Siddaganga Institute of Technology, Tumakuru

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi,
Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified)

A Report on Open Ended Problem titled

"Snakes and Ladders"

submitted

in the partial fulfillment of the requirements for III semester Data Structures Laboratory

Bachelor of Engineering

in

Computer Science and Engineering

by

Aveek Chakraborty	**1SI21CS024**
Fahad Alam	**1SI21CS043**
Aryan Abrol	**1SI21CS023**
Girish Chandra	**1SI21CS044**

Department of Computer Science & Engineering

(Program Accredited by NBA)

Siddaganga Institute of Technology

B.H Road, Tumakuru-572 103, Karnataka, India.

Web:www.sit.ac.in

March, 2023

Siddaganga Institute of Technology, Tumakuru

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified)

Department of Computer Science and Engineering

(Program Accredited by NBA)

CERTIFICATE

This is to certify that open ended problem titled "Snakes and Ladders" is a bonafide work carried out by Aveek Chakraborty (1SI21CS024), Fahad Alam (1SI21CS043), Aryan Abrol (1SI21CS023), Girish Chandra (1SI21CS044) of III semester **Bachelor of Engineering in Computer Science and Engineering** of the **SIDDAGANGA INSTITUTE OF TECHNOLOGY** (An Autonomous Institution, affiliated to VTU, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified) during the academic year 2022-2023.

Name of the Examiners **Signature with Date**

1. **Mrs. Kavitha M**
2. **Mrs. Nousheen Taj**

ABSTRACT

This project focuses on developing a Snakes and Ladders game using the C programming language. The objective of the game is to provide an engaging and interactive experience to the players. The game will involve any number of players who will compete amongst themselves to reach the finish line. The game will also feature game elements such as ladders, snakes, and difficulty levels that will add to the challenge and excitement of the game.

The project will involve developing the game logic, designing the game interface and implementing simple graphics. The game logic will include defining the rules of the game, setting up the game board, and managing player movements. The game interface will be designed to provide a user-friendly experience and allow players to interact with the game elements. The graphics will be created using simple shapes to enhance the visual appeal of the game.

The game will be developed using the C programming language and will use a console-based interface. This project will provide an opportunity to learn and apply programming concepts and techniques in a practical way while developing an exciting and fun game. Overall, the project will help students to develop skills in programming, game development, and problem-solving.

Data Structures used:

- Singly Linked List with header node
- Arrays

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Snakes and Ladders is a classic board game that has been enjoyed by people of all ages for generations. The game involves rolling a dice to move across a board, which is filled with snakes and ladders that can either advance or set back the players. With the increasing popularity of video games, it is no surprise that Snakes and Ladders has made its way into the digital world. In this project, we will be developing a Snakes and Ladders game using the C programming language.

The game will be developed using the C programming language, which is a popular and widely used language in the field of programming. The project will require developing the game logic, designing the game interface, implementing graphics, and creating sound effects. The game logic will include defining the rules of the game, setting up the game board, and managing player movements. The game interface will be designed to provide a user-friendly experience and allow players to interact with the game elements. Graphics will be created using simple shapes to provide an immersive experience to the players.

When a user selects to launch our game application, he encounters our main screen. Then based upon the user's choice operations will be performed…...!

- Select difficulty level
    1. Easy
    2. Medium
    3. Hard

- Select number of players

- Rolling the dice

Instructions

- Each player starts from step 0.
- Turns will be provided to each player to roll the dice.
- If any player lands at the bottom of a ladder, he/she will be moved up to the top of the ladder.
- If any player lands on the head of a snakes, he/she will be moved down to the bottom of the ladder.
- The first player to reach 100 is the winner.

CHAPTER 2

PROBLEM STATEMENT AND OBJECTIVES

2.1 PROBLEM STATEMENT

To build an application for Snake and ladder games which will allow the user to roll the dice, move across the game board and encounter different snakes and ladders at different steps using C programming language.

2.2 OBJECTIVES

Snakes and Ladders is a popular board game that has been enjoyed by people of all ages for many years. It is an exciting and engaging game that can be played with two or more players, making it an ideal choice for family gatherings, parties, and other social events. In creating a snakes and ladders game using C language, there are several objectives that need to be considered.

