THE SNAKE GAME

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR INNOVATIVE WORK
UNDER

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by:

VARIN THAKUR & SIDDHANT KHANNA
(2K20/CO/475 & 2K20/CO/441)

Under the supervision of:

MRS. INDU SINGH
(ASSISTANT PROFESSOR, CSE)
Department of CSE, DTU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

11/ 21

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CANDIDATES' DECLARATION

We, Varin Thakur (2K20/CO/475) & Siddhant Khanna (2K20/CO/441), students of B.TECH (COE) declare that the Innovative Project Report titled "THE SNAKE GAME" which is submitted by us to the Department of Computer Science and Engineering, Delhi Technological University, Delhi, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma, Fellowship or other similar title or recognition.

Place: DTU, Delhi VARIN THAKUR & SIDDHANT KHANNA

Date: 13/11/2021 2K20/CO/475 & 2K20/CO/441

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CERTIFICATE

I, hereby certify that the Project titled "THE SNAKE GAME" submitted BY VARIN THAKUR & SIDDHANT KHANNA, Roll numbers: 2K20/CO/475 & 2K20/CO/441, Department of Computer Science and Engineering, Delhi Technological University, Delhi, as part of Innovative Work is a record of project work carried out by the student under my supervision. To the best of my knowledge, this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: DTU, Delhi
Date: 13/11/21

MRS. INDU SINGH
(Assistant Professor, CSE,DTU)
SUPERVISOR

ACKNOWLEDGEMENT

We are very thankful to Mrs. Indu Singh (Assistant Professor, Computer Science Eng. Dept.) and all the faculty members of the Computer Science Engineering Dept. of DTU. They all provided immense support and guidance for the completion of the project undertaken by us. It is with their supervision that this work came into existence.

We would also like to express our gratitude to the university for providing the laboratories, infrastructure, test facilities and environment which allowed us to work without any obstructions.

We would also like to appreciate the support provided by our lab assistants, seniors and peer group who aided us with all the knowledge they had regarding various topics.

VARIN THAKUR
B.TECH (CSE)
2K20/CO/475

SIDDHANT KHANNA
B.TECH (CSE)
2K20/CO/441

ABSTRACT

THE SNAKE GAME is a user-friendly game where a snake which is controlled by a player has to collect fruits over an area. As the Snake eats the fruit, the length of its tail increases which is marked by an increase in the score of the user by a constant factor. The Game ends or is over in the cases when the snake touches the walls or when the snake touches its own tail.

Our project aims to make this game using basic concepts of Object Oriented Programming in the Programming Language of C++.

CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

The Snake Game entails the use of a snake (its head and tail) to rover across a map of a custom height and width (set by the programmer) and eats fruits along its way. The Snake's head is denoted by a ‘BLUE’ character and parts of tail are depicted in the game by ‘PURPLE’ characters. The Fruit (denoted by a ‘RED’ character) is randomly placed across the map and the user through the input via a keyboard controls the direction of the movement of the snake to make it eat the fruit. As the snake eats the fruit, the length of the tail increases making the snake more vulnerable to eat itself instead of eating the fruit. The score escalates as by a non-changing factor. The game terminates when the Snake hits the outer boundaries denoted by 'YELLOW' characters or a part of its tail.

The same task is achieved using the object oriented approach in the language of C++. It involves the use of concepts like classes, objects, inheritance, data types, enumerations, variables, conditional statements, iterations and loops, functions and standard input and output operations.

Chapter 2

MAKING CLASSES AND MEMBER FUNCTIONS

Header files used in the code are: -

(i) #include<iostream.h> - to perform input output operations

(ii) #include<conio.h> - used to include functions _kbhit() and _getch()

(iii) #include<windows.h> - used to include the sleep() function.

(iv) #include<ctime> - used to seed the srand function with current time.

We have made 4 classes to implement this game which are as follows.

(i) GOD – Responsible for the controls of game_Over and score.

(ii) MAP – Responsible for the controls of height and width of the map.

(iii) SNAKE – Responsible for the head and tail of the snake.

(iv) FRUIT – Responsible for the locations of the fruit.

2.1 THE GOD CLASS

DATA MEMBERS are private.

- bool Game_Over – the game runs while the value of this variable is false and terminates otherwise.
- int score – records the score of the player.

MEMBER FUNCTIONS

- God() - Constructor to initialize Game_Over to false and score to 0.
- void cursorReset() - Resets the cursor position and makes it point to {0, 0}.
  void EndGame() - Ends the game by updating the value of Game_Over to true.
- void ScoreIncrease() - Increments the score when the snake eats the fruit by a factor of 10.
- void display_Score() - Displays the score of the player.
- bool get_Game_Over() - returns the value of the Game_Over variable.

