Deep Sea Mapper

DIVE INTO CALM, DIVE INTO GIS

USER MANUAL

Introduction

Welcome to the user manual for Deep Sea Mapper. This manual is designed to introduce you to Deep Sea Mapper, and will guide you through the methods and procedures for conducting a GIS project. No matter if you are new to GIS, or already an avid user, Deep Sea Mapper can help you feel like you are on vacation while still getting the job done. Let us dive into calm, dive into GIS.

 1. Welcome to *Deep Sea Mapper*

Welcome aboard *Deep Sea Mapper*, your gateway to exploring the depths of geographic information! Designed to combine productivity with the serenity of an underwater escape, this software invites you to dive into GIS tasks while enjoying the feel of a relaxing vacation. Whether you're charting new waters or navigating complex spatial data, *Deep Sea Mapper* makes the journey smooth, intuitive, and even a little fun.

With *Deep Sea Mapper*, you can tackle analysis for your GIS project. The program's underwater-inspired design keeps your workflow calm and engaging, so you can focus on solving problems while feeling like you're drifting through deep blue waters.

So grab your gear, set sail, and let *Deep Sea Mapper* help you chart a course to success. Your GIS adventure begins here!

1.1 What Can You Use *Deep Sea Mapper* for?

Deep Sea Mapper is designed for performing simple operations and analysis on raster data. Whether you're visualizing datasets, running spatial analyses, or processing raster imagery, the program provides the essential tools to get the job done efficiently and effectively.

1.2 What data can be used?

Deep Sea Mapper can read and visualize ASCII files and text files.

1.3 Tasks that can be performed

With *Deep Sea Mapper,* you can import and visualize geographical data as layers, making it easier to explore and understand your raster datasets. You can also perform raster operations on a single layer, or combine multiple layers in order to get new insights from your data.

2. Getting started

This chapter will guide you through everything you need to know before diving into *Deep Sea Mapper*. From system requirements and launching the program to understanding the interface and adding your first layers, this section will provide you with the foundation you need to get started.

2.1 Prerequisites

Before you can run *Deep Sea Mapper*, ensure that the following requirements are met:

1. **Java Development Kit (JDK):**
   You'll need to have Java Development Kit (JDK) 21 installed on your system. *Deep Sea Mapper* is built using this version of Java, so it is essential for running the program
2. **Eclipse IDE:**
   Download and install Eclipse IDE for Enterprise Java and Web Developers. You can get the latest version from eclipse.org.
3. **Importing the Project:**
   After setting up Eclipse, follow these steps to import the program:
   - Create a new project in Eclipse by selecting **File > New > Java Project**
   - Name the project **DeepSeaMapper** and select **Finish**.
   - The project called "DeepSeaMapper" will appear to the left in the package explorer. Right click "DeepSeaMapper" and select **Properties > Libraries > ModulePath** and select **"Add External JARs"**

- Choose the JAR file that DeepSeaMapper.jar where you stored it on your computer.
- Press **Apply and Close.**

Once these prerequisites are in place, you'll be ready to start using *Deep Sea Mapper*.

2.2 Starting the program

- To start Deep Sea Mapper, you open Eclipse and right click your "DeepSeaMapper" in the Project Explorer and choose **Run as > Run with configurations.**
- Make sure the project selected is "DeepSeaMapper", and that the Main class selected is "com.tutorial.project.GUI".
- Press **Run** to start the program.

2.3 Navigating the interface

This section provides an overview of the components of the interface. When you first open the program, you'll see the main workspace divided into several sections.

