danger """
    import processing
    #see what population is covered by danger
    filename = self.own_dir + '/temp_population' + str(self.step3count) + '.shp'
    try:
        os.remove(filename)
    except OSError:
        pass

    parameters = { 'DISCARD_NONMATCHING' : False,
                'INPUT' : self.populationLayer,
                'JOIN' : self.dangerLayer,
                'JOIN_FIELDS' : [],
                'METHOD' : 0,
                'OUTPUT' : filename,
                'PREDICATE' : [0],
                'PREFIX' : 'danger_' }

    processing.run('qgis:joinattributesbylocation', parameters)

    # see which roads are covered by danger
    filename = self.own_dir + '/temp_roads' + str(self.step3count) + '.shp'
    try:
        os.remove(filename)
    except OSError:
        pass

    parameters = { 'DISCARD_NONMATCHING' : False,
                'INPUT' : self.roadsLayer,
                'JOIN' : self.dangerLayer,
                'JOIN_FIELDS' : [],
                'METHOD' : 0,
                'OUTPUT' : filename,
                'PREDICATE' : [0],
                'PREFIX' : 'danger_' }

    processing.run('qgis:joinattributesbylocation', parameters)

def centroids(self):
    """ only works once while running qgis, causes error the second time """
    import processing
    filename = self.own_dir + '/pop_centroids' + str(self.step3count) + '.shp'
    try:
```

```
            os.remove(filename)
        except OSError:
            pass

        parameters = { 'ALL_PARTS' : False,
                    'INPUT' : self.tempPopulationLayer,
                    'OUTPUT' : filename }

        processing.run('native:centroids', parameters)

    def population_distance(self):
        """ Calculates distance to nearest 'to'feature from each 'from' feature.
        Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they did
not yet exist """
        import processing
        filename = self.own_dir + '/population_nearest' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = {'GRASS_MIN_AREA_PARAMETER' : 0.0001,
                    'GRASS_OUTPUT_TYPE_PARAMETER' : 0,
                    'GRASS_REGION_PARAMETER' : None,
                    'GRASS_SNAP_TOLERANCE_PARAMETER' : -1,
                    'GRASS_VECTOR_DSCO' : '',
                    'GRASS_VECTOR_EXPORT_NOCAT' : False,
                    'GRASS_VECTOR_LCO' : '',
                    'column' : ['Host_Pop'],
                    'dmax' : -1,
                    'dmin' : -1,
                    'from' : self.popCentroidsLayer,
                    'from_output' : filename,
                    'from_type' : [0,1,3],
                    'output' : 'TEMPORARY_OUTPUT',
                    'to' : self.waterLayer,
                    'to_column' : None,
                    'to_type' : [0,1,3],
                    'upload' : [1] }

        processing.run('grass7:v.distance', parameters)


    def count_population(self):
        #""" Calculates distance to nearest 'to'feature from each 'from' feature.
        #Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they
did not yet exist """
        import processing
        filename = self.own_dir + '/pop_voronoi' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = {'DISCARD_NONMATCHING' : False,
                    'INPUT' : self.voronoiLayer,
                    'JOIN' : self.popCentroidsLayer,
                    'JOIN_FIELDS' : ['TotPop', 'Host_Pop'],
                    'OUTPUT' : filename,
                    'PREDICATE' : [0],
```

```
                'SUMMARIES' : [5, 6] }
        processing.run('qgis:joinbylocationsummary', parameters)


    def clip(self):
        #""" Calculates distance to nearest 'to'feature from each 'from' feature.
        #Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they
did not yet exist """
        import processing
        filename = self.own_dir + '/pop_voronoi_trimmed' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = {'INPUT' : self.popVoronoiLayer,
                'OUTPUT' : filename,
                'OVERLAY' : self.tempPopulationLayer }

        processing.run('native:clip', parameters)


    def area(self):
        #""" Calculates distance to nearest 'to'feature from each 'from' feature.
        #Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they
did not yet exist """
        import processing
        filename = self.own_dir + '/pop_voronoi_area' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = { 'CALC_METHOD' : 2,
                'INPUT' : self.trimmedVoronoiLayer,
                'OUTPUT' : filename }

        processing.run('qgis:exportaddgeometrycolumns', parameters)


    def centrality(self):
        #""" Calculates distance to nearest 'to'feature from each 'from' feature.
        #Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they
did not yet exist """
        import processing
        filename = self.own_dir + '/centrality_points' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        roads = self.tempRoadsLayer

        parameters = { '-a' : True,
                '-g' : False,
                'GRASS_MIN_AREA_PARAMETER' : 0.0001,
                'GRASS_OUTPUT_TYPE_PARAMETER' : 0,
                'GRASS_REGION_PARAMETER' : None,
                'GRASS_SNAP_TOLERANCE_PARAMETER' : -1,
                'GRASS_VECTOR_DSCO' : '',
```