The first objective is to create a program that is easy to understand and use. The game should be designed in a way that is intuitive and user-friendly, so that players of all ages can quickly grasp the rules and mechanics of the game. This can be achieved by using clear and concise instructions, visual aids, and a simple interface that is easy to navigate.

The second objective is to ensure that the game is entertaining and engaging. The program should be designed to provide players with a fun and enjoyable experience, with a variety of different challenges and obstacles to overcome. This can be achieved by including different types of snakes and ladders and different difficulty levels.

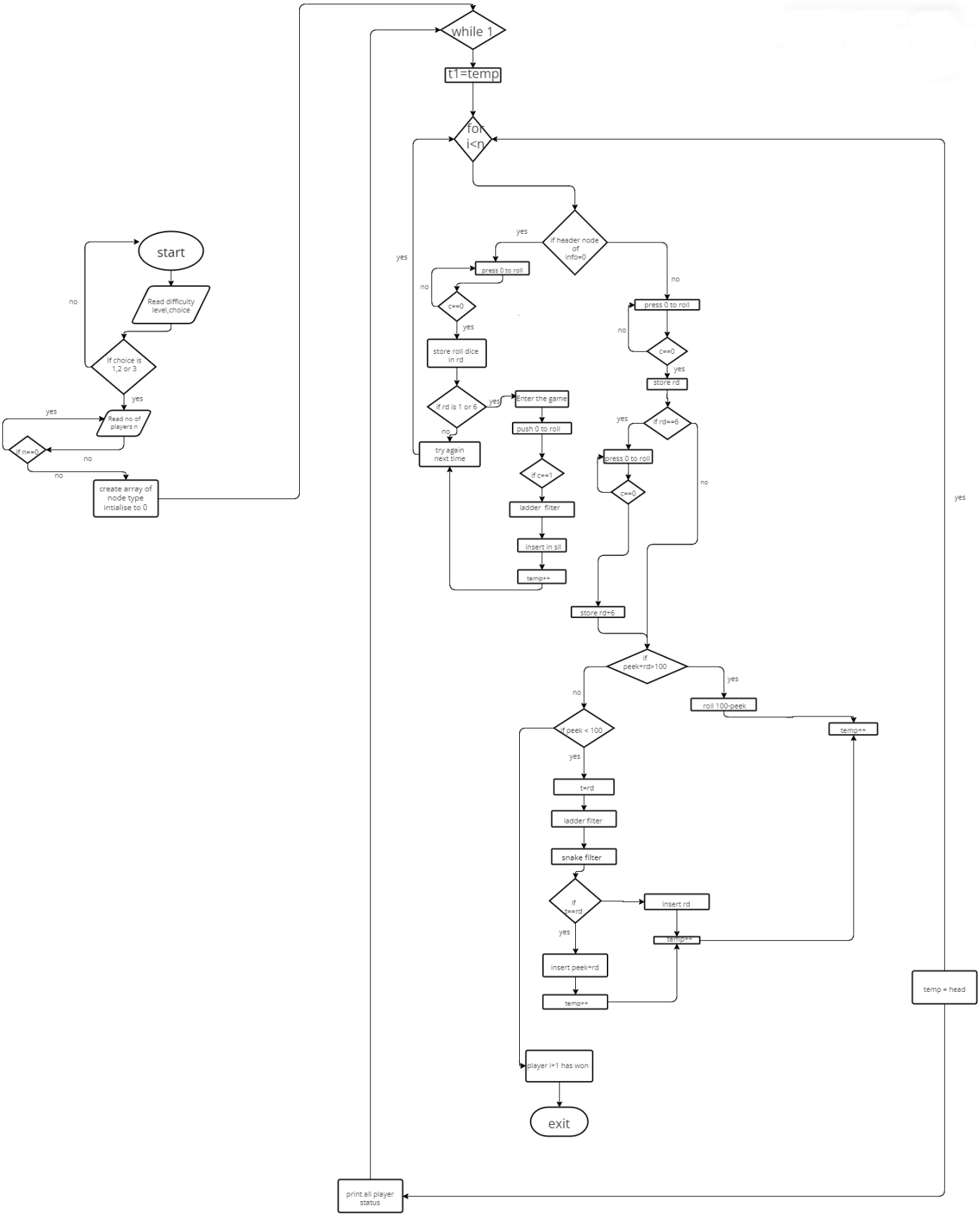
The third objective is to create a game that is customizable and flexible. The program should be designed in a way that allows players to modify the game settings and rules according to their preferences such as deciding the number of players.

