SNAKE AND LADDERS GAME

A Course End Project Report

ADVANCED DATA STRUCTURES LABORATORY (A8513)

In partial fulfillment of the requirements for the award of the degree of

BATCHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

M.Rohith Naidu
(22881A0537)

Under the guidance of

Mr. Naresh Goud M
Assistant Professor
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified
Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India.

January, 2024

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

Affiliated to JNTUH, Approved by AICTE, Accredited by NAAC with A++ Grade, ISO 9001:2015 Certified
Kacharam, Shamshabad, Hyderabad – 501218, Telangana, India.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Course End Project titled **"Snake and Ladders game"** is carried out by **Mr. M. Rohith Naidu**, Roll Number **22881A0537** towards **A8513 – Advanced Data Structures Laboratory** course and submitted to **Department of Computer Science and Engineering**, in partial fulfilment of the requirements for the award of degree of **Bachelor of Technology** in **Department of Computer Science and Engineering** during the Academic year 2023-24.

Instructor:	**Head of the Department:**
Mr. Naresh Goud M	**Dr. Ramesh Karnati ,**
Assistant Professor,	Head of the Department,
Dept of Computer Science and Engineering,	Dept. of Computer Science and Engineering,
Vardhaman College of Engineering,	Vardhaman College of Engineering,
Hyderabad.	Hyderabad. .

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We wish to express our deep sense of gratitude to **Mr. Naresh Goud M**, Assistant Professor, Department of Computer Science and Engineering, Vardhaman College of Engineering, for his able guidance and useful suggestions, which helped us in completing the design part of potential project in time.

We particularly thankful to **Dr. Ramesh Karnati**, Associate Professor & Head, Department of Computer Science and Engineering for his guidance, intense support and encouragement, which helped us to mould our project into a successful one.

We show gratitude to our honorable Principal **Dr. J.V.R.Ravindra**, for having provided all the facilities and support.

We avail this opportunity to express our deep sense of gratitude and heartfelt thanks to **Dr. Teegala Vijender Reddy**, Chairman and **Sri Teegala Upender Reddy**, Secretary of VCE, for providing a congenial atmosphere to complete this project successfully.

We also thank all the staff members of Computer Science and Engineering for their valuable support and generous advice. Finally, thanks to all our friends and family members for theircontinuous support and enthusiastic help.

M.Rohith Naidu

(22881A0537)

TABLE OF CONTENTS

➢ Introduction

◎The Snake and Ladder game is a classic board game that has been enjoyed by people of all ages for generations. It involves two or more players who take turns rolling a six-sided die and moving their game pieces forward based on the outcome of the dice roll. The game board consists of numbered squares, and players aim to reach the final square to win the game.

➢ Problem Statement

◎ Implement a console-based Snake and Ladder game in C, allowing two players to take turns rolling a dice and moving their tokens on a 10x10 game board. The game should handle snakes and ladders at specific positions on the board, affecting the player's position accordingly. The objective is to be the first player to reach position 100.

Objectives

◎The objectives of this project are to:

1. Develop a console-based Snake and Ladder Game in C.

2. Implement a turn-based system where players roll dice to move on the board.

3. Incorporate snakes and ladders with corresponding effects on player positions.

4. Design the game to continue until a player reaches or exceeds a specified final position.

5. Provide an engaging and interactive experience for two players in a text-based environment.

Primary objectives are -

1. Game Development:

- Create a console-based Snake and Ladder Game using the C programming language.

2. Turn-Based System:

- Implement a turn-based mechanism allowing players to roll dice and progress on the game board.

3. Special Board Elements:

- Incorporate snakes and ladders on the board, affecting player positions based on specific locations.

4. Game Continuation:

- Design the game to persist until a player reaches or surpasses a predefined final position.

5. Interactive Experience:

- Provide an engaging and interactive gaming experience for two players within a text-based environment.URL Extraction: The crawler identifies and extracts hyperlinks from the parsed HTML, adding new URLs to the URL frontier for subsequent crawling.

6. Player Position Validation:

- Ensure that player positions are within the valid range of the game board.

7. Visual Representation:

- Display the game board after each turn, providing a visual representation of player positions.

8. Game Termination:

- Implement a termination condition for ending the game, such as a player reaching a specific position.

➢ SYSTEMREQUIREMENTS

◎**Algorithms**:

1. Initialize Game:

- Create a game board with positions 1 to 100.

- Initialize player positions at the starting point (e.g., position 1).

2. Player Turn:

- Display the current positions of Player 1 and Player 2.

- Prompt the user to choose Player 1's turn, Player 2's turn, or Exit.

3. Dice Roll:

- Generate a random number between 1 and 6 to simulate a dice roll.

4. Update Player Position:

- Add the dice roll value to the current player's position.

- Check for snake or ladder positions and update accordingly.

5. Check Win Condition:

- If a player reaches or surpasses position 100, end the game, and declare the player as the winner.

6. Display Board:

- Visualize the current state of the game board after each turn.

7. Switch Players:

- Enqueue both players in a queue data structure for turn management.

- Dequeue the players in sequence for their turns.

8. Repeat:

- Repeat the process until a player wins or the user chooses to exit.

➢ <u>Data Structures</u>

1. Queue:

- Purpose: To manage the turn-based gameplay, ensuring that players take their turns in a sequential order.

- Operations:

  - Enqueue: Add a player to the end of the queue.