```
                'GRASS_VECTOR_EXPORT_NOCAT' : False,
                'GRASS_VECTOR_LCO' : '',
                'arc_backward_column' : None,
                'arc_column' : None,
                'betweenness' : 'betw',
                'cats' : '',
                'closeness' : '',
                'degree' : 'degree',
                'eigenvector' : '',
                'error' : 0.1,
                'input' : roads,
                'iterations' : 1000,
                'node_column' : None,
                'output' : filename,
                'points' : self.tempHospitalLayer2,
                'threshold' : 50,
                'where' : '' }


        processing.run('grass7:v.net.centrality', parameters)


    def buffer(self):
        #""" Calculates distance to nearest 'to'feature from each 'from' feature.
        #Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they
did not yet exist """
        import processing
        filename = self.own_dir + '/hospital_centrality_buffer' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = { 'DISSOLVE' : False,
                'DISTANCE' : 200,
                'END_CAP_STYLE' : 0,
                'INPUT' : self.tempHospitalLayer2,
                'JOIN_STYLE' : 0,
                'MITER_LIMIT' : 2,
                'OUTPUT' : filename,
                'SEGMENTS' : 5 }

        processing.run('native:buffer', parameters)


    def average_centrality(self):
        #""" Calculates distance to nearest 'to'feature from each 'from' feature.
        #Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they
did not yet exist """
        import processing
        filename = self.own_dir + '/average_centrality' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = {'DISCARD_NONMATCHING' : False,
                'INPUT' : self.hospitalBuffer,
                'JOIN' : self.centralityLayer,
                'JOIN_FIELDS' : ['degree', 'betw'],
```

```
                'OUTPUT' : filename,
                'PREDICATE' : [0],
                'SUMMARIES' : [6] }
        processing.run('qgis:joinbylocationsummary', parameters)

    def join(self):
        #""" Calculates distance to nearest 'to'feature from each 'from' feature.
        #Creates duplicate layer and adds distance to a defined field and adds fid and cat fields if they
did not yet exist """
        import processing
        filename = self.own_dir + '/danger_pop_dist_area' + str(self.step3count) + '.shp'
        try:
            os.remove(filename)
        except OSError:
            pass

        parameters = { 'DISCARD_NONMATCHING' : False,
                'FIELD' : 'id',
                'FIELDS_TO_COPY' : ['TotPop_sum','Host_Pop_m','area'],
                'FIELD_2' : 'id',
                'INPUT' : self.hospitalDangerLayer,
                'INPUT_2' : self.areaVoronoiLayer,
                'METHOD' : 1,
                'OUTPUT' : filename,
                'PREFIX' : '' }
        processing.run('native:joinattributestable', parameters)

    def toggleLegend(self):
        if self.legend.isVisible():
            self.legend.close()
            self.message('The legend is now hidden', 5)
            self.dlg3.dock.areaButton.setChecked(False)
        else:
            self.legend.show()
            self.message('The legend is now active', 5)
            self.dlg3.dock.areaButton.setChecked(True)

    def togglePopulation(self):
        if
QgsProject.instance().layerTreeRoot().findLayer(self.populationLayer.id()).itemVisibilityChecked():

QgsProject.instance().layerTreeRoot().findLayer(self.populationLayer.id()).setItemVisibilityChecked(F
alse)
            self.message('Population intensity is now hidden', 5)
        else:

QgsProject.instance().layerTreeRoot().findLayer(self.populationLayer.id()).setItemVisibilityChecked(Tr
ue)
            self.message('Population intensity is now shown in green', 5)

    def togglePopulationArea(self):
        if not self.areaVoronoiLayer:
            self.dlg3.dock.areaButton.setChecked(False)
            reply = QMessageBox.question(QMessageBox(), "Warning", "The hospital service area is
available once the table has been updated", QMessageBox.Ok, QMessageBox.Ok)
            if reply == QMessageBox.Ok:
                return
            else:
                return
            return
```

```
        if
QgsProject.instance().layerTreeRoot().findLayer(self.areaVoronoiLayer.id()).itemVisibilityChecked():