The final objective is to create a high-quality program that is reliable and efficient. The game should be designed to run smoothly and without any glitches or bugs, ensuring that players can enjoy a seamless and uninterrupted gameplay experience. This can be achieved by using efficient coding techniques such as dynamic allocation, the use of data structures, rigorous testing, and ongoing maintenance and support. Overall, the creation of a snakes and ladders game using C language requires careful planning and execution to ensure that the program meets the needs of the players and provides a fun and engaging experience.

CHAPTER 3

DESIGN AND IMPLEMENTATION

3.1 DESIGN OF ALGORITHM


start
Read difficulty level,choice
If choice is 1,2 or 3
Read no of players n
If n==0
create array of node type intialise to 0
while 1
t1=temp
for i<n
if header node of info=0
press 0 to roll
c==0
store roll dice in rd
If rd is 1 or 6
Enter the game
push 0 to roll
if c==1
ladder filter
insert in sll
temp++
try again next time
press 0 to roll
c==0
store rd
if rd==6
bress 0 to roll
c==0
store rd+6
if peek+rd>100
roll 100-peek
temp++
if peek < 100
t=rd
ladder filter
snake filter
if t==rd
insert rd
temp++
insert peek+rd
temp++
temp = head
player i+1 has won
exit
print all player status
yes
no

3.2 IMPLEMENTATION

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

typedef struct node
{
    int info;
    struct node *next;
} NODE;

void ins_last(NODE *head, int data)
{
    NODE *newnode, *temp;
    newnode = (NODE *)malloc(sizeof(NODE));
    newnode->info = data;
    newnode->next = NULL;
    temp = head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newnode;
    head->info++;
}

int peek(NODE *head)
{
    NODE *temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
```

```
    return (temp->info);
}

int ladder(int n)
{
    switch (n)
    {
    case 4:
        printf("Congrats!! You encountered a ladder. You have now climbed to 14\n");
        return 14;
    default:
        return n;
    }
}
int ladder2(int p, int n)
{
    switch (n + p)
    {
    case 4:
        printf("Congrats!! You encountered a ladder. You have now climbed to 14\n");
        return 14;
    case 8:
        printf("Congrats!! You encountered a ladder. You have now climbed to 30\n");
        return 30;
    case 21:
        printf("Congrats!! You encountered a ladder. You have now climbed to 42\n");
        return 42;
    case 28:
        printf("Congrats!! You encountered a ladder. You have now climbed to 76\n");
        return 76;
    case 32:
        printf("Congrats!! You encountered a ladder. You have now climbed to 49\n");
        return 49;
    case 50:
        printf("Congrats!! You encountered a ladder. You have now climbed to 67\n");
        return 67;
```

```
    case 71:
        printf("Congrats!! You encountered a ladder. You have now climbed to 92\n");
        return 92;
    case 84:
        printf("Congrats!! You encountered a ladder. You have now climbed to 99\n");
        return 99;
    default:
        return n;
    }
}
int snake(int p, int n, int c)
{
    if (c == 1)
    {
        switch (n + p)
        {
        case 35:
            printf("Oh no!! You encountered a snake. You have now fallen down to 10\n");
            return 10;
        case 39:
            printf("Oh no!! You encountered a snake. You have now fallen down to 6\n");
            return 6;
        case 62:
            printf("Oh no!! You encountered a snake. You have now fallen down to 18\n");
            return 18;
        case 48:
            printf("Oh no!! You encountered a snake. You have now fallen down to 26\n");
            return 26;
        case 88:
            printf("Oh no!! You encountered a snake. You have now fallen down to 24\n");
            return 24;
        case 95:
            printf("Oh no!! You encountered a snake. You have now fallen down to 56\n");
            return 56;
        case 97:
            printf("Oh no!! You encountered a snake. You have now fallen down to 78\n");
            return 78;
```

