**Introduction:**

In the era of computers and technology, we have seen many extraordinary new apps. Coding and programming have become a dominant industry, and many developers have come up with exceptional forms of entertainment which include game development. Modern games have become very spectacular and as aspiring young coders, we wanted to explore the field of game production. Snake and Ladder is a very popular game that is played by both children and adults. So, we decided to make a snake and ladder game which adds some new features to this well-known board game. Our game has many interesting new features which we will be discussing later in the report. We will also calculate the complexities, discuss the game structure, and analyze the major functions in our code.

**Implementation and Features:**

We implemented the game using the graph data structures and algorithms. In most normal snake and ladder games, the board is generated with snakes and ladders being placed in certain fixed positions every time the game is run. In our game, the snakes and ladders are placed randomly every time the game is run. Moreover, we allow the user to define the size of the board as well as the difficulty of the game. As the difficulty level increases, the number of snakes increases. Another feature of the game is that each time a user rolls a 6, he gets to play again. However, there is only a max of 3 moves that he can do in one turn. If he rolls 6 three times in a row, then he can only move 2 times each time he moves 6 positions. If he 6 twice in a row then rolls any number other than 6, then he can move three times in a row. As for the actual implementation of the graph, we decided to use the adjacency list, which shows the adjacency of various vertices in the graph, instead of an adjacency matrix.

**Adjacency List vs Adjacency Matrix:**

Snakes and Ladders, a game of chance and colorful squares, might seem like a simple childhood pastime. But beneath its playful facade lies a fascinating connection to the world of graphs and data structures. When representing this game as a graph, the choice between an adjacency list and an adjacency matrix becomes crucial. While both can technically work, the adjacency list shines brighter for several compelling reasons:

1. Sparse Graph:

Unlike a typical grid-based game, Snakes and Ladders boasts a sparse graph. Many squares have no direct connections to others since each tile would only have 1 destination. An adjacency matrix, with its fixed size for all squares, becomes inefficient. It wastes space storing zeros for non-existent connections, bloating memory usage. The adjacency list, however, thrives on sparsity. It stores only the actual connections for each tile, creating a compact and memory-friendly graph representation. In the game we developed, we allow the user to generate a new board every time the game is run. This is a very expensive process since it essentially rebuilds the entire graph. Now imagine we are going to build a 10 by 10 board. This means that the game will have 100 tiles. In an adjacency matrix, we are going to store a fixed 1002 new entries every time we generate a new board. However, in an adjacency list, we are going to only store 100 new values since each node will only have 1 connection to another node.

2. Graph Traversal:

Searching the graph for possible moves is a core operation in Snakes and Ladders. An adjacency list excels here. Every tile in the game would essentially have only one destination. In an adjacency matrix, searching involves scanning the entire row, potentially wasting time on irrelevant squares, especially for sparse boards. It would be required to check the relationship between the current tile and every other tile in the game. This can hamper the performance of algorithms like finding the luckiest path to win the game. However, with an adjacency list, there will be no searching involved since each tile in the list will only point to one other tile which is its destination. So, the complexity of using an adjacency matrix would be O(n), while the complexity of using an adjacency list would be O(1). This makes the game run faster and way more efficiently.

In conclusion, while both adjacency list and matrix can represent a Snakes and Ladders graph, the list reigns supreme for its memory efficiency as well as graph traversal efficiency.

Paragraph for board class explaining the goal of each function.

Paragraph for dice class explaining the goal of each function.

Paragraph for the player class explaining the goal of each function.

Conclusion.