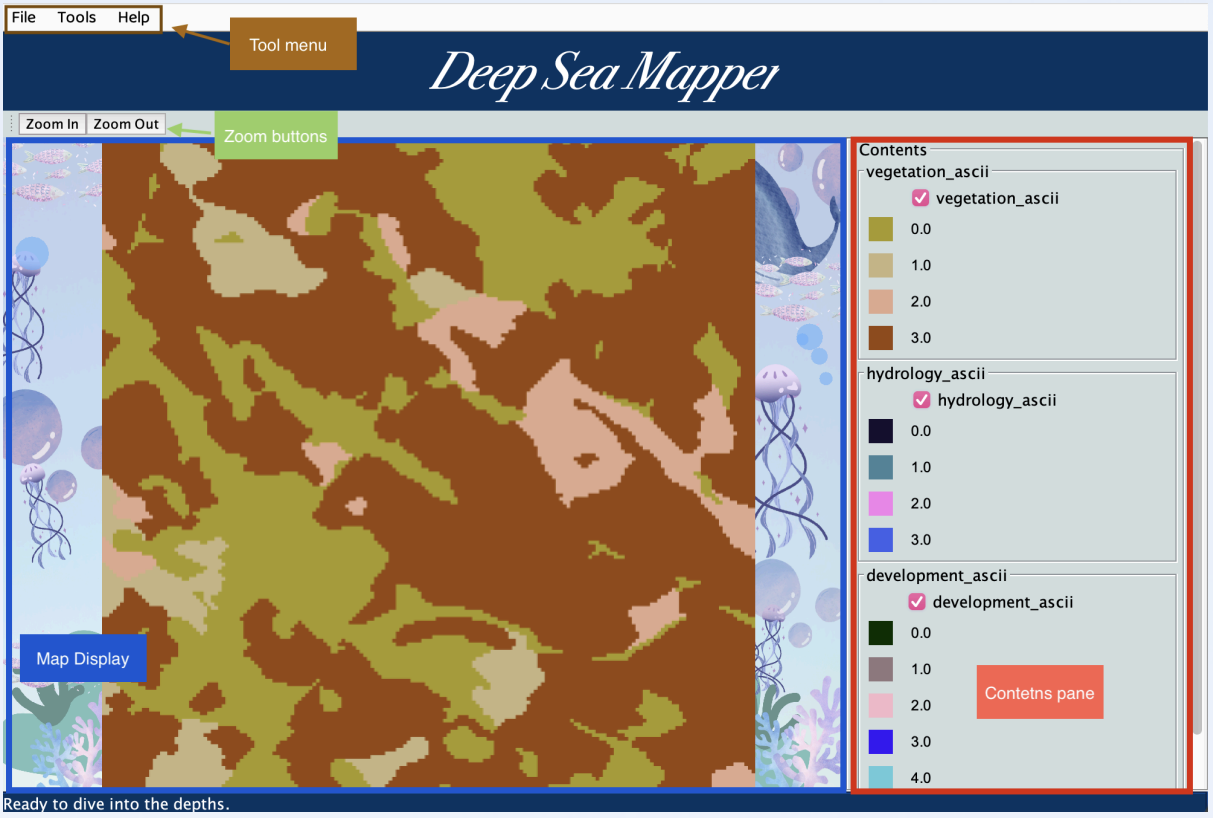
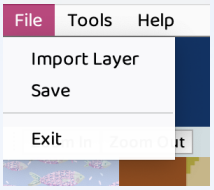


Figure 1: The workspace and its components.

Tool menu



The toolbar helps you operate the software. The **File** button offers the options **Import Layer**, **Save and Exit.** The Import Layer button allows for loading files containing geographic data and displaying them on the map display. The **Save** button saves all visible layers in the map display to text-files. When pressing the **Figure 2:** File **Exit** button, the program closes down.

The **Tool** button offers three types of operations that can be run, Local, Focal and Zonal.

The **Help** button includes the **User Instructions** option, which will lead to this guide in the case it is needed. The About option displayed information of the program, including its version and developers.

Output Window
This window displays imported files as well as the output layers that are results from performed operations.

Contents Pane
The contents pane allows you to manage the visibility of different map layers. Each layer displayed in the map is also displayed in the contents pane with a legend, where each value and its corresponding color in the layer is shown.

Zoom in and **Zoom out** adjust the map's display scale, allowing focus on specific areas, or get a broader view of the layer.

2.4 Adding layers to the map

In order to import a ASCII or text file, you press Import Layer in the Toolbar. The files on the device will be shown and you may select any compatible file and press "Open", see Figure 3. The chosen file will be displayed in the Map display.

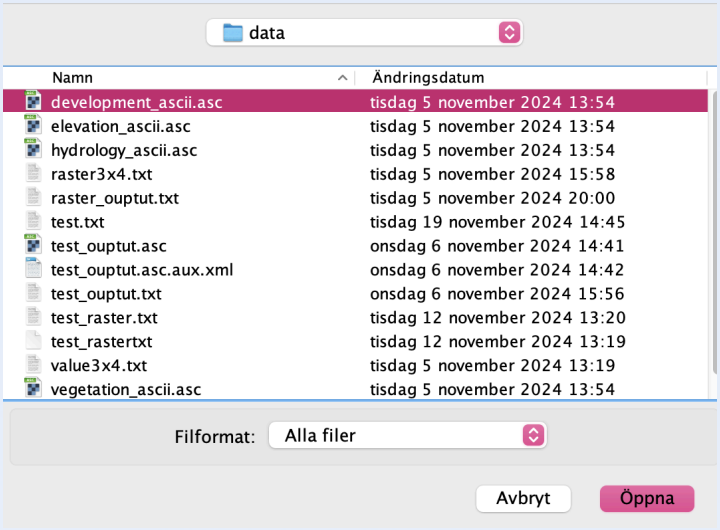


Figure 3: The window for selecting a file to import.

