Assignment 1: Uninformed Search

Artificial Intelligence
WS 2023
Due: 2023-10-23, 12:00 noon

Katharina Hoedt
Verena Praher
Florian Schmid

1 Environment Setup

First, you will need to setup your programming environment. We use the `conda` environment for this exercise. If the instructions we provide are not enough for you, you can always consult the `conda` documentation[1].

Whenever you see text such as the following:
`$ command param0 param1`

We assume you have access to a sane shell environment (i.e. `bash`) and a nice terminal emulator, in which you can type these commands. The symbol `$` represents your command prompt, and it can look different for your shell.

We will now **download** the `Miniconda` **installer**, and **setup a** `conda` **environment**:

- Navigate a browser to `https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html#regular-installation`, and get the Miniconda distribution for your operating system of choice.
- Follow the rest of the instructions for your operating system to install Miniconda.
- Create a new environment:
  `$ conda create --name py311_ai_assignments python=3.11`
- Activate your virtual environment:
  `$ conda activate py311_ai_assignments`
- For most shells, your command prompt should now have changed, to indicate that the virtual conda environment named `py311_ai_assignments` is now **active**.
- Download the **A1 Framework** from the MOODLE course page.
- Unzip the `ai_assignment1.zip` into an empty directory.
- We will refer to this directory as your **base** directory.
- You should find the following directory structure (only a subset is shown):

[1] `https://docs.conda.io/projects/conda/en/latest/index.html`

```
<this/is/the/path/to/your/base/directory>
|-- ai_assignments
|   |-- ...
|   `-- search
|       |-- __init__.py
|       |-- bfs.py      <-- we will edit this file to solve `Breadth First Search`
|       |-- rs.py       <-- we will edit this file to solve `Random Search`
|       `-- ucs.py      <-- we will edit this file to solve `Uniform Cost Search`
|-- setup.py
|-- generate.py
|-- solve.py  <-- use this script to apply your solvers to problem instances
`-- view.py   <-- use this script to view problem instances and solutions
```

- In your shell, navigate to the **base** directory.
- Issue the following command (including the `.` ← it points to the current directory):
  `$ pip install -e .`
  (it will install the `ai_assignments` package and its dependencies into your active `conda` environment)
- You are now **ready** to tackle the practical part of assignment 1!

2 Theoretical Questions (8 pts)

There is a **separate quiz** on MOODLE for theoretical questions, in which you will encounter questions regarding different search trees (an example is shown in Figure 2.1).

A few hints:

- Successors of a node are added to the fringe from left to right (with the exception of the uniform-cost search algorithm, obviously).
- For UCS: If a node with the same cost alreay exists in the queue, the 'new' node will be inserted **after** the existing one.
- If the order of nodes is asked, please enter the answer in capital letters without spaces, arrows, ... an accepted answer could be **ABC**, but not **A->B->C** or **A B C**.

For the **theoretical questions** you have an **unlimited number** of attempts, **but no feedback** whether or not your answers are correct.

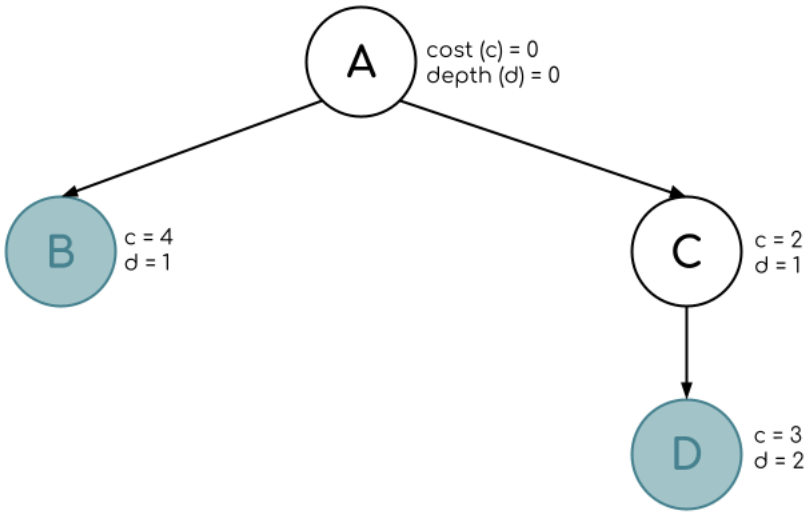

Figure 2.1: ***A search tree.*** *The cost (c) and depth (d) of each node are written next to it. Goal nodes are highlighted in* ***teal****.*