```
        default:
            return n;
        }
    }
    else if (c == 2)
    {
        switch (n + p)
        {
        case 35:
            printf("Oh no!! You encountered a snake. You have now fallen down to 10\n");
            return 10;
        case 39:
            printf("Oh no!! You encountered a snake. You have now fallen down to 6\n");
            return 6;
        case 62:
            printf("Oh no!! You encountered a snake. You have now fallen down to 18\n");
            return 18;
        case 48:
            printf("Oh no!! You encountered a snake. You have now fallen down to 26\n");
            return 26;
        case 88:
            printf("Oh no!! You encountered a snake. You have now fallen down to 24\n");
            return 24;
        case 95:
            printf("Oh no!! You encountered a snake. You have now fallen down to 56\n");
            return 56;
        case 97:
            printf("Oh no!! You encountered a snake. You have now fallen down to 78\n");
            return 78;
        case 29:
            printf("Oh no!! You encountered a snake. You have now fallen down to 7\n");
            return 7;
        case 12:
            printf("Oh no!! You encountered a snake. You have now fallen down to 5\n");
            return 5;
        case 43:
            printf("Oh no!! You encountered a snake. You have now fallen down to 20\n");
```

```
            return 20;
        case 56:
            printf("Oh no!! You encountered a snake. You have now fallen down to 22\n");
            return 22;
        default:
            return n;
        }
    }
    else if (c == 3)
    {
        switch (n + p)
        {
        case 35:
            printf("Oh no!! You encountered a snake. You have now fallen down to 10\n");
            return 10;
        case 39:
            printf("Oh no!! You encountered a snake. You have now fallen down to 6\n");
            return 6;
        case 62:
            printf("Oh no!! You encountered a snake. You have now fallen down to 18\n");
            return 18;
        case 48:
            printf("Oh no!! You encountered a snake. You have now fallen down to 26\n");
            return 26;
        case 88:
            printf("Oh no!! You encountered a snake. You have now fallen down to 24\n");
            return 24;
        case 95:
            printf("Oh no!! You encountered a snake. You have now fallen down to 56\n");
            return 56;
        case 97:
            printf("Oh no!! You encountered a snake. You have now fallen down to 78\n");
            return 78;
        case 29:
            printf("Oh no!! You encountered a snake. You have now fallen down to 7\n");
            return 7;
        case 12:
```

```
            printf("Oh no!! You encountered a snake. You have now fallen down to 5\n");
            return 5;
        case 43:
            printf("Oh no!! You encountered a snake. You have now fallen down to 20\n");
            return 20;
        case 56:
            printf("Oh no!! You encountered a snake. You have now fallen down to 22\n");
            return 22;
        case 57:
            printf("Oh no!! You encountered a snake. You have now fallen down to 19\n");
            return 19;
        case 77:
            printf("Oh no!! You encountered a snake. You have now fallen down to 34\n");
            return 34;
        case 84:
            printf("Oh no!! You encountered a snake. You have now fallen down to 55\n");
            return 55;
        case 92:
            printf("Oh no!! You encountered a snake. You have now fallen down to 11\n");
            return 11;
        default:
            return n;
        }
    }
}

int roll_dice()
{
    time_t reference = 1646413200;

    struct timespec current;
    clock_gettime(CLOCK_REALTIME, &current);

    struct timespec reference_ts = {.tv_sec = reference, .tv_nsec = 0};

    long long diff_sec = current.tv_sec - reference_ts.tv_sec;
    long long diff_nsec = current.tv_nsec - reference_ts.tv_nsec;
```