QgsProject.instance().layerTreeRoot().findLayer(self.areaVoronoiLayer.id()).setItemVisibilityChecked(
False)
            self.message('Hospital service area is now hidden', 5)
        else:

QgsProject.instance().layerTreeRoot().findLayer(self.areaVoronoiLayer.id()).setItemVisibilityChecked(
True)
            self.message('Hospital service area is now shown in different colors', 5)

    def zoomToFull(self):
        self.iface.mapCanvas().setExtent(self.populationLayer.extent())
        self.iface.mapCanvas().refresh()

        # not yet working video play
#       inputVideo = self.own_dir+'/video.wmv'
#       self.player = QMediaPlayer(None, QMediaPlayer.VideoSurface)
#       self.player.setMedia(QMediaContent(QUrl.fromLocalFile(inputVideo)))
#       video_widget = QVideoWidget()
#
#       self.player.setVideoOutput(video_widget)
#       video_widget.show()
#       self.player.setPosition(0)
#       self.player.play()

    def helpdlg2 (self):
        if self.dlg2.dock.help.isChecked():
            self.dlg2.dock.helptekst.setText("You can press the '...' buttons to select the data
corresponding to the above map objects. Data has been selected if its path is being shown in the
white box. Pressing 'Proceed' will load those map objects which have a valid path in their white box. If
you already have data loaded, only those map objects which have a path name in their white box will
be reloaded. This will replace the current loaded map object.")
        else:
            self.dlg2.dock.helptekst.clear()

    def helpdlg3 (self):
        self.dlghelp.iface.addDockWidget( Qt.BottomDockWidgetArea, self.dlghelp.dock )
        pixmap =
QPixmap(r'C:\Users\Minou\AppData\Roaming\QGIS\QGIS3\profiles\default\python\plugins\save_attrib
utes\symbols\legend.png')
        #label = QLabel()
        #label.setPixmap(pixmap)
        #label = Qlabel(self.dlghelp.dock)
        self.dlghelp.dock.image.setPixmap(pixmap)

    def run(self):
        """Run method that performs all the real work"""

        # Create the dialog with elements (after translation) and keep reference
        # Only create GUI ONCE in callback, so that it will only load when the plugin is started
        if self.first_start == True:
            self.first_start = False
            self.dlg = WelcomeDialog(self.iface)
            self.dlg2 = SelectLayersDialog(self.iface)
            self.dlg3 = AdjustLayersDialog(self.iface)
            self.dlghelp = HelpDialog(self.iface)
            self.legend = Legend()
            #self.dlg4 = SaveAttributesDialog4()
```

```
        #self.dlg.pushButton.clicked.connect(self.select_output_file)
        #self.dlg.pushButton_2.clicked.connect(self.select_input_file)
        #self.dlg.aboutButton.clicked.connect(self.danger_distance)
        #self.dlg.aboutButton.clicked.connect(self.iface.actionVertexToolActiveLayer().trigger)
        #self.dlg.loadButton.clicked.connect(self.load_layers)
        #self.dlg.indicatorRefresh.clicked.connect(self.refresh_indicator)
        self.dlg.dock.proceed.clicked.connect(self.nextStep1)
        self.dlg3.dock.cancel.clicked.connect(self.quitdlg)
        self.dlg2.dock.goBack.clicked.connect(self.backStep1)
        self.dlg2.dock.proceed.clicked.connect(self.nextStep2)
        self.dlg2.dock.cancel.clicked.connect(self.quitdlg2)
        self.dlg2.dock.help.clicked.connect(self.helpdlg2)
        self.dlg3.dock.proceed.clicked.connect(self.nextStep3)
        self.dlg3.dock.goBack.clicked.connect(self.backStep2)
        self.dlg3.dock.cancel.clicked.connect(self.quitdlg3)
        self.dlg3.dock.help.clicked.connect(self.helpdlg3)

        self.dlg2.dock.select_fieldhospitals.clicked.connect(self.select_fieldhospitals_file)
        self.dlg2.dock.select_hospitals.clicked.connect(self.select_hospitals_file)
        self.dlg2.dock.select_roads.clicked.connect(self.select_roads_file)
        self.dlg2.dock.select_water.clicked.connect(self.select_water_file)
        self.dlg2.dock.select_danger.clicked.connect(self.select_danger_file)
        self.dlg2.dock.select_population.clicked.connect(self.select_population_file)

        self.dlg3.dock.legendButton.setChecked(True)
        self.dlg3.dock.goBack.clicked.connect(self.backStep2)
        #self.dlg3.dock.hospitalNone.setChecked(True)
        #self.dlg3.roadsNone.toggled.connect(self.iface.actionPan().trigger())