3 Random Search (2 pts)

For the **practical** parts of the assignments, you will be asked to **implement algorithms** that you heard about in the lecture. All practical assignments will have a corresponding MOODLE quiz. Those quizzes contain questions, which ask you to provide the solution to a specific problem instance. The basic procedure is as follows:

- Each **problem instance** is encoded as a **JSON file**.
- This file is **attached** to a question in the MOODLE **quiz**.
- You will need to **download** the problem instance and run your code to **solve** it.
- You will then **copy** the **answer** into the **answer field** of the **question** in the **quiz**.

This part of Assignment 1 is meant to help you become familiar with this procedure. We will now go through all the details and necessary steps to solve this problem:

- Navigate to the MOODLE quiz corresponding to assignment 1 (**A1 Problem Instances**).
- Start the quiz.
- You can **pause and resume any time**.
- You have an **unlimited amount of attempts**.
- Create a new subdirectory named `q1` in the **base** directory.
- Download the problem instance for question 1, and put it into the subdirectory `q1`.
- In your shell, navigate to the **base** directory.
- In your shell, type:
  `$ ls`
  You should see the following output (order of files may differ):
  `ai_assignments  q1 setup.py generate.py solve.py  view.py`
- Make sure your `conda` environment is **active**:
  `$ conda activate py39_ai_assignments`
- Now try the following command:
  `$ python solve.py --help`
  and enjoy the very helpful help text!
- We will try to solve the problem instance, by trying the following command:
  `$ python solve.py q1/board.json rs`
  This says that you want to apply the search algorithm named `rs` to solve the problem instance that is specified in the JSON file `q1/board.json`. You will get output that will look like this:

```
### search method:       rs ############################
nodes expanded 0
path length    0
path cost      0
solution hash  e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

- The output gives you information about the solution that a particular search algorithm has found. The search algorithm name 'rs' has been used to solve the problem instance stored in `q1/board.json`. No nodes were expanded, no path was found (its length is zero). The empty path has no costs. Finally, you find the hash value of the empty path.
- We will now **fix** random search, by **uncommenting a few lines**.
- Open the file `ai_assignments/search/rs.py` in your favourite text editor, and follow the instructions in the comments.
- Try the previous command again:
  `$ python solve.py q1/board.json rs`
- You should get output that is very similar to:

```
### search method:       rs ############################
nodes expanded <some number>
path length    <some different number>
path cost      <some different number again>
solution hash  <a long string of letters and digits>
```

- Yay! Now the random search algorithm does something!
- The **solution path** and the **solution hash** have been written to separate files in the subdirectory `q1`, where the problem instance is located.
- If you now type:
  `$ ls q1`
  You will get output like this (the order of files may differ for you):
  `board.json rs.hash rs.path`
- We can now **view** a visualization of the problem instance and the **solution path**, that was found, by typing:
  `$ python view.py q1/board.json q1/rs.path`
- You should see a GUI window similar to what is shown in Figures 3.1 and 3.2.
- To **answer the question** in the MOODLE quiz, you can either copy the **solution path** from the file `q1/rs.path`, or the **solution hash** from the file `q1/rs.hash` into the textual answer field of the question. In most cases, it will be easier to copy the hash, as the path will be very long.
- The question will tell you, whether your path/hash is correct or not. Again, you have **unlimited attempts** to answer the question.

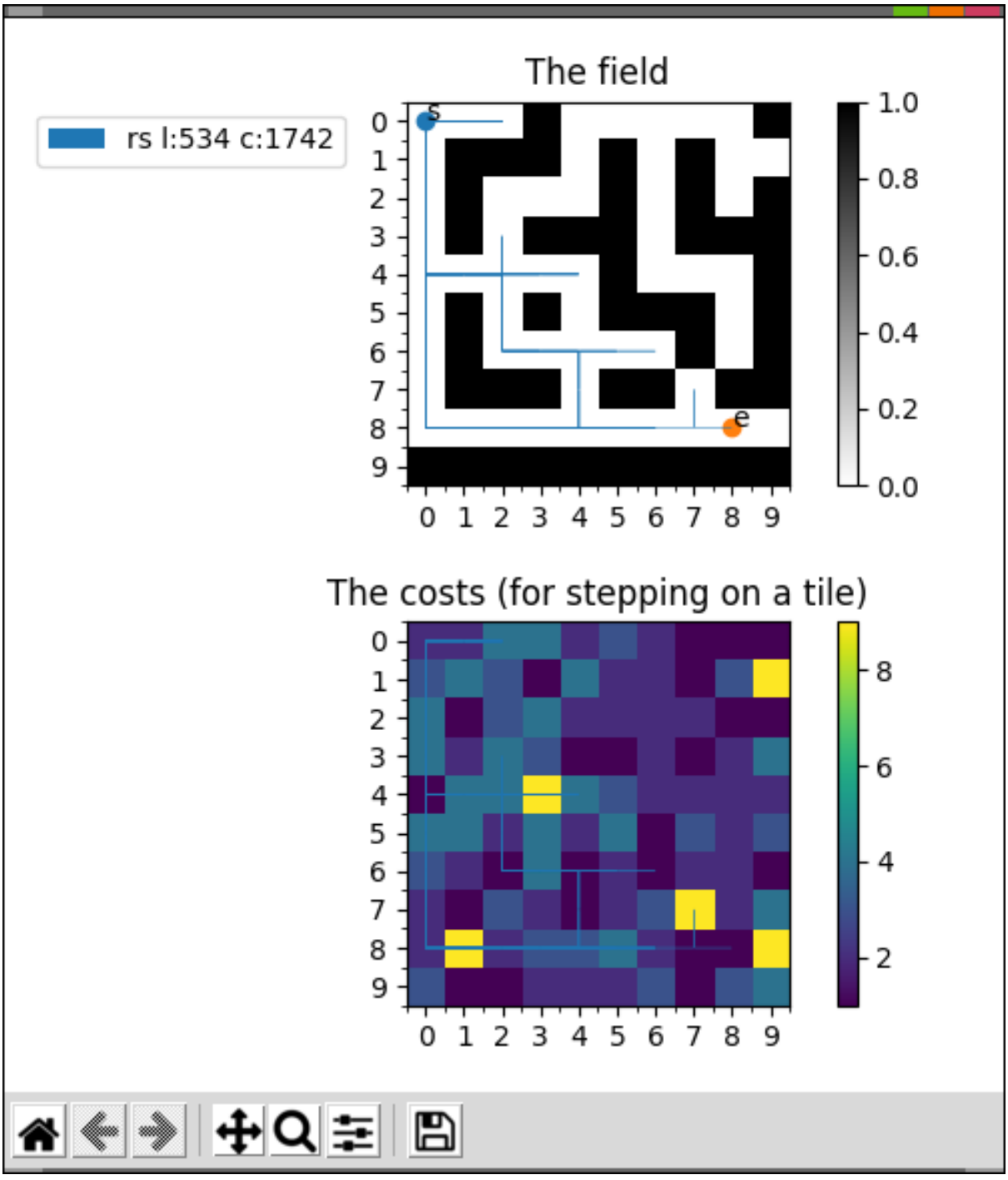

Figure 3.1: *The view you should get.*

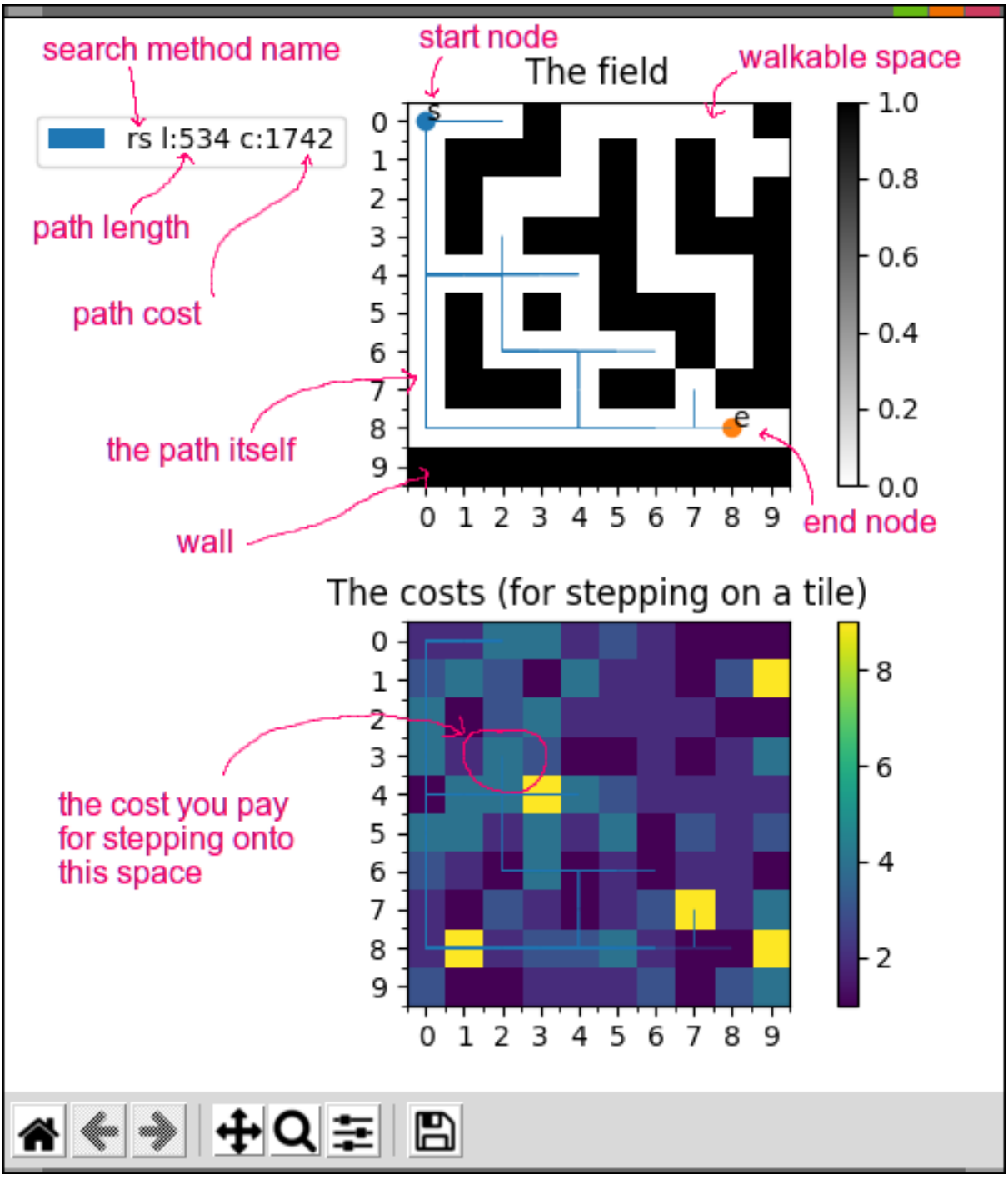

Figure 3.2: *The view you should get, annotated.*

4 Breadth First Search, Uniform Cost Search (4 + 10 pts)

After solving the introductory problem, you will be asked to implement the **Breadth First Search** and the **Uniform Cost Search** algorithm that you heard about in the lecture. The mode should be familiar from implementing **Random Search**.

Here is what to do for BFS:

- Download the BFS problem instance from MOODLE
- Implement Breadth First Search in `ai_assignments/search/bfs.py`
- In a shell, try:
  `$ python solve.py <the-problem-instance-from-the-quiz> bfs`
- Copy and paste the solution hash to the respective BFS question in the quiz.

Here is what to do for UCS:

- Download the `UCS` problem instance from MOODLE
- Implement Uniform Cost Search in `ai_assignments/search/ucs.py`
- In a shell, try:
  `$ python solve.py <the-problem-instance-from-the-quiz> ucs`
- Copy and paste the solution hash to the respective UCS question in the quiz.

Submitting your implementations

Upload your BFS and UCS implementation as a zipped file in MOODLE (this is only possible if you submitted the solution hashes)!