  - Dequeue: Remove a player from the front of the queue.

2. Node Structure:

- Purpose: To represent each player in the queue as a node with player information.

- Attributes:

  - `player`: An array to store the player identifier (e.g., "$P1$").

  - `next`: A pointer to the next node in the queue.

3. Array:

- Purpose: To represent the game board and keep track of player positions.

- Structure

- An array with 100 elements (positions 1 to 100), where each element holds information about the player at that position.

4. Dice Roll Function:

- Purpose: To simulate the rolling of a six-sided die.

- Return Value:

  - An integer between 1 and 6.

➢ DESIGN&IMPLEMENTATION

Design/Solution:

Game Board Representation:

- Represent the game board as a grid with positions numbered from 1 to 100.
- Visualize the board with rows and columns, showing the positions of players.

Player Representation:

- Utilize a queue-based approach to represent players and manage turn order.

Dice Roll Mechanism:

- Use a random number generator to simulate dice rolls.
- Generate a random number between 1 and 6 to determine player movement.

Player Movement:

- Update the player's position based on the dice roll.
- Check for snake and ladder positions and adjust accordingly.

Winning Condition:

- Define the winning condition as reaching position 100.
- Upon reaching this position, declare the player as the winner.

Snakes and Ladders Configuration:

- Define the positions of snakes and ladders on the board.
- Implement logic to handle the impact of landing on these positions.

Turn-Based System:

- Establish a turn-based system using a queue.
- Players take turns in the order of their arrival in the queue.

User Input and Choices:

- Prompt the user to choose between Player 1's turn, Player 2's turn, or exiting the game.
- Validate user input to ensure a correct choice.

Error Handling:

- Implement error-handling mechanisms to handle unexpected situations gracefully.
- Provide meaningful error messages for better user understanding.

User Interface:

- Enhance the user interface with clear messages, prompts, and an aesthetically pleasing board display.
- Display the current positions of both players after each turn.

Code Structure:

- Organize the code into modular functions for better readability and maintainability.
- Use appropriate data structures, such as queues, to manage player turns.

Initialization:

- Initialize players at the starting position (1) before the game begins.
- Seed the random number generator for consistent and reproducible dice rolls during testing.

Testing Considerations:

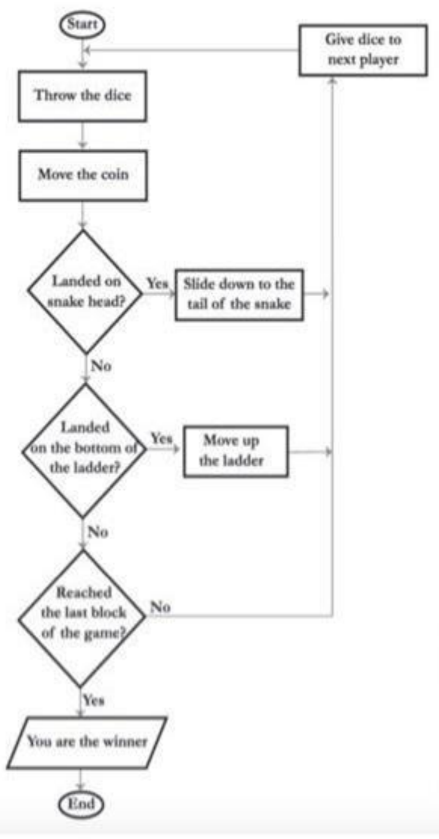
- Plan and conduct extensive testing to ensure the correctness of the game logic.
- Test various scenarios, including normal gameplay, snake encounters, ladder climbs, and victory conditions.

Conclusion:

This design outlines the key components and mechanisms of the Snake and Ladder game solution in C. By adhering to these design principles, we can implement a robust and enjoyable game with clear rules and functionalities.

➢ **<u>Flow Chart</u>**

1) Convert the following **flowchart** in the equivalent **pseudo-code** representation for the algorithm in how to play snacks and ladders game.



