SNAKE GAME PROJECT

A MINI PROJECT

Academic Year 2021-22 ODD SEMESTER

Department with Specialization: Computer Science and Engineering with Specialisation in Artificial Intelligence & Machine Learning

Semester : I

Course Code : 18CSS101J

Course Title : Programming for Problem Solving

Submitted By

Sai Rishyanth Visinigiri (Reg No: RA2111026010280)

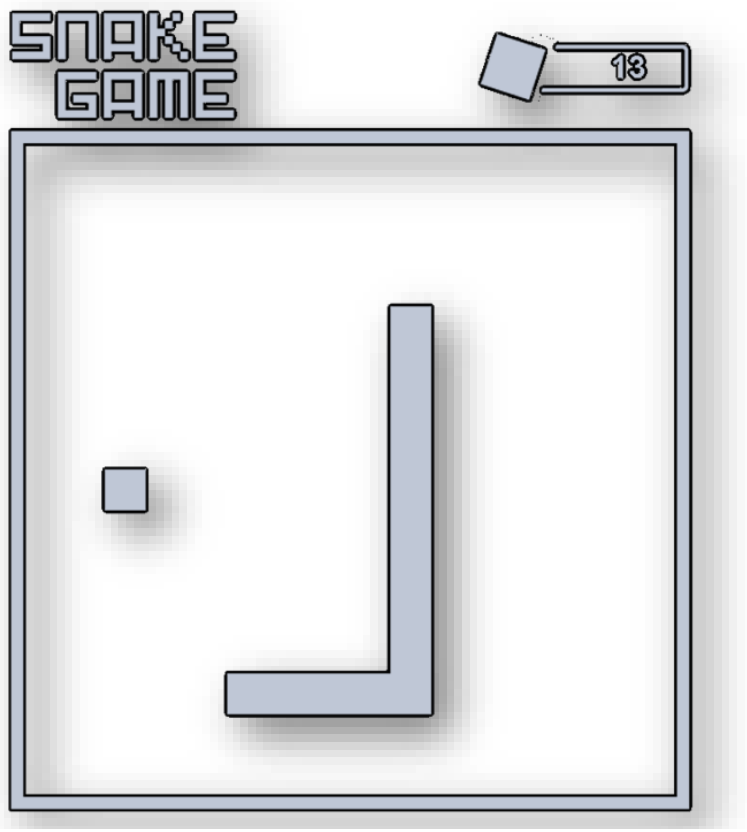
Adithya Naidu Bandaru (Reg No: RA2111026010278)

Under the Guidance of
Dr. A. Suresh
(Associate Professor, NWC)

DEPARTMENT OF COMPUTING
COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR- 603 203

JANUARY 2022

Table of Contents



ABSTRACT

The snake game is a very popular, simplistic, and addictive video game around the globe launched in 1997 with Nokia 6110. It is based on the concept where the player manoeuvres a line (snake) which grows in length every time the line eats more dots 'egg', with the line itself being a primary obstacle as it would die if it collided into itself. As a result, the growth of its length makes the game even more difficult and challenging to play further on.

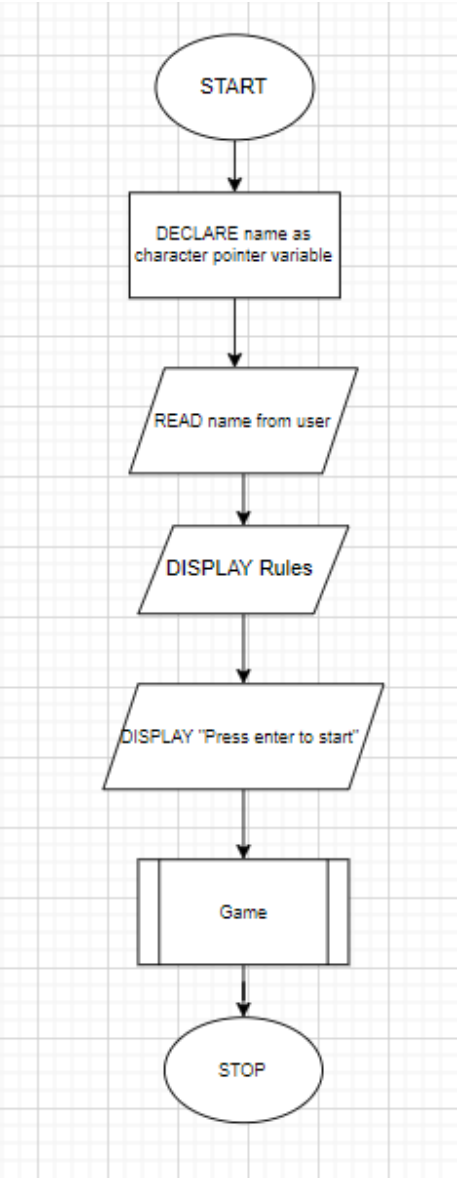
This project aims to explore a new dimension in the traditional snake game to make it more interesting and challenging. The simplicity and the fun element of the game makes it an ideal minor project in C language as we can focus on advanced concepts like multiplayer functionality and dot matrix using which the snake and egg will be displayed. Moreover, switches will be used for start and stop of the game and for snake movements left, right, up, and down.

DESIGN

ALGORITHMS & FLOWHCARTS

USER DETAILS & MAIN MENU

➔ Flowchart:



Pseudocode:

Step 1: START

Step 2: DECLARE name as character pointer variable

Step 3: ACCEPT name from user

Step 4: PRINT Game Rules

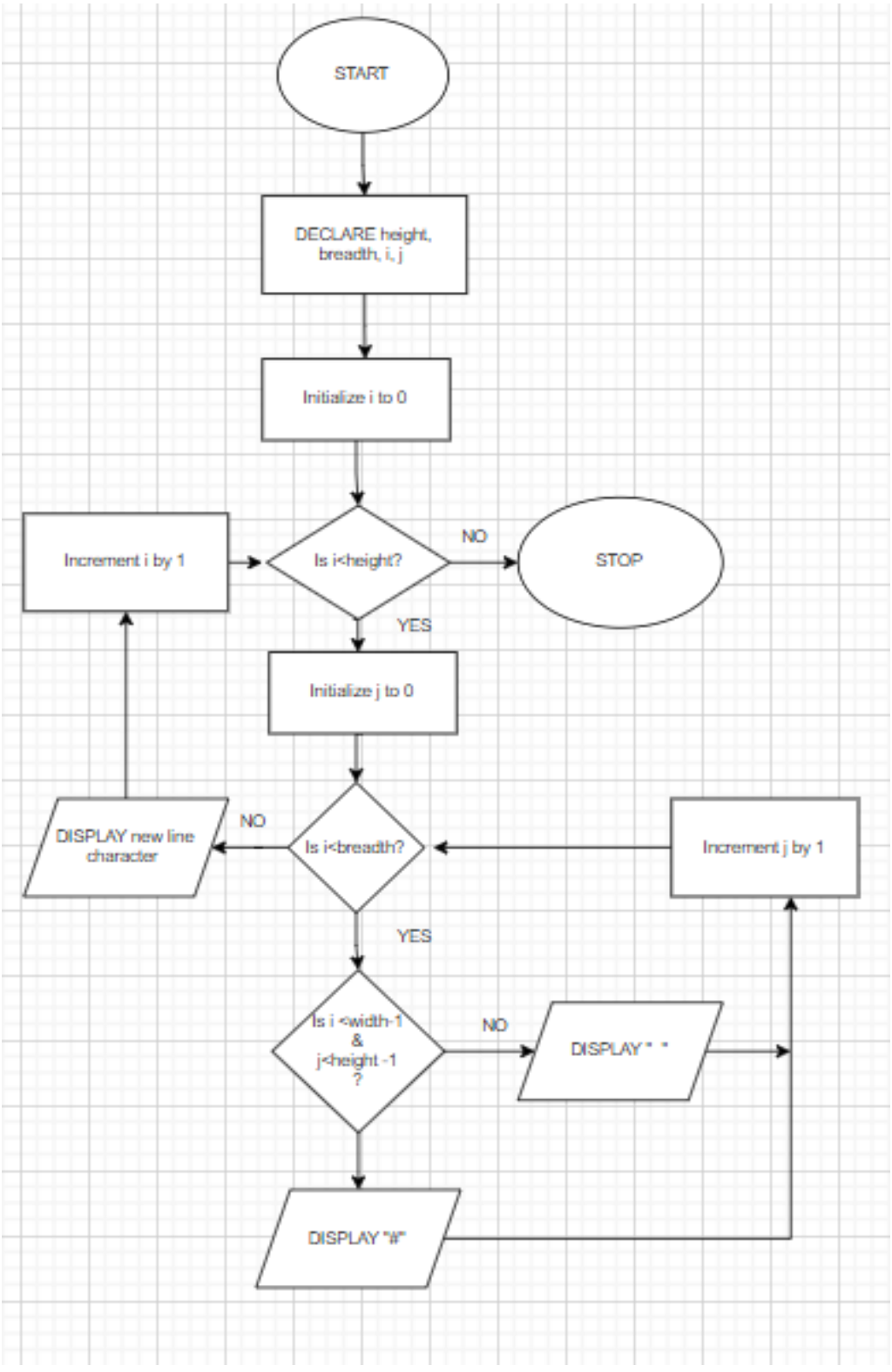
Step 5: PRINT "Press ENTER to Start Game"

Step 5: Subroutine Game

Step 6: END PROGRAM

OUTLINING BOUNDARY (Flowchart)

➔ Flowchart:



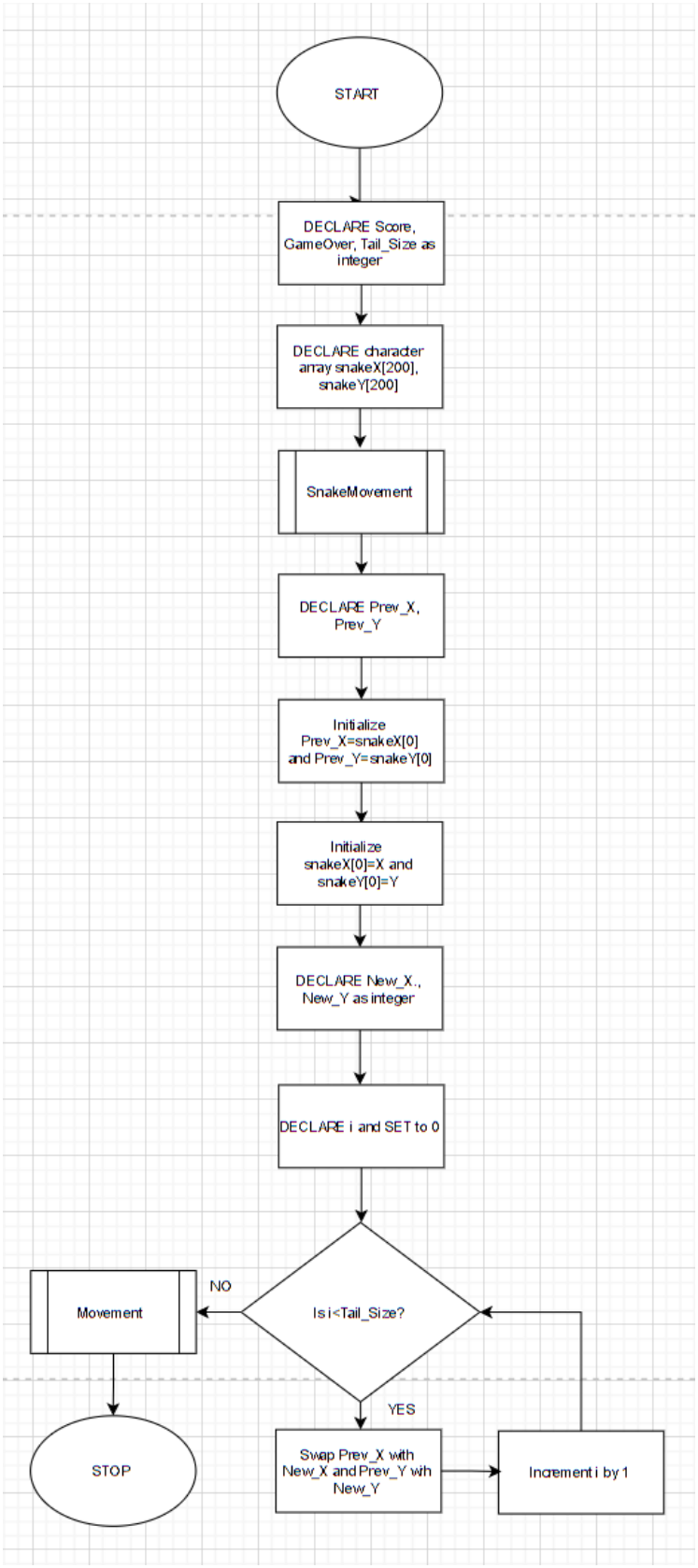
OUTLINING BOUNDARY (Pseudocode)

➔ Pseudocode:

```
CALL Boundary ()
Step 1: START
Step 2: DECLARE height, breadth, i, j as int
Step 3: Initialize I to 0 and height, breadth to 30
Step 4: IF i<height then
Step 5:   Initialize j as 0
Step 6:   IF j<breadth then
Step 7:        IF i<width-1 and j<height -1 then
Step 8:             DISPLAY "#"
Step 9:             Increment j by 1
Step 10:         ELSE
Step 11:            DISPLAY "  "
Step 12:         ENDIF
Step 13:  ELSE
Step 14:     DISPLAY new line character
Step 15:     Increment i by 1
Step 16:  ENDIF
Step 17:  END PROGRAM
END FUNCTION
```

SNAKE SETUP: DECLARATION OF COORDINATES (Flowchart)

➔ Flowchart:



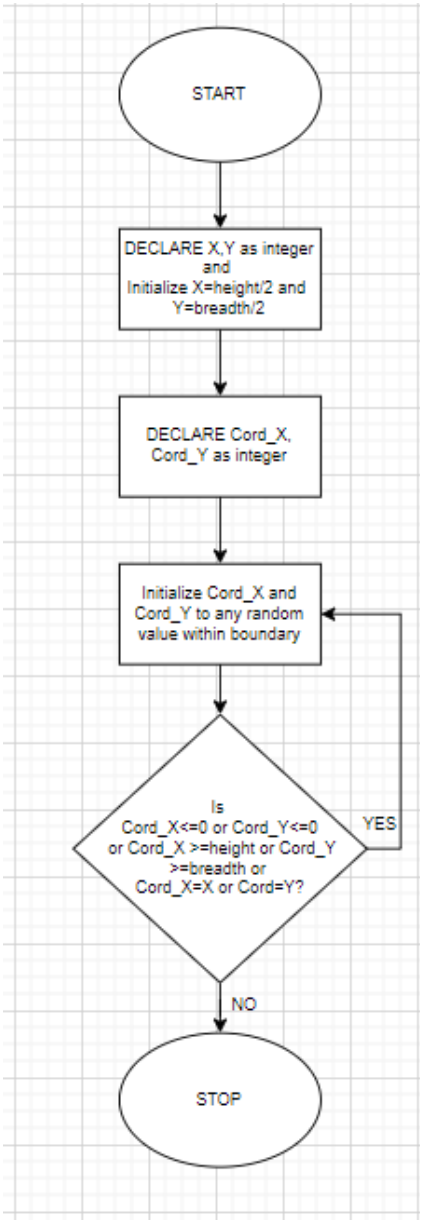
SNAKE SETUP: DECLARATION OF COORDINATES (Pseudocode)

➔ Pseudocode:

```
Step 1: START
Step 2: DECLARE Score, GameOver,
Tail_Size
Step 3: DECLARE character array
snakeX[200] and snakeY[200]
Step 4: Initialize Score, Tail_Size to 0
and GameOver=False
Step 5: CALL SnakeMovement
Step 6: DECLARE Prev_X, Prev_Y as int
Step 7: Initialize Prev_X=snakeX[0] and
Prev_Y=snakeY[0]
Step 8: Initialize snakeX[0]=X and
snakeY[0]=Y
Step 9: DECLARE New_X, New_Y as int
Step 10: DECLARE i and SET to 0
Step 11: FOR i<Tail_Size
Step 12:       swap(Prev_X, New_X)
Step 13:       swap(Prev_Y, New_Y)
Step 14:       Increment i by 1
Step 15: ENDFOR
Step 16: CALL Movement with X and Y
Step 17: END PROGRAM
```

RANDOM GENERATION OF FOOD

➔ Flowchart:

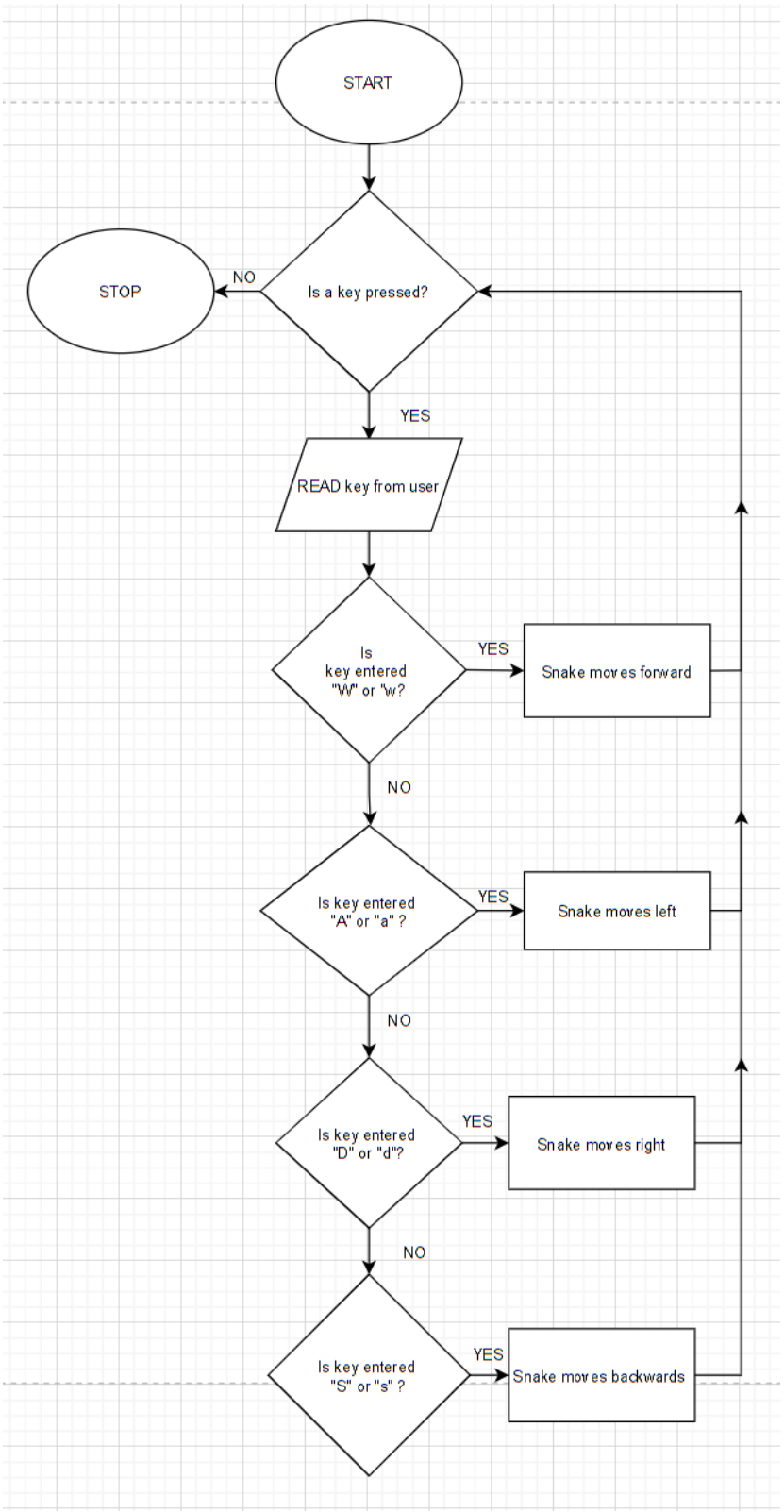


Pseudocode:

```
CALL Food ()

Step 1: START

Step 2: DECLARE X, Y as int

Step 3: Initialize X=height/2 and
Y=breadth/2

Step 4: DECLARE Cord_X, Cord_Y as int

Step 5: Initialize Cord_X=1+rand ()
%height

Step 6: Initialize Cord_Y=1+rand ()
%breadth

Step 7: IF Cord_X==0 or Cord_Y==0 or
Cord_X>=height or Cord_Y>=breadth or
Cord==X or Cord_Y==Y then

Step 8: REPEAT Step 5 and Step 6

Step 9: END PROGRAM
```

MANOEUVRING SNAKE (Flowchart)

➔ Flowchart:



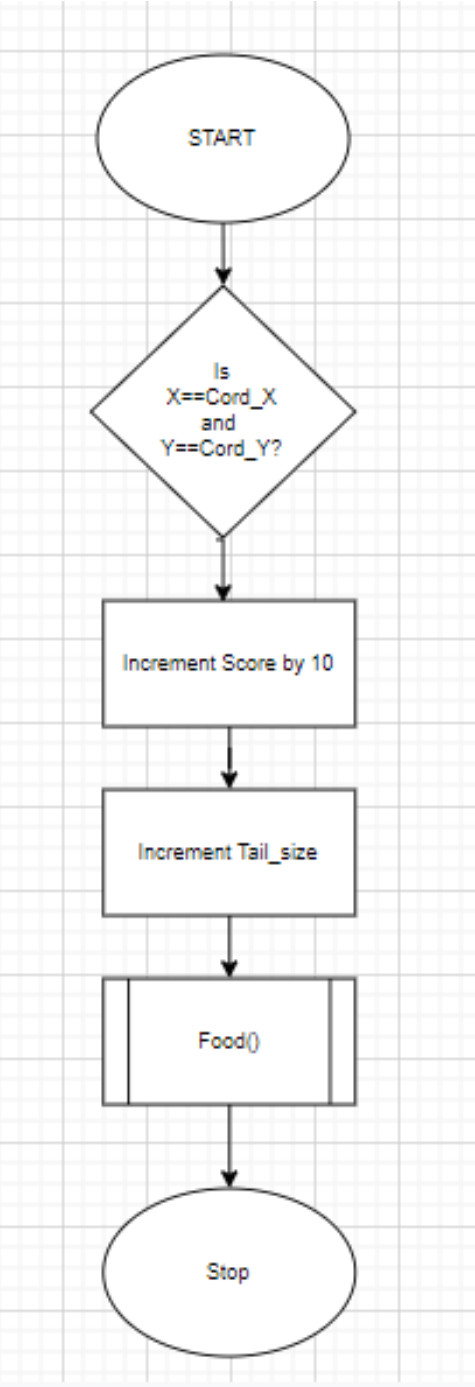
MANOEUVRING SNAKE (Pseudocode)

➔ Pseudocode:

```
CALL Manoeuvring ()
Step 1: START
Step 2: IF (khbit) then
Step 3:         READ a character from user
Step 4:         CASE (char) OF
Step 5:         condition "w" or "W": Flag =1, break
Step 6:         condition "a" or "A": Flag =2, break
Step 7:         condition "d" or "D": Flag =3, break
Step 8:         condition "s" or "S": Flag =4, break
Step 9:         DEFAULT: Exit game, break
Step 9:         ENDCASE
Step 10: ENDIF
END FUNCTION

CALL Movement () with X and Y

Step 1:  Start
Step 2:  CASE (flag) OF
Step 3:     condition 1: X=X-1, break
Step 4:     condition 2: Y=Y-1, break
Step 5:     condition 3: Y=Y+1, break
Step 6:     condition 4: X=X+1, break
Step 7:     DEFAULT: break
Step 8:  ENDCASE
Step 9:  IF flag==1 or flag==4 then
Step 10:              Sleep (25)
Step 11: END PROGRAM
```

GROWTH OF SNAKE SIZE & SCORE INCREMENTATION

➔ Flowchart:

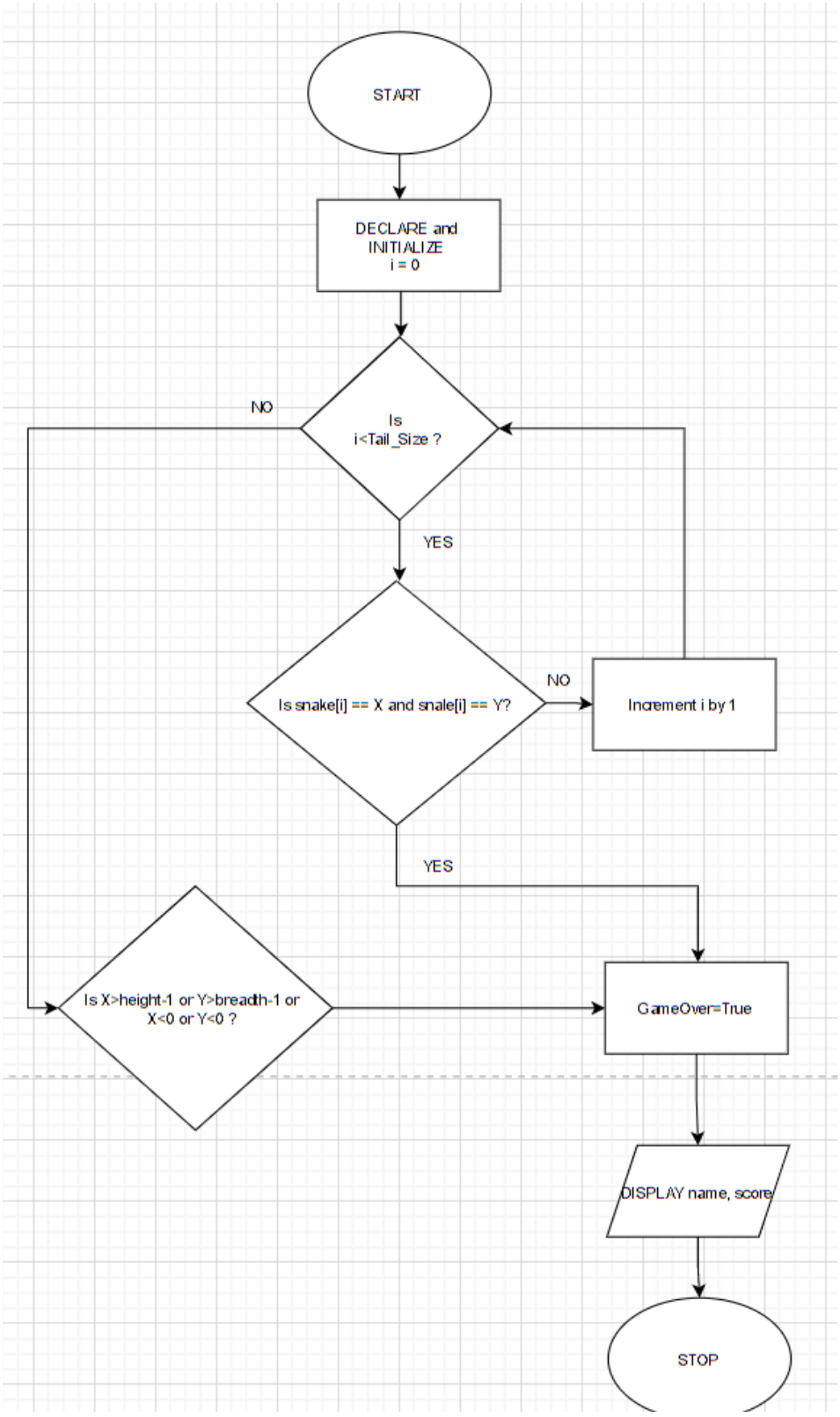


Pseudocode:

```
CALL GROW ()
Step 1: START
Step 2: IF X==Cord_X and Y==Cord_Y then
Step 3:             Increment Score=Score+10
Step 4:             Increment Tail_size by 1
Step 5:             CALL Food with X and Y
Step 6: ENDIF
Step 7: END PROGRAM
END FUNCTION
```

GAME OVER & DISPLAYING SCORE (Flowchart)

➔ Flowchart:



GAME OVER & DISPLAYING SCORE (Pseudocode)

➔ Pseudocode:

```
CALL GameOver ()
Step 1: START
Step 2: SET I to 0
Step 3: FOR i<Tail_Size
Step 4:     IF snakeX[i] ==X and snakeY[i]==Y then
Step 5:          GameOver=True
Step 6:          break
Step 7:     ELSE
Step 8:        Increment i by 1
Step 9:    ENDIF
Step 10: ENDFOR
Step 11: IF X>height-1 or Y>breadth>1 or X<0 or Y<0 then
Step 12:        GameOver=True
Step 13: ENDIF
Step 14: IF GameOver==True then
Step 15:       DISPLAY name, score
Step 16: ENDIF
Step 17: END PROGRAM
END FUNCTION
```