        #self.dlg3.dock.roadsNone.clicked.connect(self.setNoneRoad)
        self.dlg3.dock.roadsToggle.clicked.connect(self.setToggleRoad)
        self.dlg3.dock.hospitalToggle.clicked.connect(self.setToggleHospital)
        #self.dlg3.dock.hospitalNone.clicked.connect(self.setNoneHospital)
        self.dlg3.dock.hospitalAdd.clicked.connect(self.setAddHospital)
        self.dlg3.dock.hospitalMove.clicked.connect(self.setMoveHospital)
        self.dlg3.dock.hospitalRemove.clicked.connect(self.setRemoveHospital)
        #self.dlg3.dock.territoryNone.clicked.connect(self.setNoneTerritory)
        self.dlg3.dock.territoryRemove.clicked.connect(self.setRemoveTerritory)
        self.dlg3.dock.territoryAdd.clicked.connect(self.setAddTerritory)
        self.dlg3.dock.territoryMove.clicked.connect(self.setMoveTerritory)
        self.dlg3.dock.saveButton.clicked.connect(self.saveButtonClicked)
        self.dlg3.dock.popButton.clicked.connect(self.togglePopulation)
        self.dlg3.dock.areaButton.clicked.connect(self.togglePopulationArea)
        self.dlg3.dock.zoomButton.clicked.connect(self.zoomToFull)
        self.dlg3.dock.legendButton.clicked.connect(self.toggleLegend)

        #self.dlg3.territoryNone.toggled.connect()
        #self.dlg3.hospitalRemove.toggled.connect(self.setDeletePoint)



    # Fetch the currently loaded layers
    #layers = QgsProject.instance().layerTreeRoot().children()
    # Clear the contents of the comboBox from previous runs
    #self.dlg.comboBox.clear()
    # Populate the comboBox with names of all the loaded layers
    #self.dlg.comboBox.addItems([layer.name() for layer in layers])

    # show the dialog
```

```
        # Run the dialog event loop
        #result = self.dlg.exec_()
        # See if OK was pressed