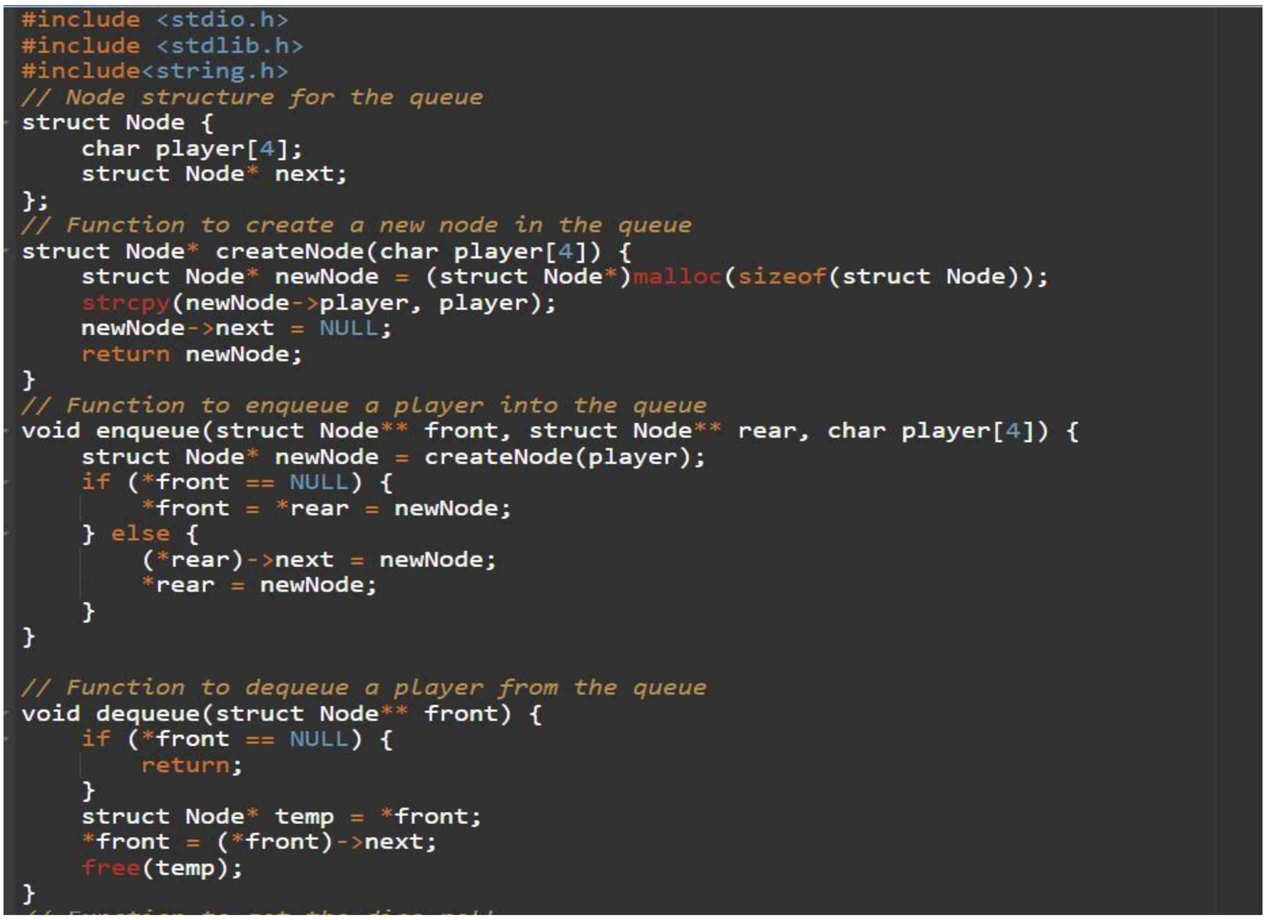
➢

SOURCE CODE :