```
    long long diff_usec = diff_sec * 1000000 + diff_nsec / 1000;

    if (diff_usec % 7 == 0)
        return 1;
    else if ((diff_usec % 7 == 7))
        return 6;
    else
        return (diff_usec % 7);
}

int main()
{
    int l;
    int i;
    int n;
    int rd;
    int c;
    int t;
    int choice;
    printf(" ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,\n");
    printf("|                                                            |\n");
    printf("|                   * SNAKES AND LADDERS *                   |\n");
    printf("|                                                            |\n");
    printf(" ''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''\n");

    printf("\n\n~~~~~~~~~~~~~~~~~~~~~~~~~INSTRUCTIONS~~~~~~~~~~~~~~~~~~~~~~~~\n\n");
    printf("1) Each player starts from step 0.\n");
    printf("2) Turns will be provided to each player to roll the dice.\n");
    printf("3) If any player lands at the bottom of a ladder, he/she will be moved up to
           the top of the ladder.\n");
    printf("4) If any player lands on the head of a snake, he/she will be moved down to
           the bottom of the ladder.\n");
    printf("5) The first player to reach 100 is the winner.\n\n");
    printf("HAVE FUN !!\n");
    printf("..............................................................\n\n");

    printf("~~~~~~~~~~~~~~~~~~~~CHOOSE DIFFICULTY LEVEL~~~~~~~~~~~~~~~~~~~\n\n");
```

```
l:
    printf("          1:EASY          2:MEDIUM          3:HARD\n\nENTER YOUR CHOICE : ");
    scanf("%d", &choice);
    printf("\n\n");
    printf("............................................................\n\n");

    if (choice == 1 || choice == 2 || choice == 3)
    {
    l0:
        printf("Enter the number of players : ");
        scanf("%d", &n);
        if (n == 0)
        {
            printf("At laest 1 player is needed\n");
            goto l0;
        }
        NODE *head;
        head = (NODE *)malloc(n * sizeof(NODE));
        NODE *temp = head;
        NODE *t1;

        for (int j = 0; j < n; j++)
        {
            temp->info = 0;
            temp->next = NULL;
            temp++;
        }

        while (1)
        {
            temp = head;
            for (i = 0; i < n; i++)
            {
                t1 = temp;
                printf("\nplayer %d turn \n", i + 1);
                if (temp->info == 0)
```

```
{
l1:
    printf("Press 0 to roll dice : ");
    scanf("%d", &c);
    if (c == 0)
    {
        rd = roll_dice();
    }
    else
    {
        goto l1;
    }

    if (rd == 1 || rd == 6)
    {
        printf("Congrats!! You have got %d.You can now enter the game\n",
                rd);
    l2:
        printf("Press 0 to roll dice : ");
        scanf("%d", &c);
        if (c == 0)
        {
            rd = roll_dice();
        }
        else
        {
            goto l2;
        }
        printf("You have got %d after rolling the dice\n", rd);
        rd = ladder(rd);
        ins_last(temp, rd);
        temp++;
    }
    else
    {
        printf("You have got %d. Try again next time\n", rd);
        temp++;
```

```
        }
    }
    else
    {
    l3:
        printf("Press 0 to roll dice : ");
        scanf("%d", &c);
        if (c == 0)
        {
            rd = roll_dice();
            if (rd != 6)
            {
                printf("You have got %d after rolling the dice\n", rd);
            }
        }
        else
        {
            goto l3;
        }

        if (rd == 6)
        {
            printf("Congrats!! you have got 6. You can roll again\n");
        l4:
            printf("Press 0 to roll dice : ");
            scanf("%d", &c);
            if (c == 0)
            {
                rd = roll_dice();
                printf("You have got %d this time\n", rd);
            }
            else
            {
                goto l4;
            }
            rd += 6;
        }
```

```
            if ((peek(temp) + rd) > 100)
            {
                printf("Roll %d to win the game\n", 100 - (peek(temp)));
                temp++;
            }
            else if ((peek(temp) + rd) < 100)
            {

                t = rd;
                rd = ladder2(peek(temp), rd);
                rd = snake(peek(temp), rd, choice);
                if (t == rd)
                {
                    ins_last(temp, peek(temp) + rd);
                    temp++;
                }
                else
                {
                    ins_last(temp, rd);
                    temp++;
                }
            }
            else
            {
                printf("\n.....................................................
                        ..................... \n");
                printf("HURRAYYY!!!!!  Player %d has reached 100 and has won the
                        game in %d valid steps\n", i + 1, (t1->info + 1));
                printf("..........................................................
                        ....................\n");
                return 0;
            }
        }
    }
    temp = head;
    printf("\nPlayer status \n");
```

