

# CIS6020 Artificial Intelligence

## Assignment 2

Enshen Zhu

1194726

### I. Part 1

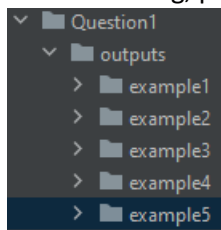
#### 1. Acknowledge

- a) The concept ideas and general object-oriented structure are derived from the following webpages
- <https://towardsdatascience.com/solving-nonograms-with-120-lines-of-code-a7c6e0f627e4>
  - <https://github.com/topics/nonogram-solver>
- b) Due to the technical difficulties and time limitations, the following functions inside the nonogram.py are referred from the <https://towardsdatascience.com/solving-nonograms-with-120-lines-of-code-a7c6e0f627e4>

```
• display_board()
• save_board()
• check_done
• should_terminate
```

#### 2. How To Run

- a) Before starting, please create the folder structure as follow:



(Add an output folder in the same directory where the nonogram.py is located, and add five folders: example1, example2, etc., inside the outputs folder.)

- b) Please refer to the <https://www.jetbrains.com/help/pycharm/managing-dependencies.html#create-requirements> and install the dependency libraries from the *requirement.txt*

- c) At the director where nonogram.py is located, open the terminal and enter **python nonogram.py <example\_name>**. For example, if we need to run the puzzle solving for example 1, we should enter **python nonogram.py example1**

### 3. Solving Strategy

The strategy for solving the Nonogram can be split into three steps:

a) Step 1:

At the beginning of the Nonogram solving, we need to find out every possible option to fill out the rows and columns with black squares.

b) Step 2:

Explicitly fill out the cells that only have one possible option.

|  |   |   |   |  |
|--|---|---|---|--|
|  | √ | √ | √ |  |
|--|---|---|---|--|

Label: [4]

For example, as it is shown above, if the drawing board has the size of 5\*5 and one of the rows is labeled as [4], it means that this row is filled in black with either the first 4 cells or the last 4 cells. However, no matter in which case, the 3 cells in the middle are determinately needed to fill in black. (Check marks represent the cells to be filled in black)

c) Step 3

Based on the filling result from step 2, we may remove some options which do not meet the criteria.

|  |  |  |   |  |  |
|--|--|--|---|--|--|
|  |  |  | X |  |  |
|--|--|--|---|--|--|

Label: [3]

For example, as it is shown above, for a single row with 6 cells labeled as [3], the original possible options may be [0,1,2], [1,2,3], [2,3,4], or [3,4,5]. However, if the result from step 2 indicates that cell 4 (indexed as 3 if starting from 0), the only possible option for this row will be [0,1,2] and shown as follow (check marks represent the cells to be filled in black)

|   |   |   |   |  |  |
|---|---|---|---|--|--|
| √ | √ | √ | X |  |  |
|---|---|---|---|--|--|

Label: [3]

## 4. Results

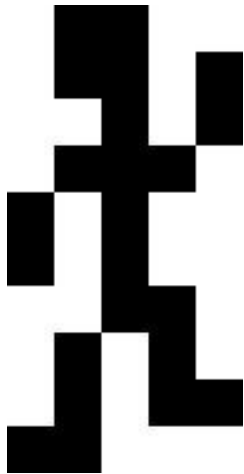
1. Here are the following outputs images from example 1 to example 4