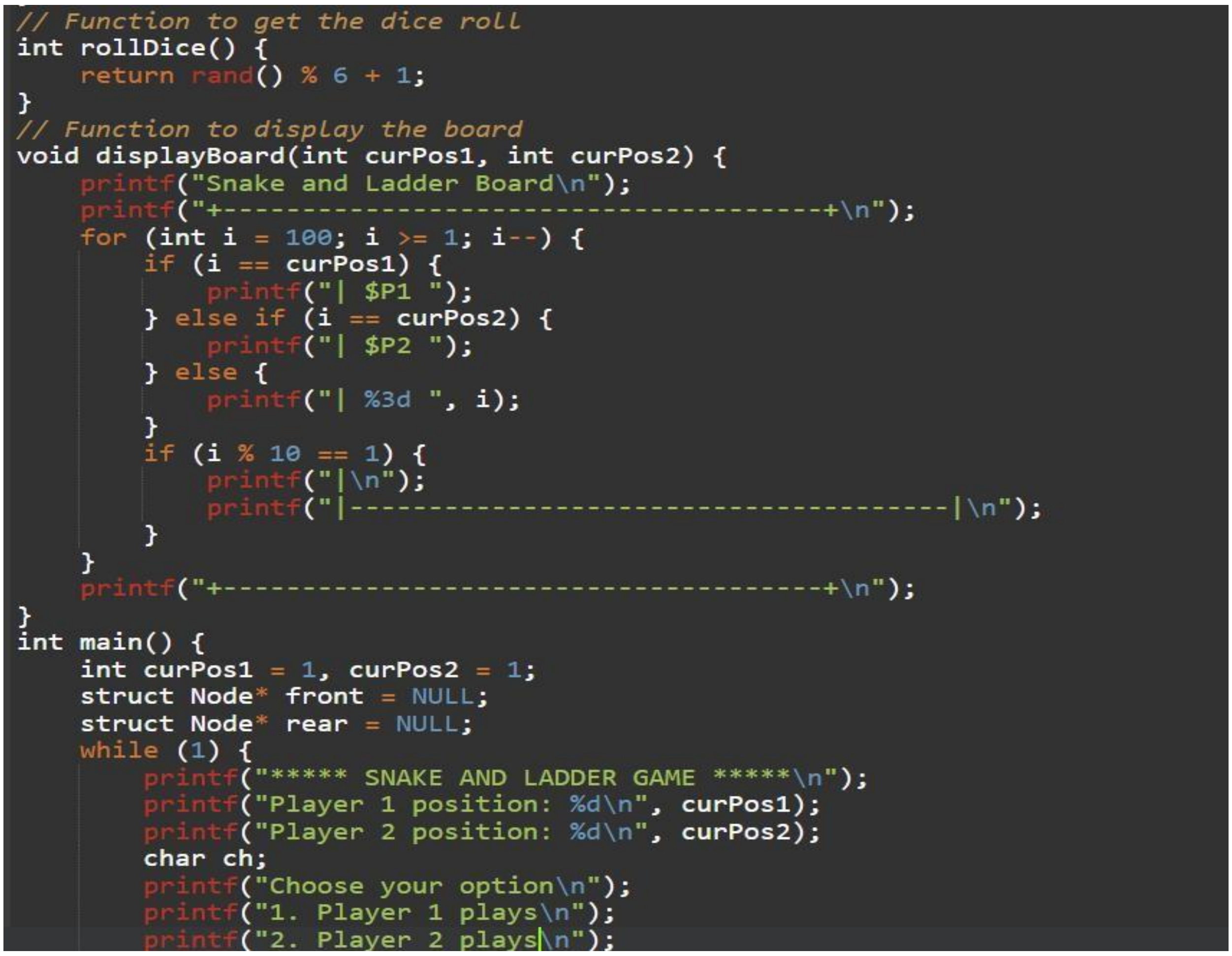
```c
#include  <stdio.h>

#include <stdlib.h>

#include<string.h>

// Node structure for the queue

struct Node {

   char       player[4];

   struct Node* next;

};

// Function to create a new node in the queue

struct Node* createNode(char player[4]) {

   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

   strcpy(newNode->player, player);

   newNode->next = NULL;

   return newNode;

}

// Function to enqueue a player into the queue

void enqueue(struct Node** front, struct Node** rear, char player[4]) {

   struct Node* newNode = createNode(player);

   if (*front == NULL) {

      *front = *rear = newNode;

   } else {

      (*rear)->next = newNode;

      *rear = newNode;

   }

}


// Function to dequeue a player from the queue

void dequeue(struct Node** front) {

   if (*front == NULL) {

      return;
```

```
    }
    struct Node* temp = *front;
    *front = (*front)->next;
    free(temp);
}
// Function to get the dice roll
int rollDice() {
    return rand() % 6 + 1;
}
// Function to display the board
void displayBoard(int curPos1, int curPos2) {
    printf("Snake and Ladder Board\n");
    printf("+---------------------------------+\n");
    for (int i = 100; i >= 1; i--) {
        if (i == curPos1) {
            printf("| $P1 ");
        } else if (i == curPos2) {
            printf("| $P2 ");
        } else {
            printf("| %3d ", i);
        }
        if (i % 10 == 1) {
            printf("|\n");
            printf("|---------------------------------|\n");
        }
    }
    printf("+---------------------------------+\n");
}
int main() {
    int curPos1 = 1, curPos2 = 1;
    struct Node* front = NULL;
    struct Node* rear = NULL;

    while (1) {

        printf("***** SNAKE AND LADDER GAME *****\n");

        printf("Player 1 position: %d\n", curPos1);

        printf("Player 2 position: %d\n", curPos2);

        char ch;

        printf("Choose your option\n");

        printf("1. Player 1 plays\n");

        printf("2. Player 2 plays\n");

        printf("3. Exit\n");

        scanf(" %c", &ch);

        int dice = rollDice();

        printf("Dice Roll: %d\n", dice);

        switch (ch) {

            case '1':

                curPos1 += dice;

                break;

            case '2':

                curPos2 += dice;

                break;

            case '3':

                exit(0);

                break;

            default:

                printf("Incorrect choice. Try Again\n");

                continue;

        }

        // Check for snakes and ladders

        if (curPos1 == 99 || curPos1 == 65 || curPos1 == 25)

            curPos1 = 1;

        else if (curPos1 == 70)
```