```
            for (i = 0; i < n; i++)
            {
                printf("Player %d is at %d\n", i + 1, peek(temp));
                temp++;
            }
            printf("\n..... \n");
        }
    }
    else
    {
        printf("Enter valid choice\n\n");
        printf(".....\n\n");
        goto l;
    }
}
```

CHAPTER 4

RESULTS

Figure 4.1: Welcome screen

Figure 4.1 *shows the welcome screen: Here in welcome screen we Get one type of entry i.e selecting Difficulty level.*

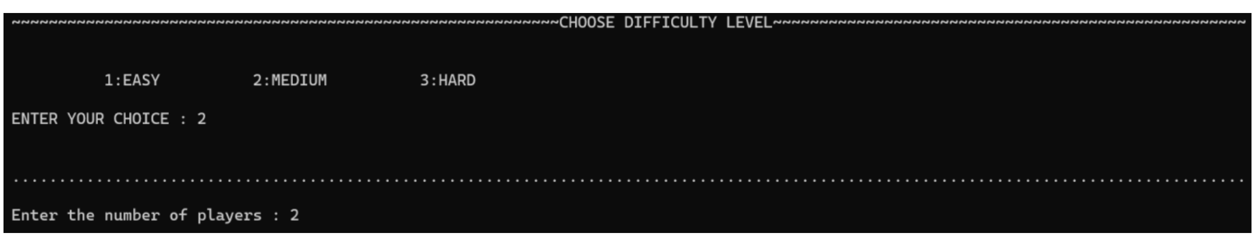


Figure 4.2: Entering number of players

Figure 4.2 *Here number of players is taken in input from User(s).*

```
player 1 turn
Press 0 to roll dice : 0
Congrats!! You have got 6.You can now enter the game
Press 0 to roll dice : 0
You have got 1 after rolling the dice

player 2 turn
Press 0 to roll dice : 0
You have got 4. Try again next time

Player status
Player 1 is at 1
Player 2 is at 0

.....


player 1 turn
Press 0 to roll dice : 0
You have got 5 after rolling the dice

player 2 turn
Press 0 to roll dice : 0
You have got 3. Try again next time

Player status
Player 1 is at 6
Player 2 is at 0

.....
```

Figure 4.3 Different player' turns

Figure 4.3 *shows players entering 0s to roll the dice and getting different outputs.*

```
player 1 turn
Press 0 to roll dice : 0
Congrats!! you have got 6. You can roll again
Press 0 to roll dice : 0
You have got 3 this time

player 2 turn
Press 0 to roll dice : 0
Congrats!! you have got 6. You can roll again
Press 0 to roll dice : 0
You have got 4 this time
Oh no!! You encountered a snake. You have now fallen down to 5

Player status
Player 1 is at 19
Player 2 is at 5

.....
```

Figure 4.4 Player 1 getting 6 and hence getting second chance, Player 2 encountering Snake

Figure 4.4 *shows the details of the players encountering different aspects of the game that is in from of snakes and sixes.*

```
player 1 turn
Press 0 to roll dice : 0
You have got 2 after rolling the dice
Congrats!! You encountered a ladder. You have now climbed to 92

player 2 turn
Press 0 to roll dice : 0
You have got 1 after rolling the dice

Player status
Player 1 is at 92
Player 2 is at 26

.....
```

Figure 4.5 Player 1 encountering a ladder

Figure 4.5 gives the details of Player 1 who encounters a ladder.

CONCLUSION

In conclusion, the Snake and Ladder game created using C language is a fun and interactive project that not only helped to enhance our programming skills but also provided an enjoyable gaming experience. We were able to implement various programming concepts like loops, conditional statements, functions, and arrays to develop a functional game with proper game rules and user interface. Additionally, the project allowed us to showcase our problem-solving skills, creativity, and teamwork.

Overall, the Snake and Ladder game created using C language has been an excellent learning experience that has taught us how to use programming to create engaging and interactive games. We hope that this game will bring joy and entertainment to everyone who plays it.

REFERENCES

1. ANCI, Programming in C, 7th Edition- E.Balagurusamy.
2. Data Structures using C/C++ - Yedidyah Langsam, Moshe J., Augenstein, Aaron M. Tenenbaum.