NPGR035 Project - Maze run page reader

Mgr. Matúš Goliaš, Doc. RNDr. Elena Šikudová PhD.

Assigned: 13. 12. 2022

Deadline: 12. 2. 2023

Backstory

A week ago, you learned about a large diamond hidden in a maze and decided to retrieve it for your collection. You searched through the maze for days but had not found anything. With endless corridors, cobwebs and cobblestones, with no respite in sight, you pressed onward and finally discovered something. There was a small, talkative robot, and you agreed that he would search for the diamond and tell you where it was. The robot zipped away with incredible speed, leaving you for a much-needed rest.

It took several hours, but eventually, the robot returned. You beamed at the sight, but your joy was short-lived as you noticed the robot wobbling and stumbling into walls until it collapsed at your feet. With its last breath (let's not question breathing robots), it told you that it not only got drunk on some leftover engine oil right after it left you, but it had been unable to remember the correct path. Moreover, the robot's direction notes got shuffled as it had fallen over before returning to you. With a heavy sigh, you kicked away the unconscious robot and started deciphering its notes.

Description

The instructions are written in separate lines on several pages. Each page has a number describing the original order of the lists. Unfortunately, the pages are shuffled and have to be reordered before we can follow the instructions. Not only that, the drunken robot's steps are unclear since it decided to be very eloquent when writing them down. Thus, we have to decipher the meaning of its instructions as well.

Figure 1 shows examples of paths within their mazes. Both mazes and paths are randomly generated and consist of steps with instructions to move or turn in some direction.

Further, Figure 2 shows an example of the instruction pages. They are greyscale images filled with letters and numbers from the EMNIST dataset

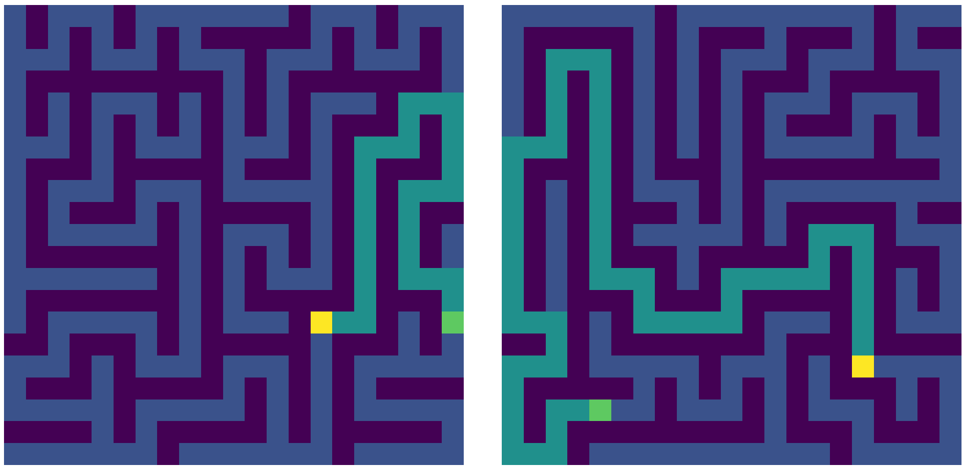
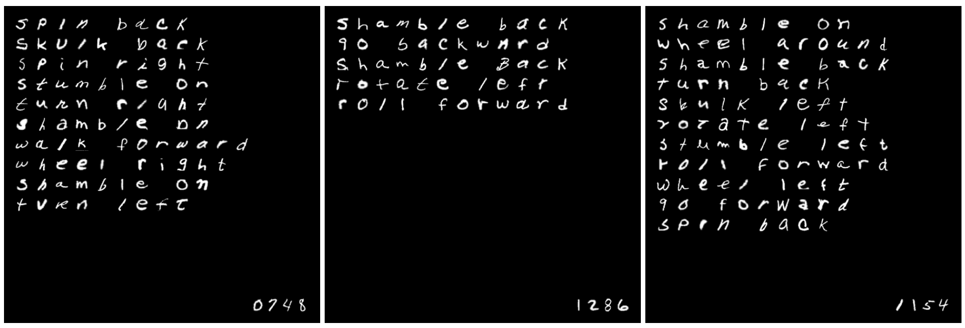


Figure 1: Two examples of mazes and the described path within. Green point shows the beginning of the path and yellow marks the end.



spin back
skulk back
spin right
stumble on
turn right
shamble on
walk forward
wheel right
shamble on
turn left
0748

shamble back
go backward
shamble back
rotate left
roll forward
1286

shamble on
wheel around
shamble back
turn back
skulk left
rotate left
stumble left
roll forward
wheel left
go forward
spin back
1154

Figure 2: Instruction pages for the maze run shown in Figure 1 (Left).