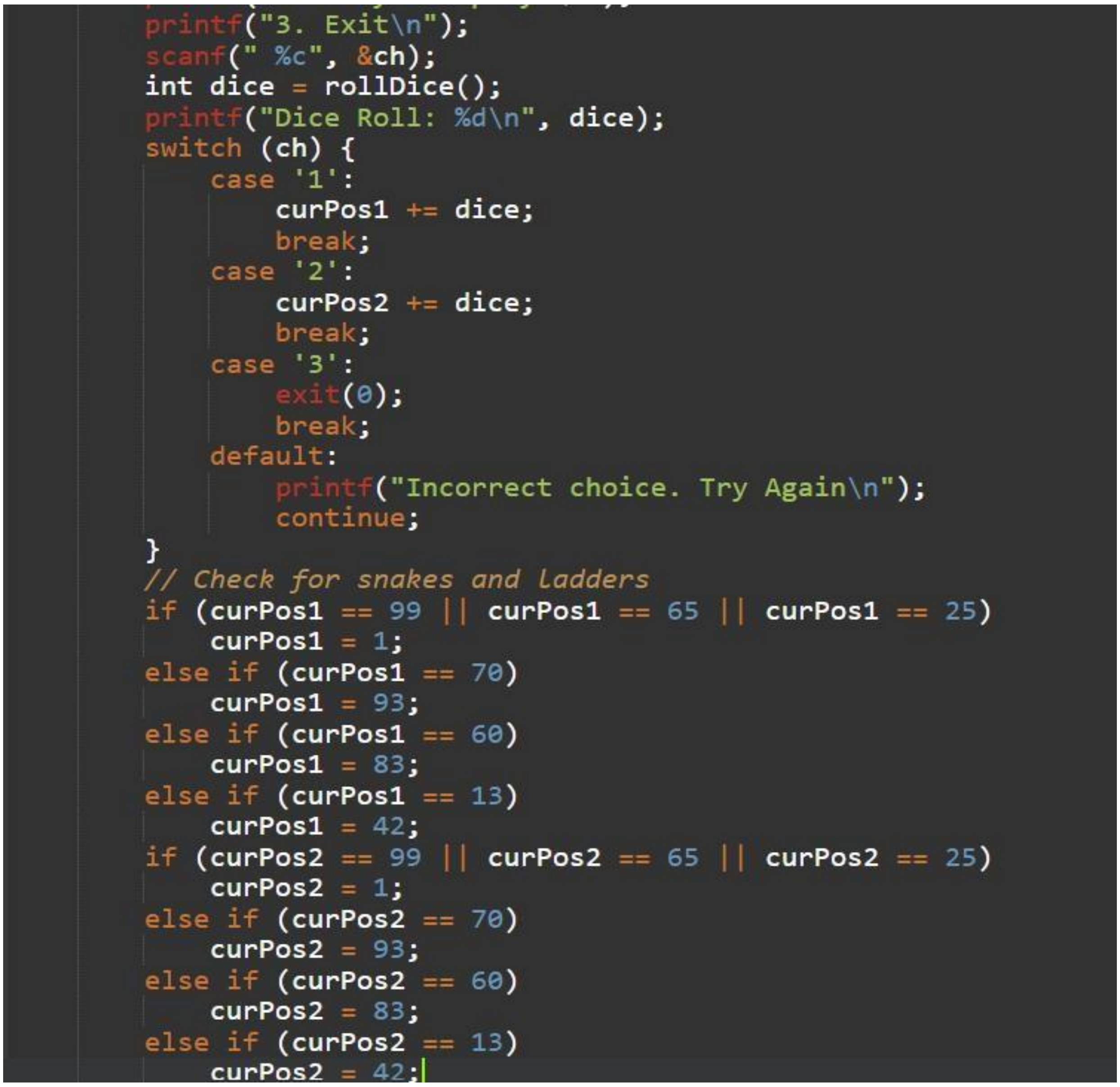
```
            curPos1 = 93;
        else if (curPos1 == 60)
            curPos1 = 83;
        else if (curPos1 == 13)
            curPos1 = 42;
        if (curPos2 == 99 || curPos2 == 65 || curPos2 == 25)
            curPos2 = 1;
        else if (curPos2 == 70)
            curPos2 = 93;
        else if (curPos2 == 60)
            curPos2 = 83;
        else if (curPos2 == 13)
            curPos2 = 42;
        // Display the board
        displayBoard(curPos1, curPos2);
        // Enqueue the players for the next turn
        enqueue(&front, &rear, "$P1$");
        enqueue(&front, &rear, "$P2$");
        // Dequeue the players for the next turn
        dequeue(&front);
        dequeue(&front);
    }
    return 0;
}
```



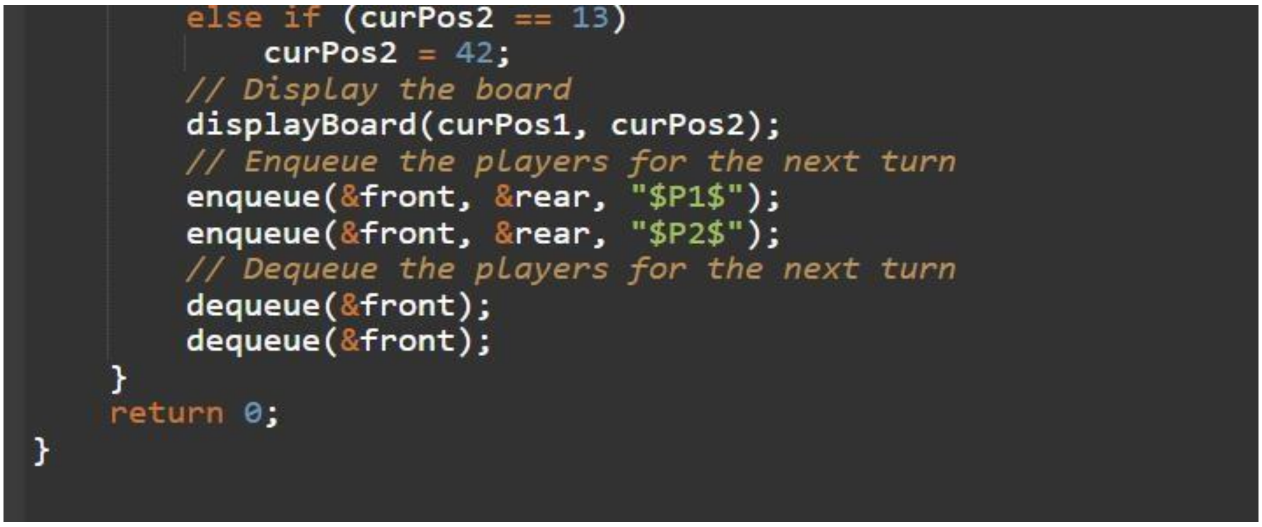
```c
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
// Node structure for the queue
struct Node {
    char player[4];
    struct Node* next;
};
// Function to create a new node in the queue
struct Node* createNode(char player[4]) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    strcpy(newNode->player, player);
    newNode->next = NULL;
    return newNode;
}
// Function to enqueue a player into the queue
void enqueue(struct Node** front, struct Node** rear, char player[4]) {
    struct Node* newNode = createNode(player);
    if (*front == NULL) {
        *front = *rear = newNode;
    } else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
}

// Function to dequeue a player from the queue
void dequeue(struct Node** front) {
    if (*front == NULL) {
        return;
    }
    struct Node* temp = *front;
    *front = (*front)->next;
    free(temp);
}
```



```c
// Function to get the dice roll
int rollDice() {
    return rand() % 6 + 1;
}
// Function to display the board
void displayBoard(int curPos1, int curPos2) {
    printf("Snake and Ladder Board\n");
    printf("+--------------------------------------+\n");
    for (int i = 100; i >= 1; i--) {
        if (i == curPos1) {
            printf("| $P1 ");
        } else if (i == curPos2) {
            printf("| $P2 ");
        } else {
            printf("| %3d ", i);
        }
        if (i % 10 == 1) {
            printf("|\n");
            printf("|--------------------------------------|\n");
        }
    }
    printf("+--------------------------------------+\n");
}
int main() {
    int curPos1 = 1, curPos2 = 1;
    struct Node* front = NULL;
    struct Node* rear = NULL;
    while (1) {
        printf("***** SNAKE AND LADDER GAME *****\n");
        printf("Player 1 position: %d\n", curPos1);
        printf("Player 2 position: %d\n", curPos2);
        char ch;
        printf("Choose your option\n");
        printf("1. Player 1 plays\n");
        printf("2. Player 2 plays\n");
```

