Grand Valley State University

Project 1 – Snakes and Ladders Analysis

Repository: [github.com/vandents/Snakes-and-Ladders-Markov-Analysis](https://github.com/vandents/Snakes-and-Ladders-Markov-Analysis)

A screenshot of a computer screen

Description automatically generated

Scott VandenToorn

MTH 302-02

Professor Brian Drake

18 February 2020

A command line application was written in Python for this project. The application is able to perform an analysis on any sized square board with any configuration of snakes and ladders. A link to the repository is located on the cover page. I recommend giving it a run on your own machine (make sure the terminal is full screen). The game board in Figure 1 shows the configuration that we’ll analyze in this write up. The board has four consecutive snakes in a row and a single ladder starting at square 6 and ending all the way up at square 22. The length of the board is 26 squares. In order to win, the player must make it to the last square or beyond; the player does not have to land exactly on the last square to win.

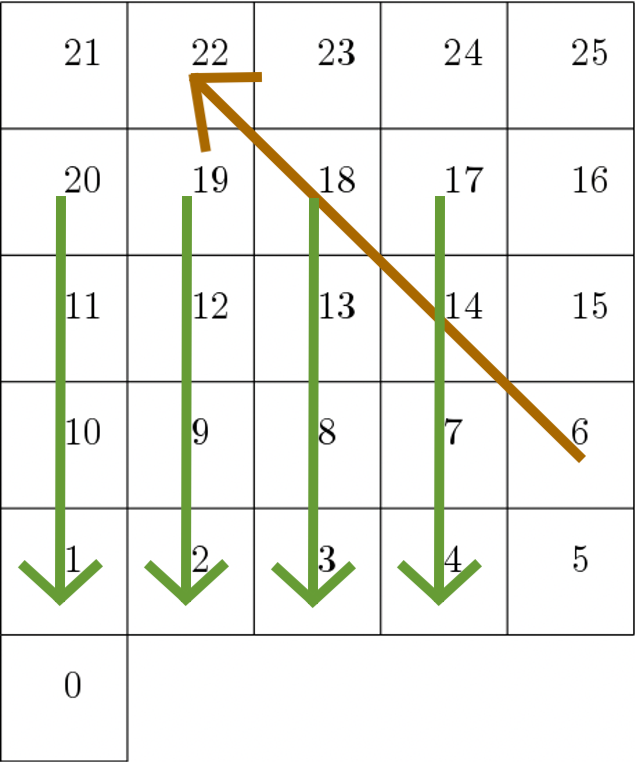


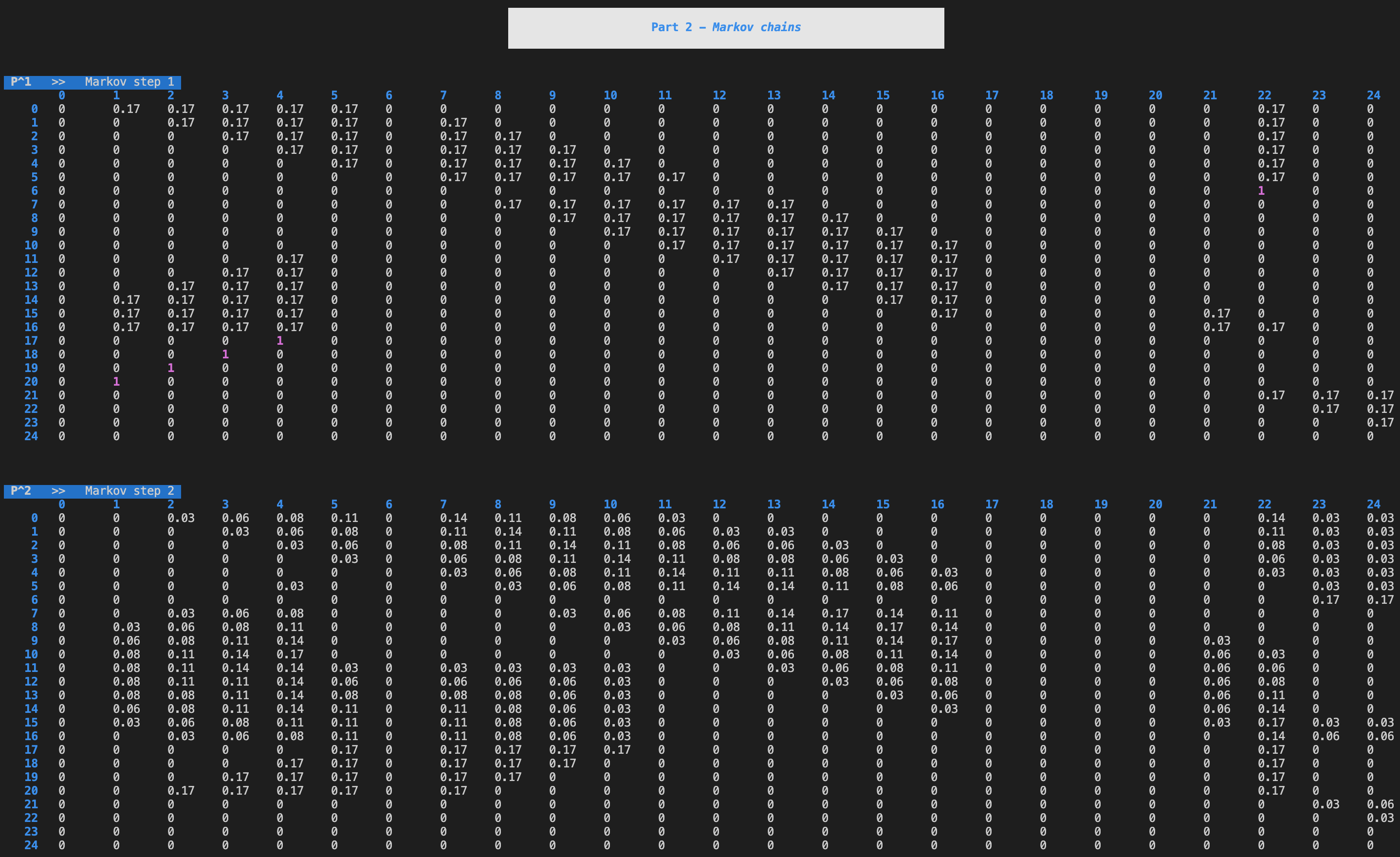
Figure 1. Game board to be analyzed

The matrix in Figure 2 is the transition matrix for our game. Figure 1 shows us that there is a snake or ladder in the squares 6, 17, 18, 19, 20. The snakes and ladders force the probabilities in columns 6, 17, 18, 19, 20 to be zero since it isn’t actually possible to land on these squares. Deleting columns removes probabilities and causes rows to not add up to 1. To counteract this, the deleted probability is added to the outlet of a snake/ladder. The rows 6, 17, 18, 19, 20 all have a 1 in the column of the corresponding snake/ladder end square because there is a 100% probability that the player will go to the end square next.