[1]. Each page contains several instructions written in separate lines with no empty lines between them. Further, each page has a number in the lower right corner marking the true order of the pages. The number is always composed of four digits with leading zeros in case of a lower number. There is always some space between the lowermost instruction and the page number.

Similarly, individual characters are separated by spaces, so they never intersect, and the space between letters is effectively a blank character. Together with this text, we provide a training dataset for letters and digits generated from EMNIST and maze runs created either from the training dataset or a validation set, which is not provided. The original EMNIST data can be downloaded from the webpage of the authors [2]. Classifiers should be trained only on the provided data, even though it is only a fraction of the entire dataset. The required performance is adjusted accordingly.

The assignment files

The assignment contains several files which may (or must) be used in the implementation. Here is the complete list with their short descriptions.

- **Source files** Template for the solution.
  - `utils.py` Helper file with EMNIST data loading class and valid instructions.
  - `page_reader.py` Source file for the solution.
  - `evaluator.py` Evaluation script which computes accuracy of the solution on the maze run tasks.
- **Training data** Data intended for classifier training.
  - `emnist_small_letters_train.npz` Dataset consisting of 1000 greyscale images for each small letter. All images are 28x28 pixels.
  - `emnist_digits_selection_train.npz` Dataset consisting of 4000 greyscale images for each digit. All images are 28x28 pixels.
- **Evaluation data** Maze run files intended for evaluation of the solution.
  - `python_train` Set of files generated from the training dataset.
  - `python_validation` Set of files generated from a separate dataset which is not provided.
- `emnist-byclass-mapping.txt` Text file with a mapping between character labels and their ASCII code.
- **Assignment text** This file is the assignment text.

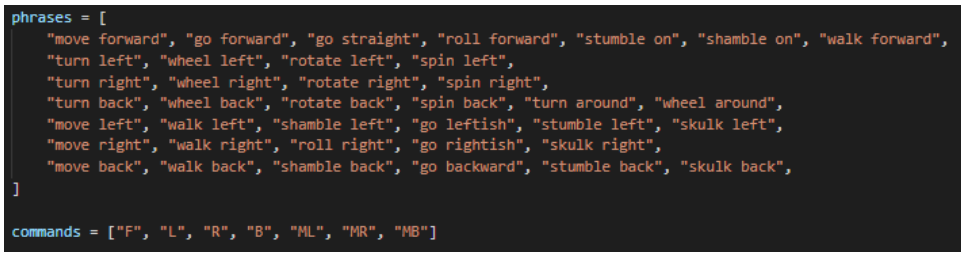
We have worked with MNIST files extensively throughout the semester, so the format should be familiar. All solutions will be evaluated on maze runs generated from yet another dataset, so make sure to use validation files only for testing.

Implementation requirements

The solution must classify the individual characters, including the space between words (there is always only one). These characters have to be ordered as in the written text. That means the solution has to return a list of strings representing lines from top to bottom for each page, including the page number. Further, the solution must match all detected lines (except for the number) with

valid phrases and return a list of detected phrases for each page. Finally, the solution has to be able to order the pages according to their number and return the instructions belonging to the detected phrases in a single list for the entire maze run. The text and detected phrases have to be returned in the given order of pages, and the final instructions have to be in the numbered order of the pages. Most importantly, all solutions must run without issues through the provided evaluator, which will compute the accuracy. The template source files contain examples of how to run the evaluation.