5 Toy environments, and their connection to the "real world"

For educational and testing purposes, you can also **generate** problem instances, and **study** the **behavior** of your **search algorithms**. **None** of the following is **necessary**, but might be **interesting** to you! The (tentative) connection to the real world is explained in the captions for Figures 5.1, 5.2 and 5.3.

- At the command prompt type:
  `$ python generate.py --help`
  and read the very helpful help text!
- We will now **generate** a new problem instance of the type "maze", a field size of 31×31, and store it in a JSON file `test/board.json`:
  `$ python generate.py maze 31 test/board.json`
- You can of course **view** the newly generated problem instance:
  `$ python view.py test/board.json`
- You can get a different random instance, if you change the **seed** of pseudo-random number generator:
  `$ python generate.py maze 31 test/board.json --seed 101`
- You can also select **different generators**:
  `$ python generate.py terrain 31 test/board.json`
  `$ python generate.py rooms 71 test/board.json`

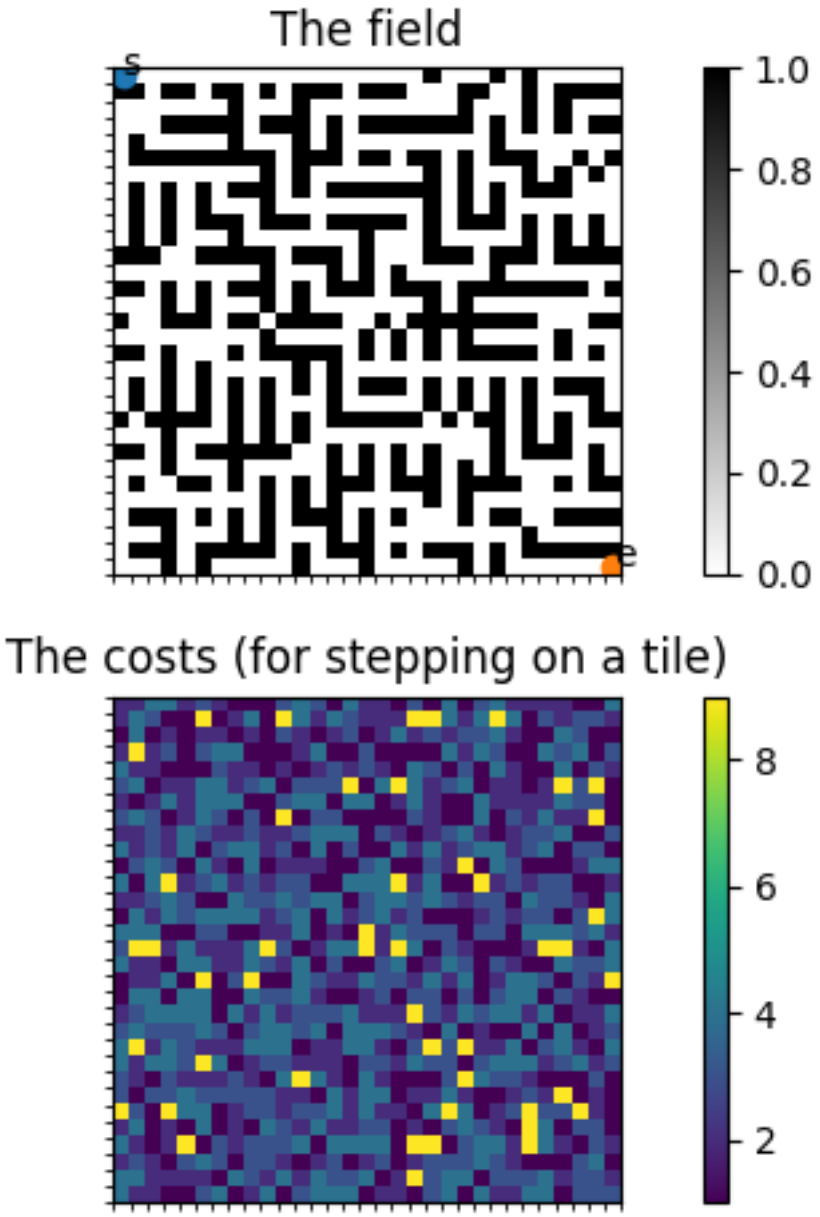