#       if result:
#           # Do something useful here - delete the line containing pass and
#           # substitute with your code.
#           filename = self.dlg.lineEdit.text()
#           with open(filename, 'w') as output_file:
#             selectedLayerIndex = self.dlg.comboBox.currentIndex()
#             selectedLayer = layers[selectedLayerIndex].layer()
#             fieldnames = [field.name() for field in selectedLayer.fields()]
#             # write header
#             line = ','.join(name for name in fieldnames) + '\n'
#             output_file.write(line)
#             # wirte feature attributes
#             for f in selectedLayer.getFeatures():
#               line = ','.join(str(f[name]) for name in fieldnames) + '\n'
#               output_file.write(line)
#           self.iface.messageBar().pushMessage(
#             "Success", "Output file written at " + filename,
#             level=Qgis.Success, duration=3)
```

Appendix D: SDSS Adjustment Tools Python Script

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 17 17:34:24 2019

@author: Thijs
"""
from qgis.core import *
from qgis.gui import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *


#class MapToolMixin:
#    def setLayer(self, layer):
#        self.layer = layer
#
#    def transformCoordinates(self, screenPt):
#        return (self.toMapCoordinates(screenPt), self.toLayerCoordinates(self.layer, screenPt))
#
#    def calcTolerance(self, pos):
#        pt1 = QPoint(pos.x(), pos.y())
#        pt2 = QPoint(pos.x() + 10, pos.y())
#        mapPt1,layerPt1 = self.transformCoordinates(pt1)
#        mapPt2,layerPt2 = self.transformCoordinates(pt2)
#        tolerance = layerPt2.x() - layerPt1.x()
#        return tolerance
#
#
class AddPointTool(QgsMapTool):
    def __init__(self, canvas, layer, onPointAdded):
        QgsMapTool.__init__(self, canvas)
        self.canvas = canvas
        self.setCursor(Qt.CrossCursor)
        self.layer = layer
        self.onPointAdded = onPointAdded
        active_index = self.layer.fields().indexFromName('active')
        self.layer.setDefaultValueDefinition(active_index, QgsDefaultValue('y'))



    def canvasReleaseEvent(self, event):
        ids = []
        for feature in self.layer.getFeatures():
            fid = feature['id']
            ids.append(fid)
        for num in ids:
            QgsMessageLog.logMessage(str(num)+'1', tag="debug" )
        try:
            ids = [ num for num in ids if num.isdigit() ]
        except:
            ids = [ num for num in ids if type(num) == int ]
        if type(ids) == list:
            ids = list(map(int, ids))
            for num in ids:
                QgsMessageLog.logMessage(str(num)+'2', tag="debug" )
        else:
            ids = int(ids)
```

```
            QgsMessageLog.logMessage(str(ids)+'3', tag="debug" )

        try:
            self.new_id = str(max(ids) + 1)
        except:
            self.new_id = str(1)

        self.point = self.toLayerCoordinates(self.layer, event.pos())
        self.feature = QgsFeature(self.layer.fields())
        self.feature.setGeometry(QgsGeometry.fromPointXY(self.point))
        attrs = [None] * len(self.layer.fields())
        idx1 = self.layer.fields().indexFromName("id")
        idx2 = self.layer.fields().indexFromName("active")
        attrs[idx1] = self.new_id
        attrs[idx2] = 'y'
        self.feature.setAttributes(attrs)
        self.layer.addFeature(self.feature)
        self.layer.updateExtents()
        self.layer.updateFields()
        self.layer.triggerRepaint()
        self.canvas.refresh()

        self.onPointAdded()


class MovePointTool(QgsMapToolIdentify):
    def __init__(self, mapCanvas, layer, onPointMoved):
        QgsMapToolIdentify.__init__(self, mapCanvas)
        self.setCursor(Qt.CrossCursor)
        self.dragging = False
        self.feature = None
        self.layer = layer
        self.onPointMoved = onPointMoved


    def canvasPressEvent(self, event):
        found_features = self.identify(event.x(), event.y(), [self.layer], self.TopDownAll)
        if len(found_features) > 0:
            self.dragging = True
            self.feature = found_features[0].mFeature

        else:
            self.dragging = False
            self.feature = None

    def canvasMoveEvent(self, event):
        #self.iface.messageBar().pushMessage(str(self.dragging), duration=3)
        if self.dragging:
            point = self.toLayerCoordinates(self.layer, event.pos())
            geometry = QgsGeometry.fromPointXY(point)
            self.layer.changeGeometry(self.feature.id(), geometry)
            self.canvas().refresh()

    def canvasReleaseEvent(self, event):
        self.dragging = False
        self.feature = None
        self.onPointMoved()


class DeletePointTool(QgsMapToolIdentify):
```

```
    def __init__(self, mapCanvas, layer, onPointDeleted):
        QgsMapToolIdentify.__init__(self, mapCanvas)
        self.setCursor(Qt.CrossCursor)
        self.feature = None
        self.layer = layer
        self.onPointDeleted = onPointDeleted


    def canvasPressEvent(self, event):
        found_features = self.identify(event.x(), event.y(), mode = self.ActiveLayer)
        if len(found_features) > 0:
            self.feature = found_features[0].mFeature
            self.layer.deleteFeatures([self.feature.id()])
            self.layer.triggerRepaint()
            self.canvas().refresh()
            self.onPointDeleted()

        else:
            self.feature = None


    def canvasReleaseEvent(self, event):
#        found_features = self.identify(event.x(), event.y(), [self.layer], self.TopDownAll)
#        if len(found_features) > 0:
#            #if self.feature.id() == found_features[0].mFeature.id():
#            self.layer.dataProvider().deleteFeatures([self.feature.id()])

        pass
        #self.canvas().refresh()


#class ToggleRoadTool(QgsMapToolEmitPoint):
#    def __init__(self, canvas):
#        self.canvas = canvas
#        QgsMapToolEmitPoint.__init__(self, self.canvas)
#
#    def canvasPressEvent( self, e ):
#        point = self.toMapCoordinates(self.canvas.mouseLastXY())
#        print '({:.4f}, {:.4f})'.format(point[0], point[1])

class ToggleTool(QgsMapToolIdentify):
    def __init__(self, mapCanvas, layer, onToggled):
        QgsMapToolIdentify.__init__(self, mapCanvas)
        self.setCursor(Qt.CrossCursor)
        self.feature = None
        self.layer = layer
        self.onToggled = onToggled


    def canvasPressEvent(self, event):
        found_features = self.identify(event.x(), event.y(), mode = self.ActiveLayer)

        if len(found_features) > 0:
            self.feature = found_features[0].mFeature
            fieldIndex = self.feature.fieldNameIndex( 'active' )
            activity = self.feature.attributes()[fieldIndex]

            if activity == 'y':
                self.layer.changeAttributeValue(self.feature.id(), fieldIndex, 'n')
            else:
```

```
            self.layer.changeAttributeValue(self.feature.id(), fieldIndex, 'y')
            self.canvas().refresh()
            self.onToggled()

        else:
            self.feature = None


    def canvasReleaseEvent(self, event):
#         found_features = self.identify(event.x(), event.y(), [self.layer], self.TopDownAll)
#         if len(found_features) > 0:
#             #if self.feature.id() == found_features[0].mFeature.id():
#             self.layer.dataProvider().deleteFeatures([self.feature.id()])
        #self.onPointDeleted()
        pass
        #self.canvas().refresh()


#class DeleteTrackTool(QgsMapTool, MapToolMixin):
#     def __init__(self, canvas, layer, onTrackDeleted):
#         QgsMapTool.__init__(self, canvas)
#         self.onTrackDeleted = onTrackDeleted
#         self.feature = None
#         self.setLayer(layer)
#         self.setCursor(Qt.CrossCursor)
#
#     def canvasPressEvent(self, event):
#         self.feature = self.findFeatureAt(event.pos())
#
#     def canvasReleaseEvent(self, event):
#         feature = self.findFeatureAt(event.pos())
#         if feature != None and feature.id() == self.feature.id():
#             self.layer.deleteFeature(self.feature.id())
#             self.onTrackDeleted()

#Then, back in the forestTrails.py module, add the following to the end of the
#setupMapTools() method:
# self.deleteTrackTool = DeleteTrackTool(
# self.mapCanvas, self.trackLayer, self.onTrackDeleted)
# self.deleteTrackTool.setAction(self.actionDeleteTrack)
#Then replace the dummy deleteTrack() method with the following:
# def deleteTrack(self):
# if self.actionDeleteTrack.isChecked():
# self.mapCanvas.setMapTool(self.deleteTrackTool)
# else:
# self.setPanMode()
#Finally, add a new onTrackDeleted() method to respond when the user deletes
#a track:
# def onTrackDeleted(self):
# self.modified = True
# self.mapCanvas.refresh()
# self.actionDeleteTrack.setChecked(False)
# self.setPanMode()


#class SelectVertexTool(QgsMapTool, MapToolMixin):
#     def __init__(self, canvas, trackLayer, onVertexSelected):
#         QgsMapTool.__init__(self, canvas)
#         self.onVertexSelected = onVertexSelected
#         self.setLayer(trackLayer)
```

```
#       self.setCursor(Qt.CrossCursor)
#
#   def canvasReleaseEvent(self, event):
#       feature = self.findFeatureAt(event.pos())
#       if feature != None:
#           vertex = self.findVertexAt(feature, event.pos())
#       if vertex != None:
#           self.onVertexSelected(feature, vertex)
```