3. Performing operations

In this section, we will explore the basics of the different types of raster operations you can perform with *Deep Sea Mapper*. These include **local operations**, which process individual cells, **focal operations**, which analyze cells based on their neighbors, and **zonal operations**, which work across defined zones.

3.1 Local Operations

Local operations compute a raster output where the output layer's value at each location (cell) is a function of the values associated with the same location of the two input layers.

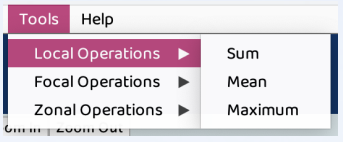


Figure 4: Local operations

- **Local Sum** adds the values of the input layers.
- **Local Mean** calculates the average of the corresponding cells from the two input raster layers. This operation provides a smoothed or generalized view of the data by averaging values across layers.
- **Local Maximum** identifies the maximum value among the corresponding cells in the two input raster layers.

All pseudo codes for the zonal operations can be found in Appendix A.1-3.

3.2 Focal Operations

Focal operations, also known as neighborhood functions, compute a raster output dataset where the output value at each location (cell) is determined by analyzing the values within a defined neighborhood around that cell.

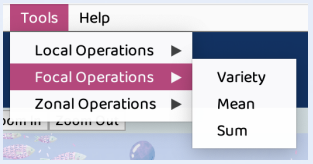


Figure 5: Focal operations

- **Focal Variety** calculates the number of unique values within the defined neighborhood for each cell. It is useful for identifying spatial diversity or heterogeneity within a localized area.
- **Focal Mean** computes the average value of all cell values within the specified neighborhood for each cell.
- Focal Sum calculates the sum of all cell values within the specified neighborhood for each cell.

All pseudo codes for the focal operations can be found in Appendix B.1-3.

3.3 Zonal Operations

Zonal operations compute a raster output dataset where the output value at each location (cell) is determined by summarizing values within a specific zone or region defined by a separate raster. These operations aggregate data based on spatial zones.

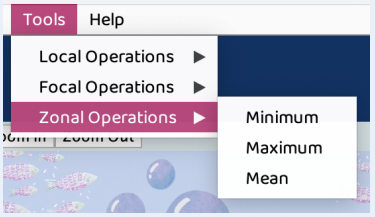


Figure 6: Zonal operations

- **Zonal Minimum** identifies the minimum value among all cell values within each defined zone or region.
- **Zonal Maximum** identifies the maximum value among all cell values within each zone.
- **Zonal Mean** calculates the average value of all cell values within each zone.

All pseudo codes for the zonal operations can be found in Appendix C.1-3.

Appendix

Appendix A - Local Operations

Appendix A.1 - Local Sum

Description: Computes the sum of corresponding cells from two input layers.
Input:
- Layer1: First input layer (grid of values)
- Layer2: Second input layer (grid of values)

Output:
- OutputLayer: A new layer where each cell is the sum of the corresponding cells in Layer1 and Layer2.

Algorithm:
For each cell index i:

```
    If Layer1[i] and Layer2[i] are valid values (not nullValue):
        OutputLayer[i] = Layer1[i] + Layer2[i]
    Else:
        OutputLayer[i] = nullValue
End For
```

Appendix A.2 - Local Mean

Description: Computes the average of corresponding cells from two input layers.
Input:
- Layer1: First input layer
- Layer2: Second input layer

Output:
- OutputLayer: A new layer where each cell is the average of the corresponding cells in Layer1 and Layer2.

Algorithm:

```
For each cell index i:
    If Layer1[i] and Layer2[i] are valid values (not nullValue):
        OutputLayer[i] = (Layer1[i] + Layer2[i]) / 2
    Else:
        OutputLayer[i] = nullValue
End For
```

Appendix A.3 - Local Maximum

Description: Finds the maximum value between corresponding cells from two input layers.
Input:
- Layer1: First input layer
- Layer2: Second input layer

Output:
- OutputLayer: A new layer where each cell contains the maximum of the corresponding cells in Layer1 and Layer2.