SOURCE CODE

```
// SNAKE GAME PROJECT
//
/*
V114
Sai Rishyanth Visinigiri (RA2111026010280)
Adithya Naidu Bandaru    (RA2111026010278)
*/

// Preprocessor Directives Used in the program
#include <stdio.h>
#include <stdbool.h>
#include <conio.h>
#include <time.h>
#include <windows.h>
#include <process.h>
#include <stdlib.h>
#include <ctype.h>

// Macros used in the program
// ASCII values of UP, DOWN ,LEFT & RIGHT keys
#define UP 72
#define DOWN 80
#define LEFT 75
#define RIGHT 77


// Globala variable declarations

int snake_len,score=0,Snake_life,len,bend_no;
bool gameOver=false;
char key;

// Function Prototypes usen in the program

void Start();
void BoundaryOutline();
void Food();
void Manoeuvre();
void Up();
void Down();
void Left();
void Right();
void GameStart();
void QuitGame();
```

```
void GameLag(long double);
void Bend();
int Score_Life();
int ScoreDisplay();
void gotoxy(int x,int y);
void GotoXY(int x,int y);

// Structure to store the coordinates of  food and snake head
struct Coord{
  int X_Coord,Y_Coord,direction;
};

typedef struct Coord Coordinate;
Coordinate food,snake,snakeBody[30],bend[500];

// Main function
int main(){

  char key;
  // Displays Main Menu of the Game
  Start();
  // CLears the output screen
  system("cls");
  // Loads the Game
  GameStart();
  snake_len=5;
  // Assigning the inital coordinates of the snake
  snake.X_Coord=25;
  snake.Y_Coord=20;
  // Initial direction of snake movement towards right
  snake.direction=RIGHT;

  // Displays the boundary of the game
  BoundaryOutline();
  // Generates food within the boundary
  Food();
  Snake_life=3;
  bend[0]=snake;
  // Initial Manoeuvring of the Snake
  Manoeuvre();
  return 0;

}
```

```
void Start(){
  printf(
        "\t\t\t\t   ======   =     =       =      =  =  =====\n"
        "\t\t\t\t   =        = =   =      = =     = =   =\n"
        "\t\t\t\t   ======   =  =  =     =   =    ==    =====\n"
        "\t\t\t\t        =   =   = =    =     =   = =   =\n"
        "\t\t\t\t   ======   =     =   =       =  =  =  =====\n"

  );
  char name[100];
  printf("\n\n\t\t\t\t\t\tPlease enter your name: ");
  scanf("%s",name);
  printf("\n\n\t\t\t\t\t\tWELCOME %s TO THE SNAKE GAME!! :)\n\n",name);
  printf("\n\t\t\t\t\tPress any key to start\n");
  getch();
  system("cls");
  // Game Instructions
  printf("\n\n");
  printf(
    "\t\t======================================GAME
RULES=============================================\n"
    "\t\t=  1. Feed the snake with as much food as possible and watch it
grow.                        =\n"
    "\t\t=  2  Use arrow keys to turn the snake toward food.
=\n"
    "\t\t=  3. Each time the snake eats the food, the score is incremented
by 10.                     =\n"
    "\t\t=  4. The longer the snake's tail grows, the higher your score.
=\n"
    "\t\t=  5. If the snake runs into its own tail or the surrounding
wall, the snake loses a life. =\n"
    "\t\t=  6. Snake gets 3 lives after which the game is over.
=\n"

"\t\t=================================================================
======================\n"

    );

    printf("\n\nPress any key to start the game....");
    // Exits the game if the user presses the escape key
    if(getch()==27)
        exit(0);
}
```

```
// Loading Process of the Game
void GameStart(){
  int i,j;
  gotoxy(38,14);
  // Loading message displayed
  printf("GAME ZONE!! LOADING...");
  gotoxy(38,15);
  for(i=1;i<=20;i++){
    for(j=0;j<=200000000;j++);
    // Graphic block character displayed slowly
    // Indicating loading of the game
    printf("%c",219);
  }
  gotoxy(38,16);
  printf("GET READY TO PLAY! :)\n");
  printf("\nPress any key to START\n");
  getch();

}
void BoundaryOutline(){
  // Clears the output Screen for the boundary to be displayed
  system("cls");
  int i;
  // Food is generated on the screen
  GotoXY(food.X_Coord,food.Y_Coord);
  printf("0");

  // Boundary created along the x-axis (left to right)

  for(i=10;i<+70;i++){
    GotoXY(i,10);
    printf("$$");
    GotoXY(i,30);
    printf("$$");

  }

  // Boundary created along the y-axis (top to bottom)

  for(i=10;i<31;i++){
    GotoXY(10,i);
    printf("$$");
    GotoXY(70,i);
    printf("$$");
  }}



// Function to display the food
```

```
// At random positions within the boundary function
// rand() function used for random number generation

void Food(){
  // If the coordinates of snake head and food are same
  // Then the snake length is incremented
  if(snake.X_Coord==food.X_Coord && snake.Y_Coord==food.Y_Coord){
    snake_len++;
    time_t TIME;
    TIME=time(0);
    // srand() used before rand() function
    // So that the program does not generate same sequence of numbers each
time it runs
    srand(TIME);
    food.X_Coord=rand()%70;
    if(food.X_Coord<=10)
        food.X_Coord+=11;
    food.Y_Coord=rand()%30;
    if(food.Y_Coord<=10)
        food.Y_Coord+=11;

  }

  // Generation of food for the first time in the Game
  /* Coordinate of food is 0 since global declarations are
     initialized with 0 */

  else if(food.X_Coord==0){
    food.X_Coord=rand()%70;
    if(food.X_Coord<=10)
       food.X_Coord+=11;
    food.Y_Coord=rand()%30;
    if(food.Y_Coord<=10)
         food.Y_Coord+=11;
  }

}



// Function Manoeuvre for Snake Movements through Keyboard Inputs
void Manoeuvre(){
  int i,Esckey;

  // Do-While Loop
  do{
    Food();
    fflush(stdin);
    len=0;
```

```
    for(i=0;i<30;i++){
      // Snake Body
      snakeBody[i].X_Coord=0;
      snakeBody[i].Y_Coord=0;
      if(i==snake_len)
          break;
    }

    GameLag(snake_len);
    BoundaryOutline();

    if(snake.direction==UP)
           Up();
    else if(snake.direction==DOWN)
           Down();
    else if(snake.direction==LEFT)
           Left();
    else if(snake.direction==RIGHT)
           Right();

    QuitGame();


  }
  // kbhit() function checks whether a key as been pressed or not
  // Returns 0 otherwise
  while(!kbhit());
  // Takes input from the user
  Esckey=getch();

  // Checks if user pressed the esc key (ASCII value of esc = 27)
  if(Esckey==27){
    // Clears the output screen
    system("cls");
    // Exits the program. Game Ends
    exit(0);
  }



  // Keyboard Inputs
  // Control statements to ensure the snake is moving in the correct
direction
  key=getch();
  if((key==RIGHT && snake.direction!=LEFT&&snake.direction!=RIGHT)||
  (key==LEFT && snake.direction!=RIGHT &&snake.direction!=LEFT)||(key==UP
&& snake.direction!=DOWN && snake.direction!=UP)||(key==DOWN &&
snake.direction!=UP && snake.direction!=DOWN))
    {
```

```
        bend_no++;
        bend[bend_no]=snake;
        snake.direction=key;

        if(key==UP)
            snake.Y_Coord--;
        if(key==DOWN)
            snake.Y_Coord++;
        if(key==LEFT)
            snake.X_Coord--;
        if(key==RIGHT)
            snake.X_Coord++;

        // Recursion used to continue taking keyboard inputs
        Manoeuvre();
    }
    // Exit the program if the escape key is pressed
    else if(key==27){
      int i,j;
        gotoxy(38,14);


        printf(" LOADING....GAME OVER");
        gotoxy(38,15);
        for(i=1;i<=20;i++){
           for(j=0;j<=200000000;j++);

           printf("%c",219);
           }
        gotoxy(38,16);
        printf("THANK YOU FOR PLAYING :)\n");
        printf("\n\t\t\tPress any key to exit game");
        getch();

        system("cls");
        exit(0);
    }
    // Recursion used to continue taking keyboard inputs
    else{
      printf("\a\a\a"); // Produces a bell sound indicating new life
      Manoeuvre();
    }

}

// gotoxy() function used to place cursor on desired location on screen

void gotoxy(int x,int y){
```

```
  // COORD structure holds the x and y coordinates of a screen location
  COORD coord;
  coord.X=x;
  coord.Y=y;
  //The SetConsoleCursorPosition function sets the cursor position in the
specified console screen buffer.
  //Handles returned by GetStdHandle can be used by applications that need
to read from or write to the console
  SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),coord);

}

void GotoXY(int x,int y){
  HANDLE var;
  COORD  var2;
  // The fflush() function is used to clear the output buffer and move the
buffered data to console
  fflush(stdout);
  var2.X=x;
  var2.Y=y;
  var=GetStdHandle(STD_OUTPUT_HANDLE);
  SetConsoleCursorPosition(var,var2);
}



// Upward Movement of the Snake - Up() Function
// y-cooridnate of the snake head is altered for up-down movements

void Up(){
  int i;
  for(i=0;i<=(bend[bend_no].Y_Coord-snake.Y_Coord) && len<snake_len; i++){
    // Creating the Snake Body
    GotoXY(snake.X_Coord,snake.Y_Coord+i);
    {
      if(len==0)
        printf("^");
      else
        printf("@");

    }
    snakeBody[len].X_Coord=snake.X_Coord;
    // y-coordinate of the snake head incremented
    // x- coordinate remains the same
    // Snake body moves upward as a result
    snakeBody[len].Y_Coord=snake.Y_Coord+i;
    len++;

  }
```

```
  Bend();
  if(!kbhit())
  // y-coordinate of the snake head decremented if no key pressed further
     snake.Y_Coord--;
}

// Downward Movement of the Snake - Down() Function
// y-cooridnate of the snake head is altered for up-down movements

void Down(){
  int i;
  for(i=0;i<=(snake.Y_Coord-bend[bend_no].Y_Coord) && len<snake_len;i++){
    GotoXY(snake.X_Coord,snake.Y_Coord-i);
    {
      if(len==0)
        printf("v");
      else
        printf("@");
    }

    // y- coordinate of the snake head decremented
    // x- coordinate of the snake head remains the same
    // Snake body moves downward as a result
    snakeBody[len].X_Coord=snake.X_Coord;
    snakeBody[len].Y_Coord=snake.Y_Coord-i;
    len++;



  }
  Bend();
  if(!kbhit())
  // y-coordinate of the snake head incremented if no key pressed further
      snake.Y_Coord++;
}

// Leftward Movement of the Snake - Left() Function
// x-coordinate of the snake head is altered for left-right movement

void Left(){
  int i;
  for(i=0;i<=(bend[bend_no].X_Coord-snake.X_Coord) && len<snake_len;i++){
    GotoXY((snake.X_Coord+i),snake.Y_Coord);
    {
      if(len==0)
         printf("<");
      else
         printf("@");
```

```
    }
    // x-coordinate of the snake head incremented
    // y-coordinate of the snake head remains the same
    // Snake body moves leftwards as a result
    snakeBody[len].X_Coord=snake.X_Coord+i;
    snakeBody[len].Y_Coord=snake.Y_Coord;
    len++;
  }
  Bend();
  if(!kbhit())
  // x-coordinate of the snake head decremented if no key pressed further
     snake.X_Coord--;
}

// Rightward Movement of the Snake - Right() Function
// x-coordinate of the snake head is altered for left-right

void Right(){
  int i;
  for(i=0;i<=(snake.X_Coord-bend[bend_no].X_Coord) && len<snake_len;i++){

    // x-coordinate of the snake head is decremented
    // y-coordinate of the snake head remains same
    // Snake body moves rightwards as a result
    snakeBody[len].X_Coord=snake.X_Coord-i;
    snakeBody[len].Y_Coord=snake.Y_Coord;




   GotoXY(snakeBody[len].X_Coord,snakeBody[len].Y_Coord);{
      if(len==0)
          printf(">");
      else
          printf("@");
    }
    len++;

  }
  Bend();
  if(!kbhit())
   // x-coordinate of the snake head incremented if no key pressed further
      snake.X_Coord++;

}

// Bend() Function includes the logic behind the movement of the snake
// In UP, DOWN, LEFT, RIGHT directions
```

```
void Bend(){

  int i,j,turn;
  for(i=bend_no;i>=0 && len<snake_len;i--){

    // If x-coordinate is same during movement
    // Movement in Up/Down direction

    if(bend[i].X_Coord==bend[i-1].X_Coord){
      turn=bend[i].Y_Coord-bend[i-1].Y_Coord;

      // Difference between adjacent y-coordinates during movment
      // If difference < 0 then movement upwards

      if(turn<0){
        for(j=0;j<=abs(turn);j++){
          snakeBody[len].X_Coord=bend[i].X_Coord;

          // y-coordinate of the snake body incremented
          // Snake moves upwards
          snakeBody[len].Y_Coord=bend[i].Y_Coord+j;
          GotoXY(snakeBody[len].X_Coord,snakeBody[len].Y_Coord);
          // Snake body added in the upward dirrection
          printf("@");
          len++;

          if(len==snake_len)
             break;
        }
      }

      // If differecne > 0 then movement is downwards

      else if(turn>0)

          for(j=1;j<=turn;j++){
            snakeBody[len].X_Coord=bend[i].X_Coord;
            // y-coordinate of the snake body decremented
            snakeBody[len].Y_Coord=bend[i].Y_Coord-j;
            GotoXY(snakeBody[len].X_Coord,snakeBody[len].Y_Coord);
            printf("@");
            len++;

            if(len==snake_len)
               break;
          }
      }
```

```
        // If y-coordinate during the snake movement remains the same
        // Snake moves in Left/Right directions


        else if(bend[i].Y_Coord==bend[i-1].Y_Coord){

          turn=bend[i].X_Coord-bend[i-1].X_Coord;

          // Difference in the adjacent x-coordinates during snake movement
          // If difference < 0 then snake movement towards right
          if(turn<0)


            for(j=1;j<=abs(turn) && len<snake_len;j++){
              // x-coordiante of the snake body is incremented
              snakeBody[len].X_Coord=bend[i].X_Coord+j;
              snakeBody[len].Y_Coord=bend[i].Y_Coord;
              GotoXY(snakeBody[len].X_Coord,snakeBody[len].Y_Coord);
              printf("@");
              len++;

              if(len==snake_len)
                 break;

            }


          // if difference is >0 then snake movement towards left
          else if(turn>0)

             for(j=1;j<=turn && len<snake_len;j++){
               // x-coordinate of the snake body is decremented
               snakeBody[len].X_Coord=bend[i].X_Coord-j;
              snakeBody[len].Y_Coord=bend[i].Y_Coord;
              GotoXY(snakeBody[len].X_Coord,snakeBody[len].Y_Coord);
              printf("@");
              len++;

              if(len==snake_len)
                 break;

             }
        }

    }
}

// Function for score calculation
```