We require the implementation of a complete solution to the problem resulting in a list of movement instructions extracted from the pages. The solution has to be able to read individual lines from the pages, match them to the valid phrases, reorder the pages themselves and report the intermediate results.



```
phrases = [
    "move forward", "go forward", "go straight", "roll forward", "stumble on", "shamble on", "walk forward",
    "turn left", "wheel left", "rotate left", "spin left",
    "turn right", "wheel right", "rotate right", "spin right",
    "turn back", "wheel back", "rotate back", "spin back", "turn around", "wheel around",
    "move left", "walk left", "shamble left", "go leftish", "stumble left", "skulk left",
    "move right", "walk right", "roll right", "go rightish", "skulk right",
    "move back", "walk back", "shamble back", "go backward", "stumble back", "skulk back",
]

commands = ["F", "L", "R", "B", "ML", "MR", "MB"]
```

Figure 3: The set of valid phrases and instructions they represent. Each line in the phrase definition corresponds to one instruction (command). This image is taken from the helper file.

The valid phrases which can appear written on the pages, as well as the represented instructions called commands, are shown in Figure 3. The helper file `utils.py` contains a mapping between phrases and commands.

We do not require computer vision knowledge from students of NPGR035, so it is not necessary to know how to split the image into individual characters. It is possible, for instance, with `skimage.measure.label` on a binarised image. Then, `np.mgrid` with character masks and `np.max/min` can be used to find the bounding boxes of the characters. Two additional problems are splitting the text into lines and disconnected letters (e.g. i). You should be able to come up with a way of dealing with these problems.

Next, matching detected words with valid phrases does not require classification, and even straightforward methods will reach extremely high accuracy. Therefore, it is not necessary to train classifiers for phrase recognition.

You can use the libraries we worked with on practicals: numpy, scipy, matplotlib, scikit-learn, scikit-image, tensorflow. Any other library has to have a detailed explanation describing why it is needed. Your solution will be tested in a virtual python environment (version 3.10) with a fresh installation of the libraries, so make sure to write the required python or library versions if you suspect a compatibility issue.

Saving and loading of models

Experimentation will probably be much easier if you save your models after training and remove the need to train them every time you want to run the evaluation. We have not talked about this feature on practicals so here is an example of how you can do it. Scikit-learn and TensorFlow have slightly different model-saving methods, so keep that in mind. Also, using pickle to save and load models between different package versions is not supported. Therefore, bear in mind that we will run the training routine of your solution during evaluation and we will not load your submitted model files. Training should not take longer than 15 minutes and it should use the best parameters (no GridSearchCV or manual cross-validation). Nevertheless, submit source files with your original model selection and training together with the final solution.

```
# Save scikit-learn model.
import lzma
import pickle
fileName = "myModel"
with lzma.open(fileName, "wb") as model_file:
        pickle.dump(trainedModel, model_file)

# Load scikit-learn model.
with lzma.open(fileName, "rb") as modelFile:
        trainedModel = pickle.load(modelFile)

# Save tensorflow model (h5 extension is important).
name = "myModel.h5"
trainedModel.save(name)

# Load tensorflow model.
trainedModel = tf.keras.models.load_model(name)
```

Performance requirements

The required performance is given in the following list. The accuracies are computed as an average through the entire maze run set. Your solution must not perform worse on the validation set given to you, and the performance on our test set will be judged accordingly. Final path accuracy (the fourth item) is a soft requirement (your solution will not be rejected if it is lower) since it strongly depends on page number recognition.

- **Character accuracy** At least 85%.
- **Phrase accuracy** At least 95%.
- **Page number accuracy** At least 90%.
- **Final path accuracy** At least 90%.

You should strive for the best possible result using the tools and approaches we discussed over the semester. Therefore, train only on the training set and use the validation files only to verify your approach. If you have any questions or problems, then feel free to contact us (Mgr. Matúš Goliaš, Doc. RNDr. Elena Šikudová PhD.) through email or discord.

Matlab differences

This file is written based on the Python assignment of the project. The Matlab version of the assignment is almost equivalent. The implementation differences are noted in the `.m` file next to this pdf. They mainly concern character extraction and model saving.

References

[1] Cohen, G., Afshar, S., Tapson, J., van Schaik, A., 2017. Emnist: an extension of mnist to handwritten letters. URL: `https://arxiv.org/abs/1702.05373`, doi:`10.48550/ARXIV.1702.05373`.

[2] Cohen, G., Afshar, S., Tapson, J., van Schaik, A., 2019. Emnist: an extension of mnist to handwritten letters. URL: `https://www.nist.gov/itl/products-and-services/emnist-dataset`.