URL:https://github.com/ZHHHHHHAO/hw4-zhh-

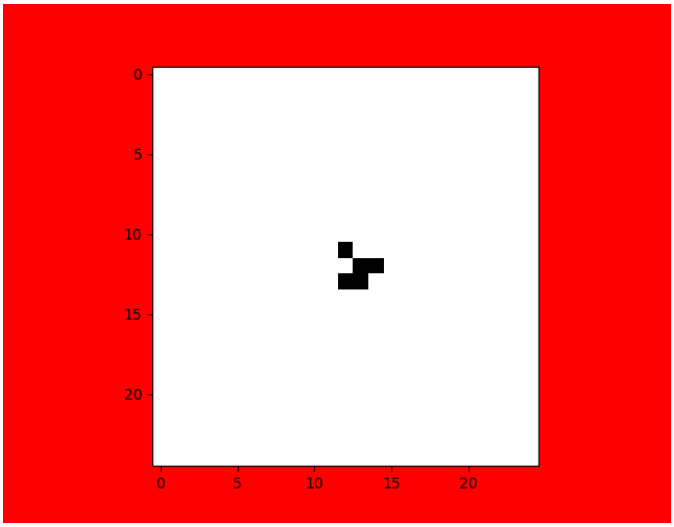
Here's a link to the Chat GPT record I used when I did the first question

https://chat.openai.com/share/a26ab35f-1e24-41ca-b645-d07540103588

Here's a link to the Chat GPT record I used when I did the second question
https://chat.openai.com/share/e4eb850e-5203-4668-8f04-0da82d45a1d1

Task 1
Define matrix A: You start by defining a matrix A that represents a connectivity matrix of a directed graph. Each column of A corresponds to a node, and each row represents the outgoing links from a node to other nodes.
Normalize columns of A: To ensure that the columns of A represent probability distributions, you normalize each column by dividing by the sum of the elements in that column. This ensures that the sum of probabilities for outgoing links from each node is equal to 1.Initialize the rank vector r: You initialize a rank vector 'r' where each element is initially set to 1/6, indicating equal probability for each node in the network.Define the damping factor (alpha): You set a damping factor 'alpha ' to 0.85, which represents the probability that a user continues to surf the web by following links, rather than jumping to a random page.Create a vector s: You create a vector 's'
where all entries are equal to 1/6, representing the equal probability of landing on any page randomly.Set a convergence threshold: You define a threshold for convergence to determine when the iterative process should stop. In this case, the threshold is set to 1e-6.Iterative PageRank computation: The code iteratively updates the rank vector 'r' using the PageRank formula, which takes into account both the probability of following links (alpha * M * r) and the probability of jumping to random page ((1 - alpha) * s).Check for convergence: At each iteration, the code checks whether the difference between the updated 'r' and the previous 'r' is smaller than the convergence threshold. If it is, the iteration stops.Calculate the eigenvector: The code uses NumPy's eig function to calculate the eigenvalues and eigenvectors of matrix M. It identifies the eigenvector corresponding to the largest eigenvalue.Print results: The code prints the converged PageRank vector and the eigenvector corresponding to the largest eigenvalue, and it also verifies if the columns of M are normalized and if the alpha value is not equal to 1.

Task2
initialize_grid(size): This function creates an initial grid of cells with a specified size and initializes it with a predefined pattern. In this case, you're using a "Glider" pattern, which is a well-known configuration that moves diagonally across the grid.evolve(grid): This function computes the next generation of the grid based on the rules of Conway's Game of Life. It iterates through each cell and determines its fate in the next generation based on the number of live neighbors it has. The rules are:Any live cell with fewer than 2 live neighbors dies (underpopulation).Any live cell with 2 or 3 live neighbors survives.Any live cell with more than 3 live neighbors dies (overpopulation).Any dead cell with exactly 3 live neighbors becomes alive (reproduction).The code then initializes the grid, sets up a function to update the plot for animation, and uses matplotlib.animation.FuncAnimation to create an animation of the Game of Life evolving over a specified number of frames (k). It saves the animation as a GIF and also saves a static image of the final state as a PNG.



This time I learned how to quickly check my own code errors with chatgpt, and then when I encountered a difficult topic with ai is also a very good way to solve the problem!