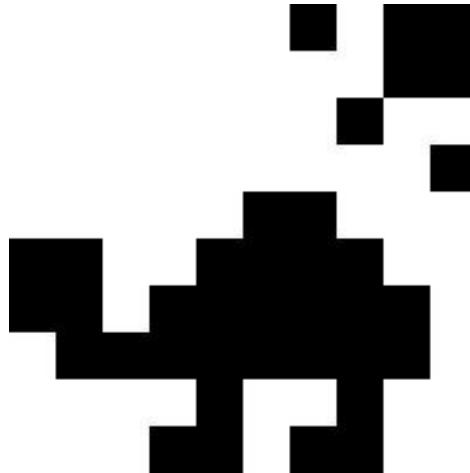
Example1



Example2



Example3

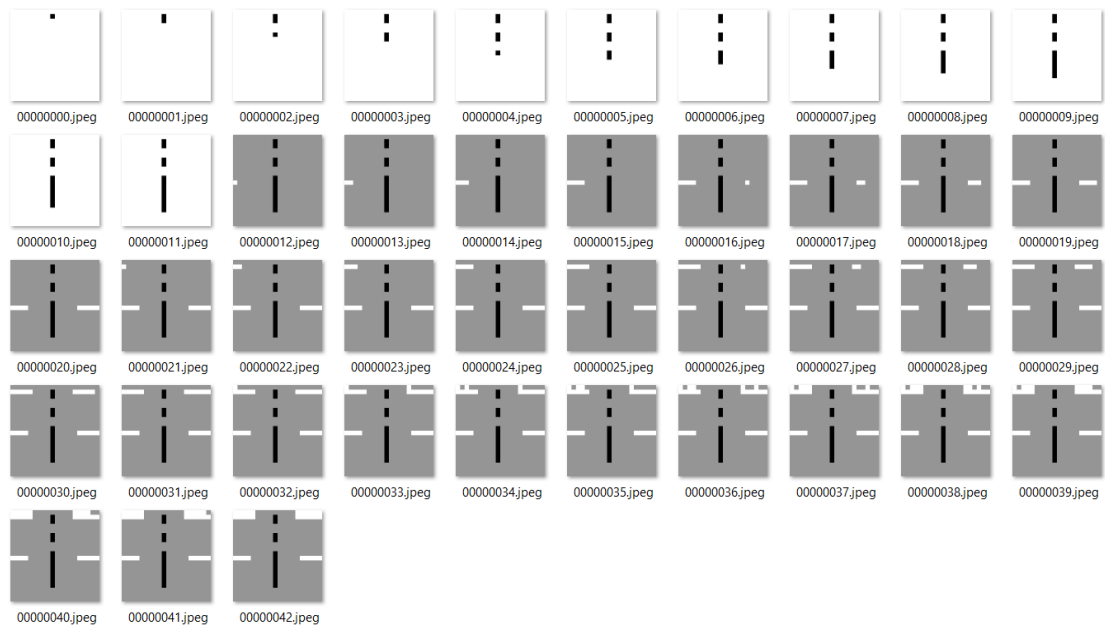


Example4

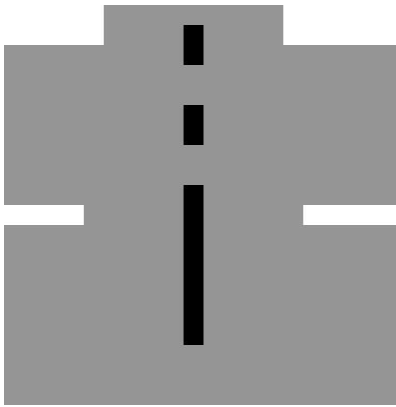
Besides, the processing times from sample1 to sample 4 are less than 0.4 seconds. Here is the photo snipping of sample 4

```
winni@Frontier MINGW64 /d/Guelph_Master/CIS6020 AI/CIS6020_AI_Assignment/Assignment2/Question1 (main)
$ python nonogram.py example4
This example takes 0.3064 second to be finished.
```

- However, the script cannot solve example 5, and it encounters infinite looping in the 43rd episode. The following two images show **every single episode's result** and the **43rd episode's result**.



Episode 0 – Episode 43



Episode 43

It is believed that when it comes to the 43<sup>rd</sup> episode, the algorithm finds multiple next-step solutions that all meet the criteria. Therefore, it may not be able to determine which cell it should paint in black or white.

To solve this issue, the algorithm may need to add an extra function to make a trial by arbitrarily painting a cell that meets the row-column criteria and exploring if the algorithm could converge at the end.

## II. Part 2

### 1. How To Run

- a) In the directory where nonogram.py is located, open the terminal and enter **python makenonogram.py <image\_name>**. For example, if we need to run the text puzzle generation for original image 1, we should enter **python makenonogram.py originalimage1**
- b) The original image, the puzzle image, and the puzzle text are located inside the “ForSubmission” directory. However, the script does not use this directory for operation and manipulation. Instead, it uses the “property” directory for image processing. Please do not make any manual changes to this directory.
- c) There are two original images; one is a seaman portal avatar, and another is a jet fighter

### 2. Solving Strategy

- a) This script is going to do the following steps to turn the original color image into the image puzzle and the text puzzle

- Degrade the original image into the scale of 100 \* 100 pixels down-sampled image
- Turn the down-sampled image into the greyscale pixel map and use the threshold method to convert it into the black and white puzzle. Specifically, if a specific pixel's greyscale is below 128, it should be regarded as a black pixel. In contrast, if the greyscale is over 128, it should be considered as a white(blank) pixel.
- Convert the puzzle into a binary pandas data frame, by which "1" means filling in the black and "0" means leaving it blank.
- Output the text puzzle in the format of a text file, which reports the row and the column profile.

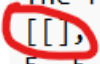
### 3. Results

- a) The followings are the original images and the puzzling images. The text puzzle can be found inside the "ForSubmission" folder.



- b) In the text puzzle text file, the row and the column profile are represented in a two-dimension array. A sample is shown as follows). You may find that some of the second-dimension arrays are left blank, which means that this specific row or column should not be filled as anything.

The row profile is:

 `[[], [1], [1, 1], [1, 1], [1,`  
`5, 5, 32, 3], [6, 8, 5, 2, 1,`

The column profile is:

`[30, 9, 5, 2], [26, 5, 5, 2],`