A close up of a computer

Description automatically generated

Figure 2. Transition matrix for current game variant



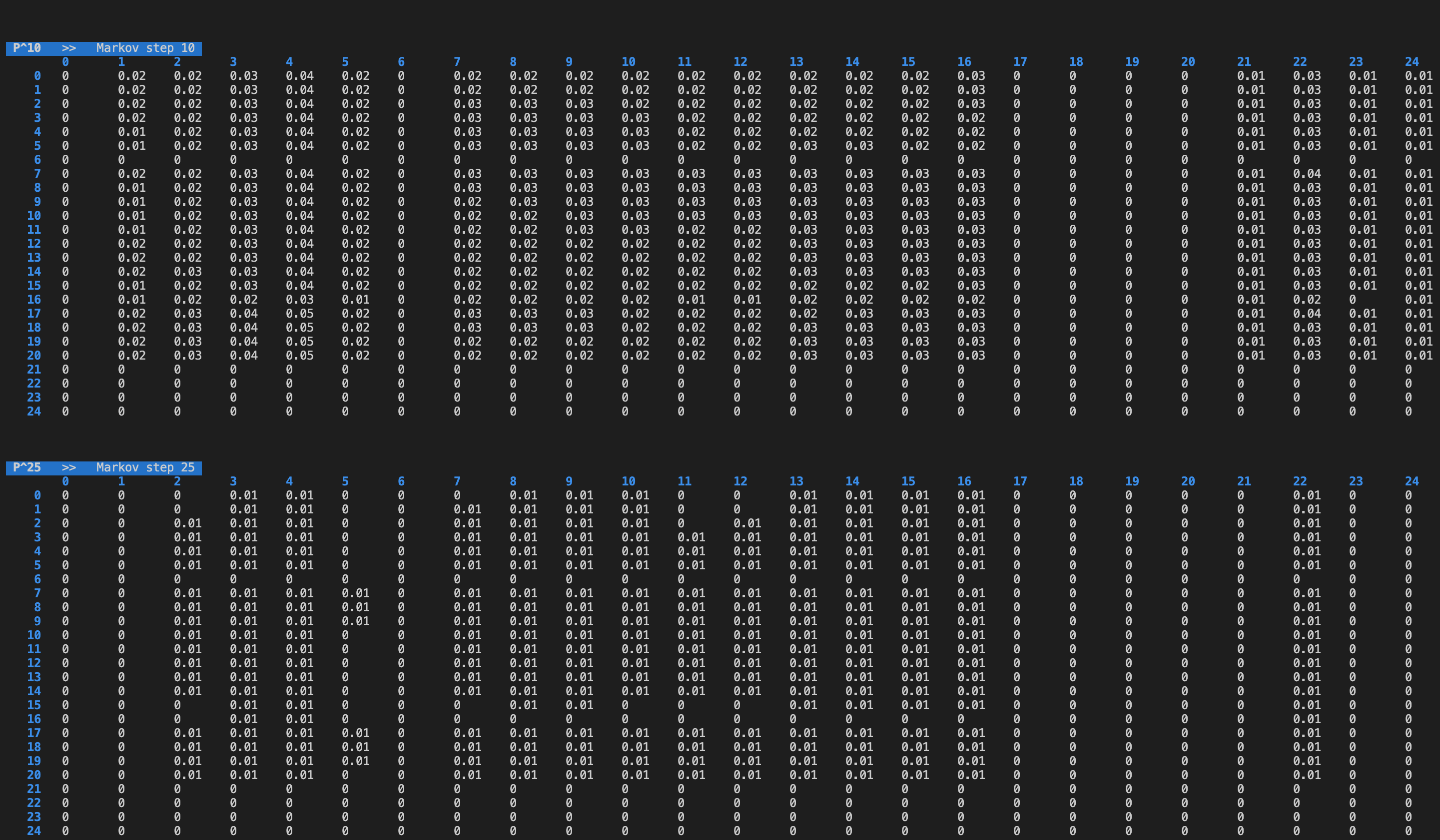


Figure 3. Markov chain at steps 1, 2, 10, 25

Let P be a matrix equal to the transition matrix with the last row and column removed. The Markov matrices in Figure 3 were computed by raising to the power of the step number.

If the player started at square 0, you would expect that after two turns the player could not possibly be on square 1 since the minimum sum of two dice rolls is 2. At step 2 of the Markov chain, you find that there is zero probability of being on square one if the player started on square 0; the value at row = 0, col = 1 is 0. At step 25, you find there is still a lingering probability that the player will be at a square located before the 4 consecutive snakes. This is due to the player getting sent back to one of the beginning squares after landing on a snake.

A screenshot of a computer

Description automatically generated

Figure 4. Eigenvalues and eigenvectors of the transition matrix

The eigenvalues and eigenvectors in Figure 4 were calculated using NumPy. Probability matrices always have an eigenvalue of 1 due to their rows summing to 0. Let be an eigenvector of probability matrix .

The vector in the first row of *eVectors*, , is the eigenvector corresponding to .

Matrix was created by deleting the last row and column of our transition matrix. The smallest such that every entry of is within 0.01 of zero was computed by looping through increasing values of until the condition was met. Figure 5 shows the calculated smallest value of as 26.

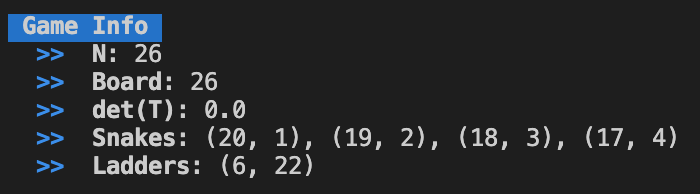


Figure 5. Information about the current game variant

A close up of a computer

Description automatically generated

A close up of a computer

Description automatically generated

Figure 6. Estimation and computation of , difference between estimation and computation

Figure 6 shows the results of estimating and computing . was estimated by computing . An alternative formula was also used, . There difference between the estimation and computation was very small.

Overall, this project was an exceptional learning experience. First of all, it was a great introduction to Python. The Python math libraries are really easy to use. Second, expressing a solution programmatically really helps solidify my understanding of a topic. One of the benefits of programming is that it forces you to break a problem down and think critically about every step of the solution.