Algorithm:

```
For each cell index i:
    If Layer1[i] and Layer2[i] are valid values:
        OutputLayer[i] = max(Layer1[i], Layer2[i])
    Else:
        OutputLayer[i] = nullValue
End For
```

Appendix B - Focal Operations

Appendix B.1 - Focal Variety

Description: Counts the number of unique values in a cell's neighborhood.
Input:
- Layer: Input layer
- radius: Distance from the cell to include neighbors
- isSquare: Boolean indicating if the neighborhood is a square (true) or circular (false)

Output:
- OutputLayer: A new layer where each cell contains the count of unique values in its neighborhood.

Algorithm:

```
For each cell index i:
    Get the neighborhood indices within radius of cell i.
    Initialize a set to store unique values.
    Add Layer[i] (cell's own value) to the set.
    For each neighbor index:
        If neighbor value is valid:
            Add neighbor value to the set.
    End For
    OutputLayer[i] = size of the set (number of unique values)
End For
```

Appendix B.2 - Focal Mean

Description: Computes the mean (average) of a cell's value and the values of its neighbors within a specified radius.
Input:
- Layer: Input layer
- radius: Distance from the cell to include neighbors
- isSquare: Boolean indicating if the neighborhood is a square (true) or circular (false)

Output:
- OutputLayer: A new layer where each cell contains the mean of values in its neighborhood.

Algorithm:

```
For each cell index i:
    Get the neighborhood indices within radius of cell i.
```

```
    Initialize sum = 0 and count = 0.
    For each neighbor index:
        If neighbor value is valid:
            sum += neighbor value
            count += 1
    End For
    If count > 0:
        OutputLayer[i] = sum / count
    Else:
        OutputLayer[i] = nullValue
End For
```

Appendix B.3 - Focal Sum

Description: Computes the sum of a cell's value and the values of its neighbors within a specified radius.

Input:

- Layer: Input layer
- radius: Distance from the cell to include neighbors
- isSquare: Boolean indicating if the neighborhood is a square (true) or circular (false)

Output:

- OutputLayer: A new layer where each cell is the sum of values in its neighborhood.

Algorithm:

```
For each cell index i:
    Get the neighborhood indices within radius of cell i using the isSquare parameter.
    Initialize sum = Layer[i] (cell's own value).
    For each neighbor index:
        If neighbor value is valid:
            sum += neighbor value
    End For
    OutputLayer[i] = sum
End For
```

Appendix C - Zonal Operations

Appendix C.1 - Zonal Minimum

Description: Finds the minimum value in a specified zone and assigns it to all cells in that zone.
Input:
- ValueLayer: Layer containing the values to analyze
- ZoneLayer: Layer defining zones (e.g., regions with the same zone ID)

Output:
- OutputLayer: A new layer where each cell contains the minimum value of its zone.

Algorithm:
Initialize a map to store minimum values for each zone.
For each cell index i in ZoneLayer:
  Get the zone ID from ZoneLayer[i].
  Get the cell value from ValueLayer[i].
  If both values are valid:
    Update the zone's minimum value in the map.
End For
For each cell index i in ZoneLayer:
  Get the zone ID from ZoneLayer[i].
  Assign the zone's minimum value to OutputLayer[i].
End For

Appendix C.2 - Zonal Maximum

Description: Finds the maximum value in a specified zone and assigns it to all cells in that zone.
Input:
- ValueLayer: Layer containing the values to analyze
- ZoneLayer: Layer defining zones

Output:
- OutputLayer: A new layer where each cell contains the maximum value of its zone.

Algorithm:
Initialize a map to store maximum values for each zone.
For each cell index i in ZoneLayer:
  Get the zone ID from ZoneLayer[i].
  Get the cell value from ValueLayer[i].
  If both values are valid:
    Update the zone's maximum value in the map.
End For
For each cell index i in ZoneLayer:
  Get the zone ID from ZoneLayer[i].
  Assign the zone's maximum value to OutputLayer[i].

End For

Appendix C.3 - Zonal Mean

Description: Computes the mean of values in a specified zone and assigns it to all cells in that zone.
Input:
- ValueLayer: Layer containing the values to analyze
- ZoneLayer: Layer defining zones

Output:
- OutputLayer: A new layer where each cell contains the mean value of its zone.

Algorithm:
Initialize a map to store sums and counts for each zone.
For each cell index i in ZoneLayer:
  Get the zone ID from ZoneLayer[i].
  Get the cell value from ValueLayer[i].
  If both values are valid:
    Add the value to the zone's sum and increment its count.
End For
For each cell index i in ZoneLayer:
  Get the zone ID from ZoneLayer[i].
  Calculate the mean for the zone (sum / count).
  Assign the zone's mean value to OutputLayer[i].
End For