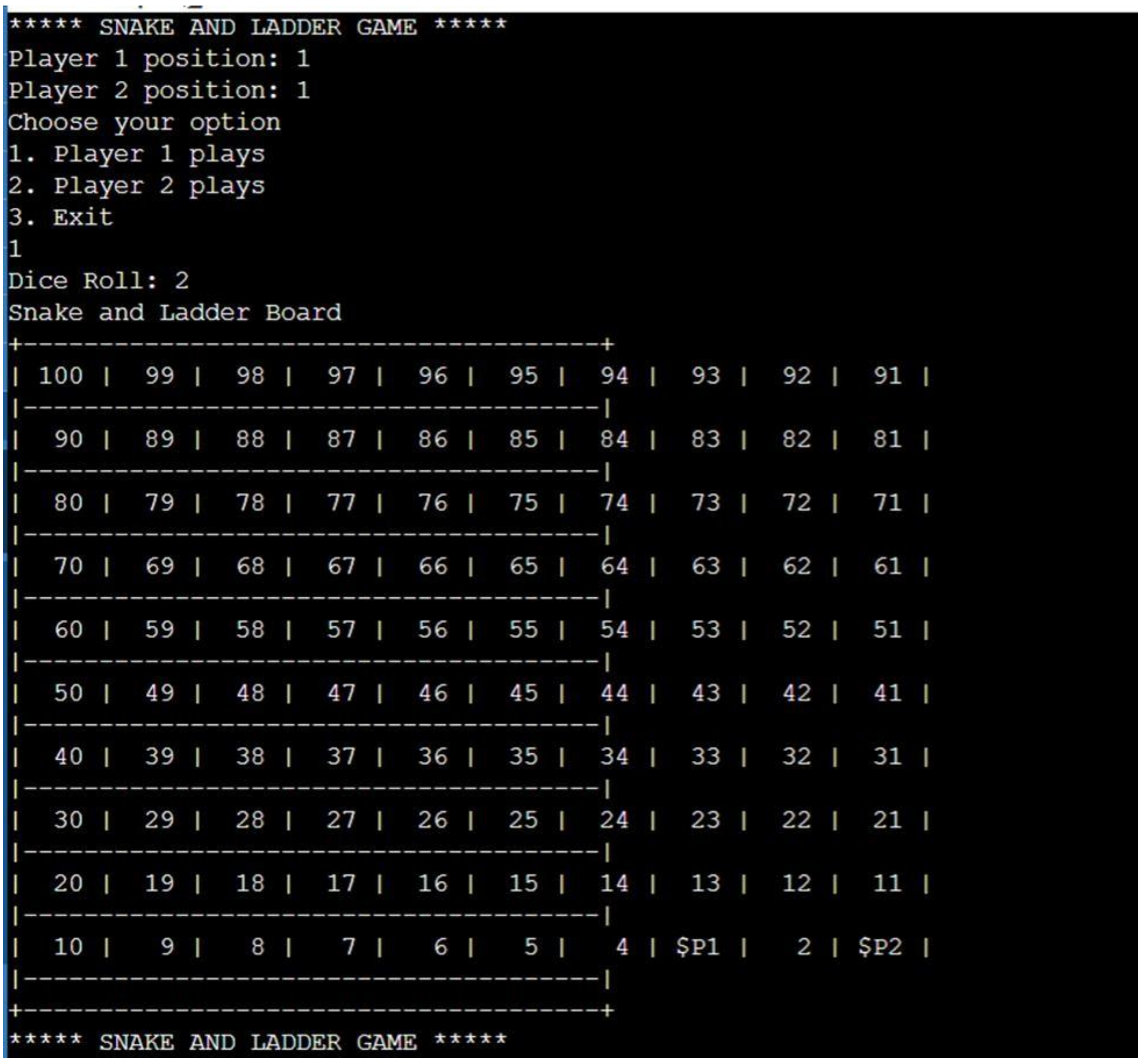


```c
        printf("3. Exit\n");
        scanf(" %c", &ch);
        int dice = rollDice();
        printf("Dice Roll: %d\n", dice);
        switch (ch) {
            case '1':
                curPos1 += dice;
                break;
            case '2':
                curPos2 += dice;
                break;
            case '3':
                exit(0);
                break;
            default:
                printf("Incorrect choice. Try Again\n");
                continue;
        }
        // Check for snakes and ladders
        if (curPos1 == 99 || curPos1 == 65 || curPos1 == 25)
            curPos1 = 1;
        else if (curPos1 == 70)
            curPos1 = 93;
        else if (curPos1 == 60)
            curPos1 = 83;
        else if (curPos1 == 13)
            curPos1 = 42;
        if (curPos2 == 99 || curPos2 == 65 || curPos2 == 25)
            curPos2 = 1;
        else if (curPos2 == 70)
            curPos2 = 93;
        else if (curPos2 == 60)
            curPos2 = 83;
        else if (curPos2 == 13)
            curPos2 = 42;
```