Figure 5.1: *This is an instance generated by the **maze** generator. Think of it as a 2D level in an adventure game, where your player character (controlled by your search algorithm) has to escape the maze without too much damage. The field in the above picture is a **representation** of the **state space**. Everything that is colored in white is walkable space. In this case, a position is equivalent to a state, and is encoded as a tuple (x, y). The generator algorithm proceeds as follows: after a perfect maze is generated, random walls are replaced with spaces, and the cost for stepping in these new spaces is set to a high value. They are the yellow pixels in the lower image. This is supposed to represent dangerous **traps**, that will damage your player character.*

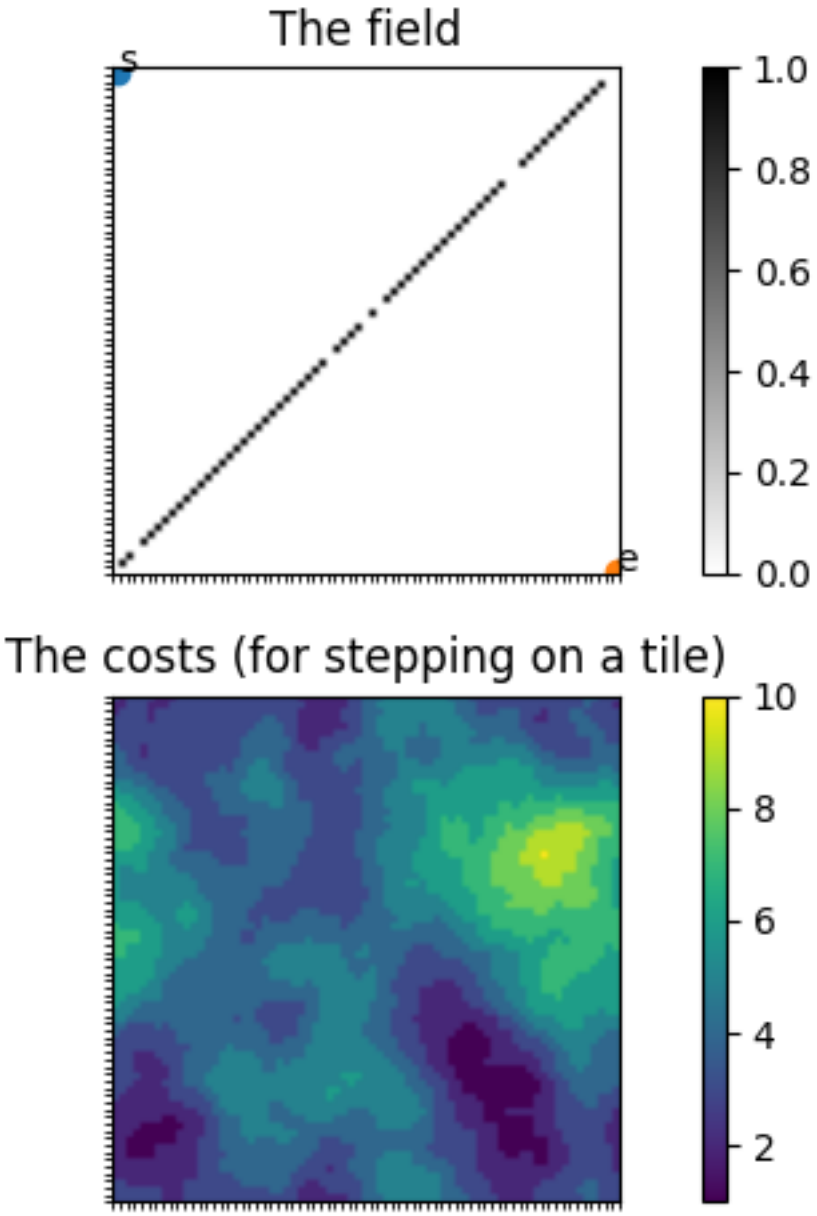

Figure 5.2: *This is an instance generated by the **terrain** generator. Think of it as a standard elevation map that shows geographical features. The costs **encode** these **geographical features**, which you can imagine **representing height** for example, or, more generally, how easy it is to travel across a particular patch of terrain. To complicate matters further, an ancient civilization built a very long and high wall directly across the terrain. As