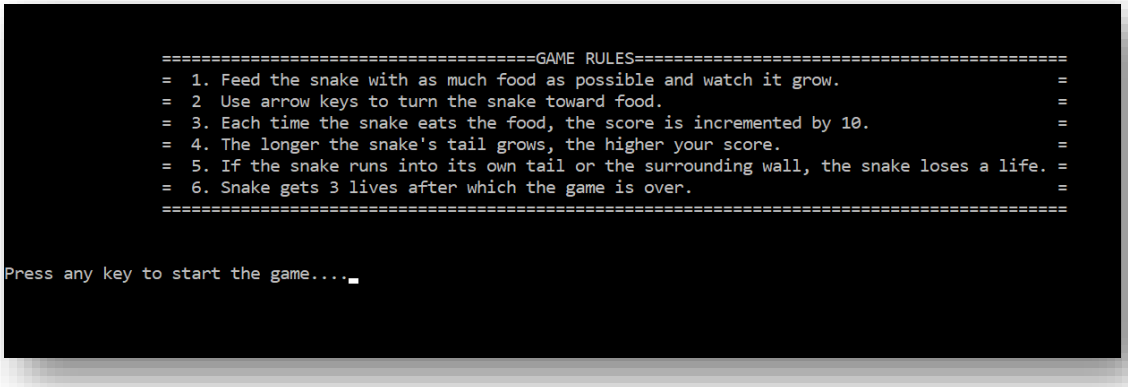
```c
// Displays Score & Lives on the screen
int Score_Life(){
  GotoXY(20,8);
  score=(snake_len-5)*10;
  printf("SCORE: %d",(snake_len-5)*10);
  score=(snake_len-5)*10;
  GotoXY(50,8);
  printf("LIVES: %d",Snake_life);
  return score;
}

int ScoreDisplay(){
  int score=Score_Life();
  system("cls");
  return score;

}

// Function to delay the gameplay
void GameLag(long double val){
  Score_Life();
  long double i;
  // Slows the game, making it more playable
  for(i=0;i<=(30000000);i++); }



// Game Over Function to display the final score
// Exit the game

void QuitGame(){
  int i;

  // Minimum size of the snake is 4

  for(i=4;i<snake_len;i++){
    // If the snake touches its body it loses a life.
    if(snakeBody[0].X_Coord==snakeBody[i].X_Coord &&
snakeBody[0].Y_Coord==snakeBody[i].Y_Coord){
      gameOver=true;
    }
    if(i==snake_len || gameOver!=false)
        break;
  }
```

```
  // If the snake exceeds the limit of the boundary, its life is
decremented
  if(snake.X_Coord<=10 || snake.X_Coord>=70 || snake.Y_Coord<=10 ||
snake.Y_Coord>=30 || gameOver != false){
    Snake_life--;
    // If lives of snakes are left, it starts afresh from its initial
position
    // Moving towards the right (East)
    if(Snake_life>=0){
      snake.X_Coord=25;
      snake.Y_Coord=20;
      bend_no=20;
      snake.direction=RIGHT;
      // Continues to take keyboard inputs for movements
      Manoeuvre();

    }
    // If the 3 lives have been used up, The game exits
    else{
      system("cls");
      printf("\n\t\t\t\t  You have used up all your lives\n");

      int i,j;
      gotoxy(38,14);

      // Loop to display the loading of the game to exit
      printf(" LOADING....GAME OVER");
      gotoxy(38,15);
      for(i=1;i<=20;i++){



     for(j=0;j<=200000000;j++);

         printf("%c",219); // ASCII Chharacter having value 219 is a
graphic block character
         }
      gotoxy(38,16);
      printf("THANK YOU FOR PLAYING :)\n");
      printf("\n\n\t\t\t\t  Press any key to exit game");

      getch();
      // Exits the program
      exit(0);
    }
  }
}
```