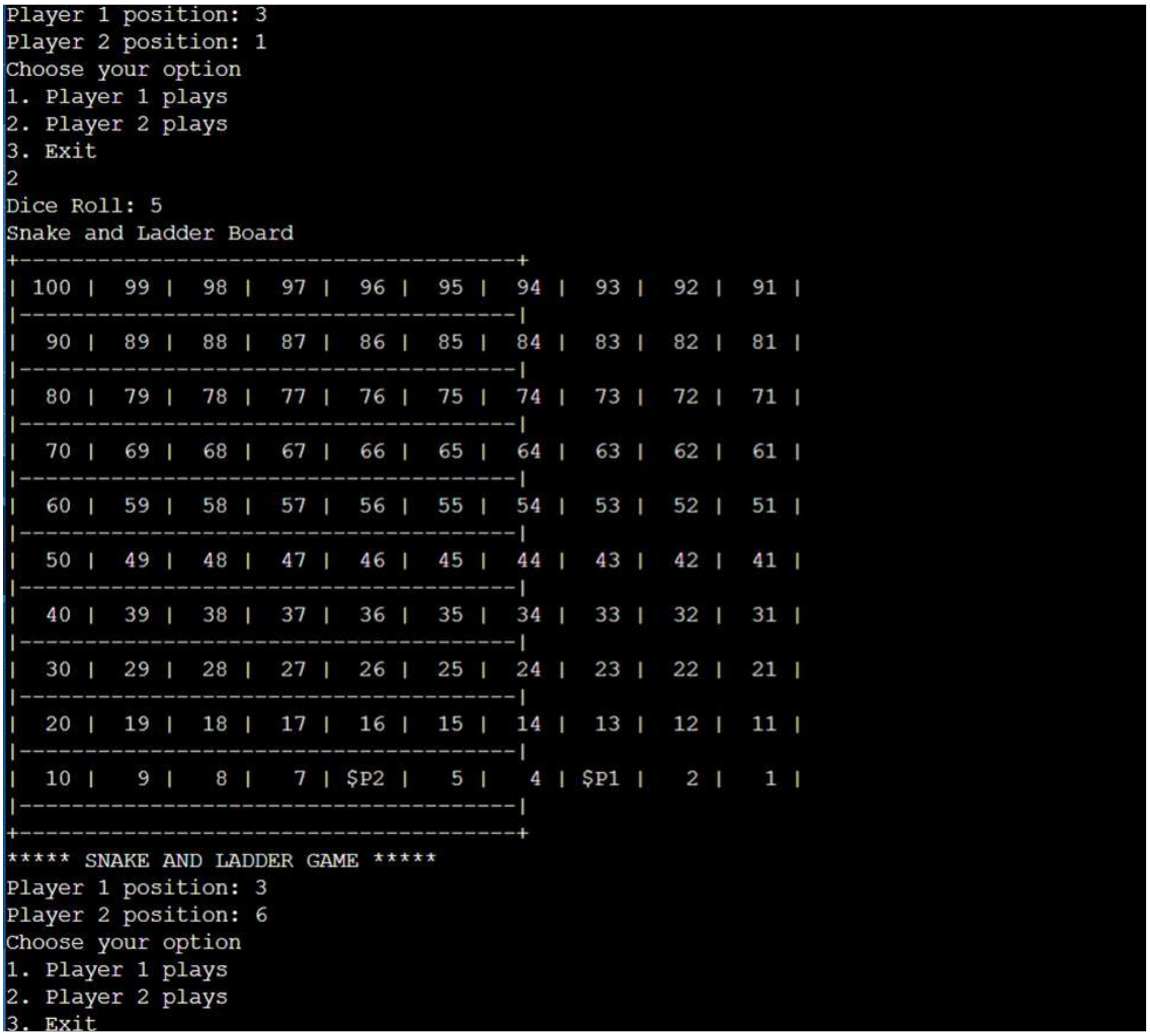


```c
        else if (curPos2 == 13)
            curPos2 = 42;
        // Display the board
        displayBoard(curPos1, curPos2);
        // Enqueue the players for the next turn
        enqueue(&front, &rear, "$P1$");
        enqueue(&front, &rear, "$P2$");
        // Dequeue the players for the next turn
        dequeue(&front);
        dequeue(&front);
    }
    return 0;
}
```