time passed, some holes formed in the wall. Your search algorithms should **plan a path** for a small vehicle, that is supposed to get from one, to the other side of the wall. Different search algorithms will find different routes through the terrain. The generator algorithm generates terrain by layering multiple samples of band limited noise.*

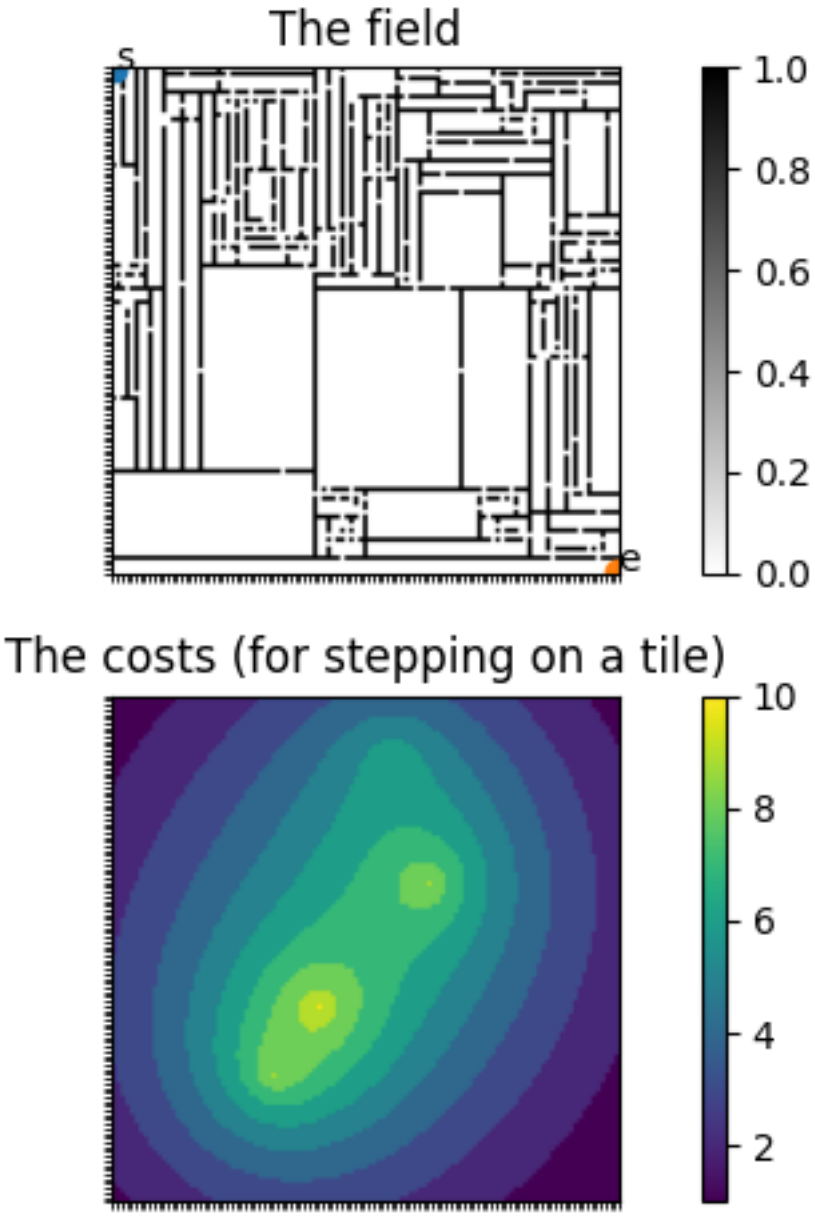

Figure 5.3: *This is an instance generated by the **rooms** generator. Think of it as a 2D floor plan, a map of the interior of a building. There are lots of rooms of different sizes, with lots of doors. Imagine that this is the floor plan of a **nuclear reactor** that had a small **meltdown**. The **costs** represent **radiation strength** measurements, and the task is to navigate a small robotic vehicle through the ruins, **without** the robot being **irradiated** too much. Using cost sensitive search algorithms will make the most sense in this scenario. The generator is a basic recursive subdivision algorithm, and the "radiation" is determined by picking random locations on the field, and computing the radiation intensity per space $\propto \sum_i 1/d_i^2$, with d_i being the Euclidean distance to the random location (x_i, y_i).*