Appendix E: Unimplemented SDSS Feedback

Intuitiveness	Performance	Reasonability	Usability	Stability	Completeness	Guidance
			Auto-update would be nice if it was computed near-immediate.	Difficulty with adding danger.	Visualize hospitals by serving population size	Add readability points in big numbers
			Move and add functions could be combined.	Non-responding when computing	Provide an overall score by weights	
			Toggle which indicators have to be showed		Add road accessibility visualization toggle	
Visualize results as well	Map refresh at every move unnecessary		Pop up save/undo dialog at switch tool attempt	Zoom by pinching not working after a long press		Could be even more clear if it was more structured
Zooming while pinching would make more sense instead of after taking the fingers off the screen	Update roads takes a bit long, but you that probably only once at the beginning of a session.	Served population is quite simplified, but an indication nonetheless.	Streets or districts should be referenced	Average distance to hospital in served area does not make sense in one area	Zoom in/out buttons.	Make guidance fully optional, so you get new map space once you do not need guidance anymore
Map centers to point on the map that is pressed. Should be disabled because people will accidently touch the map.	Different options for hospital service area calculation. From fast to accurate.			Toggles do toggle, but do not register whether the toggle object is active or not	Instead of action toggling, a menu with possible actions when pressing the map	Create a help tool which gives information about the pressed interface element
Danger move not intuitive, need to be able to move the polygon as a whole.			Apply connectivity and resilience to roads instead of junctions	Warning for invalid danger geometry not working.	Undo 1 step button	Stress that SDSS is just a support, not necessarily reality.
Would be nice to have a toggle for whether screen is touchable					Ability to toggle auto table update	Visual demo / tutorial
Add cancel to save/undo button					Toggle connectivity and resilience map	