*Futoshiki Documentation*

**Libraries:**

* Library 1
  + Purpose
  + Use
  + Link to online info
* Library 2
  + Purpose
  + Use
  + Linke to online info

**Data storage:**

* Image representing how matrix is ordered for logic file
* How numbers are stored

**Read in method:**

* Describe how method works
* Mention libraries used
* Mention why it is needed

**Print board method:**

* Same as before

**Other methods:**

**How methods are put together:**

For this project, the first step we did was format our 3x3, 4x4, and 5x5 puzzles into text files. Each puzzle would have its own, separate three text files; consisting of the numbers, logic, and solution. This was an essential step, so that we were able to be consistent with our classmates in how we formatted the puzzles. Once this was complete, we had to develop a “Read In” function that allowed us to import the puzzle onto python. This “Read In” function, essentially, would ask the user to input a desired puzzle size that it wanted to be solved. The code would automatically take the inputted response by the user and pull in all three corresponding text files that were already made. This way, we had everything we needed in the code to solve the desired futoshiki puzzle. Another thing that we did in the “Read In” function was convert the number, logic, and solution text files into a 2D array.

The next thing that we did was create the initial population with random values. We chose the population size to be 10. So, it will generate 10 boards, with the desired size, that have random numbers in each location. Once the boards are made, we will run each of them through a fitness function and assign a “reward” to each board. Since the solution to the board is already programmed in, we will use that as a basis to determine which offspring we choose as the parents. We set up a reward based system that gives each board a reward value based on how close it is to the actual solution. Then we filter through the boards and determine which ones are the most fit. We decided to use a roulette wheel selection to produce two offspring from each parent. The offspring will obtain their numbers from the more fit parents. Once we have these offspring created, we choose to use mutation, or just randomly generated numbers, to fix the incorrect inputs. We then make that process repeat until it eventually solves the puzzle.

The code will stop running whenever the puzzle has been solved or \_\_\_\_. To determine how to stop the code whenever we get a correct puzzle, we chose to make a max score that is based on the solution to the puzzle. As the population continues to grow and mutate, the reward system should continue to improve and each puzzle should get a higher and higher score. This will continue until a puzzle eventually gets the reward value that is equal to the max score. This is how we will know whenever a correct puzzle has been obtained.

-include your approach (genetic algorithm, PSO)

- key parameters (population size, mutation rate)

- issues we had

- most difficult puzzle solved