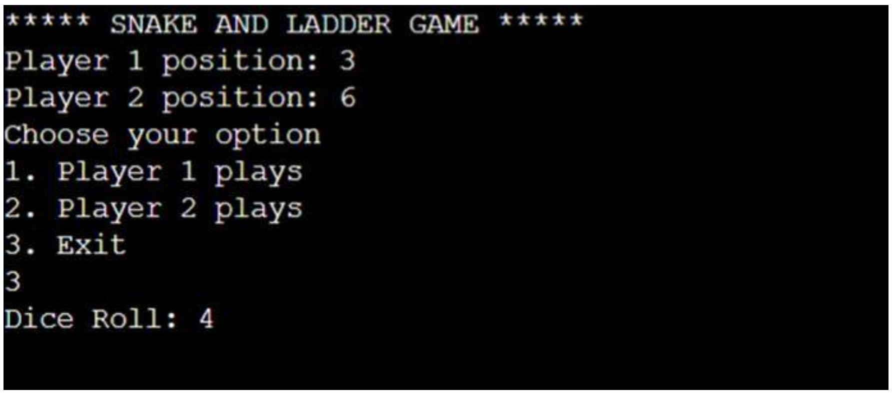
__**Output**__



```
***** SNAKE AND LADDER GAME *****
Player 1 position: 1
Player 2 position: 1
Choose your option
1. Player 1 plays
2. Player 2 plays
3. Exit
1
Dice Roll: 2
Snake and Ladder Board
+---------------------------------------+
| 100 |  99 |  98 |  97 |  96 |  95 |  94 |  93 |  92 |  91 |
|---------------------------------------|
|  90 |  89 |  88 |  87 |  86 |  85 |  84 |  83 |  82 |  81 |
|---------------------------------------|
|  80 |  79 |  78 |  77 |  76 |  75 |  74 |  73 |  72 |  71 |
|---------------------------------------|
|  70 |  69 |  68 |  67 |  66 |  65 |  64 |  63 |  62 |  61 |
|---------------------------------------|
|  60 |  59 |  58 |  57 |  56 |  55 |  54 |  53 |  52 |  51 |
|---------------------------------------|
|  50 |  49 |  48 |  47 |  46 |  45 |  44 |  43 |  42 |  41 |
|---------------------------------------|
|  40 |  39 |  38 |  37 |  36 |  35 |  34 |  33 |  32 |  31 |
|---------------------------------------|
|  30 |  29 |  28 |  27 |  26 |  25 |  24 |  23 |  22 |  21 |
|---------------------------------------|
|  20 |  19 |  18 |  17 |  16 |  15 |  14 |  13 |  12 |  11 |
|---------------------------------------|
|  10 |   9 |   8 |   7 |   6 |   5 |   4 | $P1 |   2 | $P2 |
|---------------------------------------|
+---------------------------------------+
***** SNAKE AND LADDER GAME *****
```



```
Player 1 position: 3
Player 2 position: 1
Choose your option
1. Player 1 plays
2. Player 2 plays
3. Exit
2
Dice Roll: 5
Snake and Ladder Board
+---------------------------------------+
| 100 |  99 |  98 |  97 |  96 |  95 |  94 |  93 |  92 |  91 |
|---------------------------------------|
|  90 |  89 |  88 |  87 |  86 |  85 |  84 |  83 |  82 |  81 |
|---------------------------------------|
|  80 |  79 |  78 |  77 |  76 |  75 |  74 |  73 |  72 |  71 |
|---------------------------------------|
|  70 |  69 |  68 |  67 |  66 |  65 |  64 |  63 |  62 |  61 |
|---------------------------------------|
|  60 |  59 |  58 |  57 |  56 |  55 |  54 |  53 |  52 |  51 |
|---------------------------------------|
|  50 |  49 |  48 |  47 |  46 |  45 |  44 |  43 |  42 |  41 |
|---------------------------------------|
|  40 |  39 |  38 |  37 |  36 |  35 |  34 |  33 |  32 |  31 |
|---------------------------------------|
|  30 |  29 |  28 |  27 |  26 |  25 |  24 |  23 |  22 |  21 |
|---------------------------------------|
|  20 |  19 |  18 |  17 |  16 |  15 |  14 |  13 |  12 |  11 |
|---------------------------------------|
|  10 |   9 |   8 |   7 | $P2 |   5 |   4 | $P1 |   2 |   1 |
|---------------------------------------|
+---------------------------------------+
***** SNAKE AND LADDER GAME *****
Player 1 position: 3
Player 2 position: 6
Choose your option
1. Player 1 plays
2. Player 2 plays
3. Exit
```



```
***** SNAKE AND LADDER GAME *****
Player 1 position: 3
Player 2 position: 6
Choose your option
1. Player 1 plays
2. Player 2 plays
3. Exit
3
Dice Roll: 4
```

Conclusion and Future Work

In conclusion, the Snake and Ladder game implemented in C demonstrates a well-structured and modular design. The use of a queue-based approach for turn management, random number generation for dice rolls, and careful handling of player movement on the board ensures an engaging and fair gameplay experience. The incorporation of snakes and ladders, along with error handling and an intuitive user interface, contributes to the overall quality of the game. By following this design, we can achieve a balance between simplicity and functionality, making the game enjoyable for players. The provided design serves as a foundation for the successful implementation of the Snake and Ladder game in C.

Github Link

https://github.com/RohithNai537/ADS_Report.git

References

1) *https://www.geeksforgeeks.org/snakes-and-ladders-game-using-data-structures/*

2) *https://www.edureka.co/blog/snakes-and-ladder-project-in-java/*