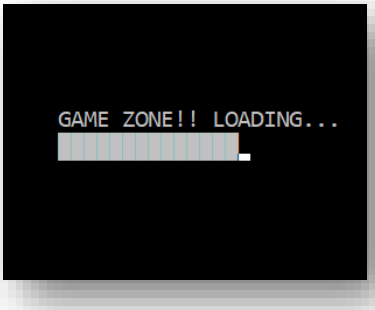
SCREENSHOTS:

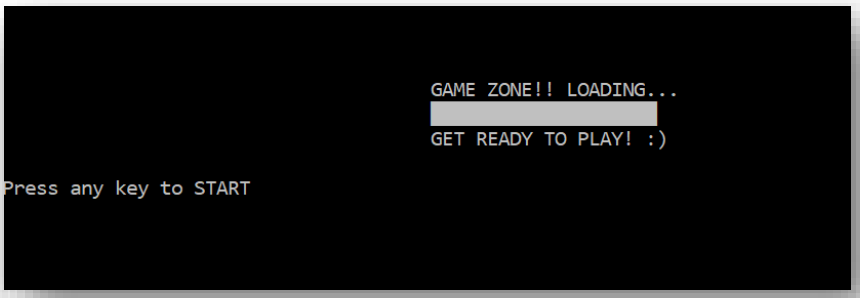
Main Menu of the Game:

Displaying Game Rules:

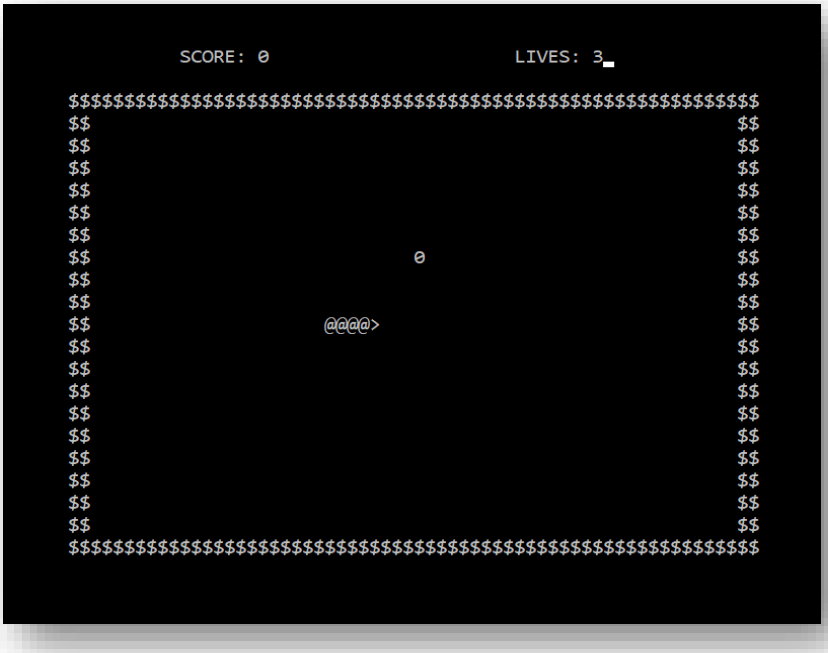


Loading Game:

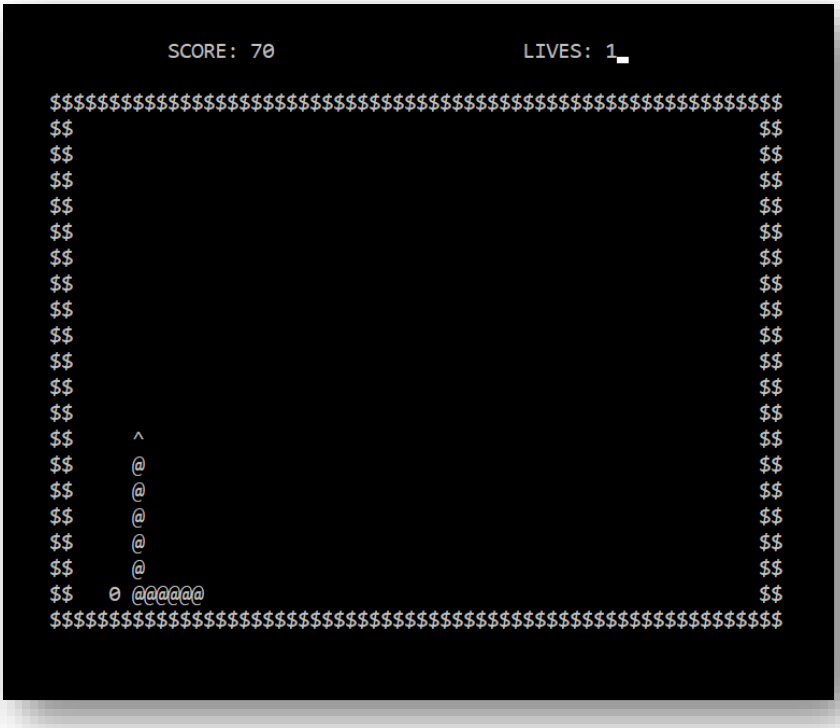




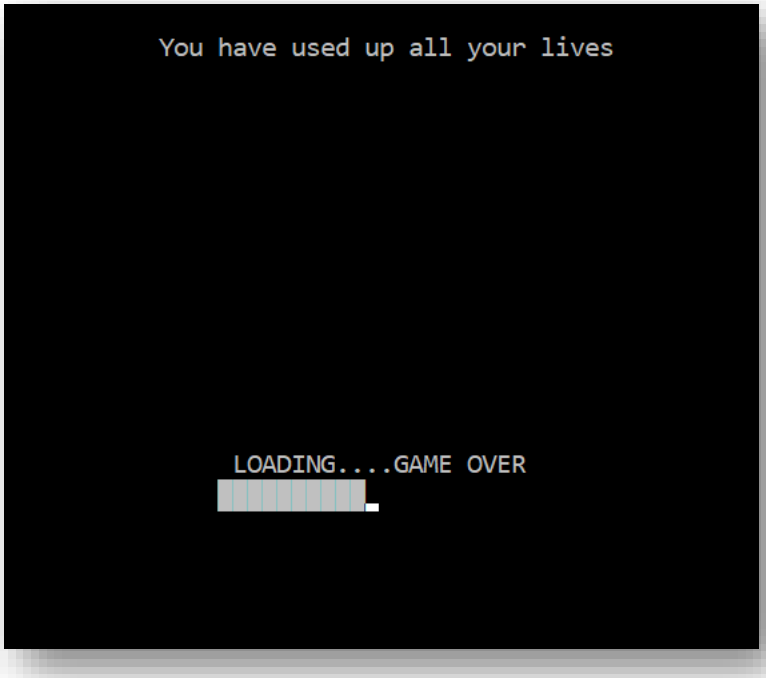
Snake Game Beginning (Score: 0 and Lives: 3)

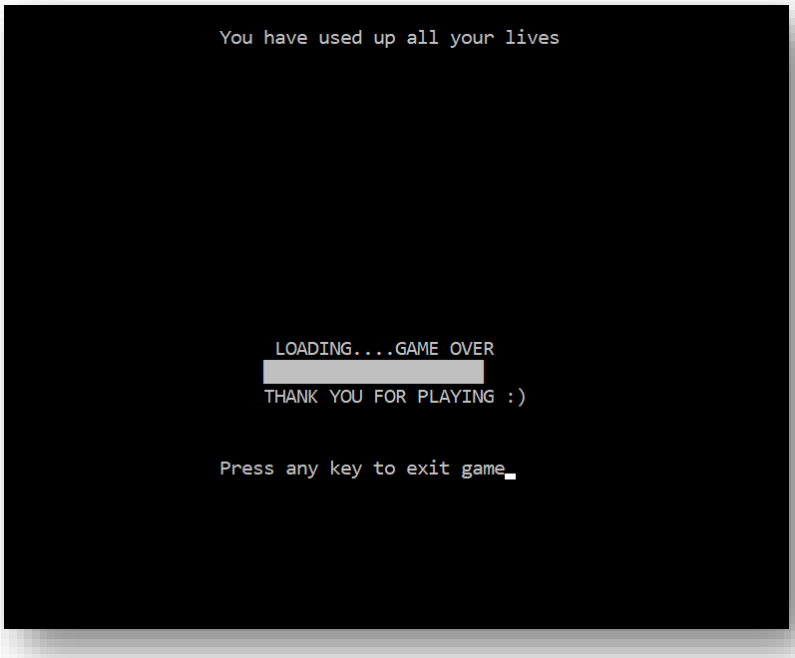


Incrementation of Snake Size & Deduction of Lives:



Game Over -> Loading Out & Exiting the Game:





RESULT:

Snake Game Program has been executed successfully.

CONCLUSION:

In conclusion, the code meets all the success criteria that were initially planned during the design stage. This has been made easier through problem decomposition that significantly highlighted the requirements for each stage in the development. The testing of each stage ensured that the program met all the success criteria. It was planned to be efficient and less memory intensive through declaring functions for each stage of development. This centralised the code as more organized through less lines of codes as calling function reduced repeating the same code again and again.

Every stage of the project was done through setting timelines that helped deliver the project on time and skip the stages that had unresolved issues. The document contains table of contents that illustrates the project as more organized and accessible for a user to go through the project report. Each stage of the code is well interpreted through algorithms and flowcharts and tested to convey the program is running successfully through presenting suitable screenshots of the Snake Game.

The source code is extensively annotated with comments to highlight the functionalities of different segments of the code. Moreover, the header files, function prototypes and global declarations have all been stacked together at the beginning of the source code making the code more organised and efficient to read. The future scope of this Snake Game Project is to incorporate various difficulty levels, multiplayer functionality and graphics providing visual representation of the objects in the game.