2.2 THE MAP CLASS

DATA MEMBERS are public.

- int height – to store the height of the map.
- int width – to store the width of the map.

MEMBER FUNCTIONS

- Map() – Constructor to initialise height and width.
- void Draw_WALL() – Prints a wall character in “Yellow” colour.
- void Draw_EMPTY_SPACE() – Prints an empty space character in “Green” colour.

2.3 THE SNAKE CLASS

Inherits from MAP class publicly to access the height and width.

DATA MEMBERS are private.

- int head_x_coordinate - stores the x-coordinate of the snake’s head.
- int head_y_coordinate – stores the y-coordinate of the snake’s head.
- vector<int> tail_x_coordinate{100} – stores x-coordinates of the elements of snake’s tail.
- vector<int> tail_y_coordinate{100} - stores y-coordinates of the elements of snake’s tail.

- int length_tail – stores the length of snake's tail.
- enum Direction – a user defined data type, a variable of type Direction stores the direction of snake's movements. (Public)
- Direction dir – stores the direction of snake's movement.

MEMBER FUNCTIONS

- Snake () – a constructor to initialize the data members of the class.
- void Draw_HEAD () – used to draw the head of the snake. Prints a head character in "BLUE" colour.
- void Draw_TAIL () – used to draw the tail of the snake. Prints a tail character in "PURPLE" colour.
- void logic_MOVE () - Controls the movement of the snake using Direction variable dir. dir is used as the switch case variable and the movement gets decided by its value.
    - dir = LEFT – head_x_coordinate gets decremented thus the snake moves left.
    - dir = RIGHT – head_x_coordinate gets incremented thus snake moves right.
    - dir = UP – head_y_coordinate gets decremented thus snake moves up.
    - dir = DOWN – head_y_coordinate gets incremented thus snake moves down.
- void logic_TAIL() - It is used to set the x and y coordinates of the elements of the tail to that of previous one using prev_x and prev_y which store the x and y coordinates of the previous elements of the snakes tail respectively.
- void Increment_TAIL() - Increments the length of snake's tail when it encounters a fruit by incrementing the length_tail variable.
- int get_head_x_coordinate() - returns x coordinate of snake's head.
- int get_head_y_coordinate() - returns y coordinate of snake's head.
- int get_tail_length() - returns the length of snake's tail.
- vector <int>::iterator get_tailX() - returns the base address of the tail_x_coordinate vector.

- vector <int>::iterator get_tailY() - returns the base address of the tail_y_coordinate vector.
- void Input(God* &g) - The function updates the value of 'dir' variable depending on the key pressed and if 'x' is pressed, it calls Endgame() function of God class to end the game.
    - kbhit() – return a Boolean value, true if a key is pressed and false otherwise.
    - _getch() – returns the ASCII value of the key pressed.

2.4 THE FRUIT CLASS

This class also inherits from the map class publicly to access the height and width of the map.

DATA MEMBERS are private.

- int fruit_x_coordinate – to store the X coordinate of the fruit.
- int fruit_y_coordinate – to store the Y coordinate of the fruit.

MEMBER FUNCTIONS

- Fruit() – constructor to initialise coordinates of the fruit.
- void logic_FRUIT() – assigns random coordinates to the fruit.
- int get_fruit_x() – returns the X coordinate of the fruit.
- int get_fruit_y() – returns the Y coordinate of the fruit.

Chapter 3

DEFINING CONTROL FUNCTIONS

Now that we have made all the classes along with their corresponding member functions, lets look at the control functions.

We have used several user- defined functions for control.

(i) Setup()- Assigns memory for all objects and calls their respective constructors.

(ii) Draw()- Calls the draw functions from the required objects, when certain specific conditions arise.

(iii) Logic()- Calls the logic functions of snake as well as handles the conditions for terminating and incrementing the score of the player whenever required.

(iv) Main() – Calls the above functions repeatedly to implement the game.

3.1 SETUP(God** g, Snake** s, Map** m, Fruit** f)

This creates an object for each of the passed pointers and also calls their constructors.

God constructor - The score of the player at the beginning must be null or 0. This is done by assigning a 0 value to the score variable. The Boolean Variable of Game_Over should be set to the value of FALSE so that the game runs.

Map constructor – The height and width are given values of 50 and 20 respectively.

Snake constructor - The Position of the Head should be known which is achieved by giving values to the x_coordinate and y_coordinate variables respectively. The snake should be shortest in its length which is ensured by the variable 'Length_of_Tail' being

assigned the value of 0. Finally, the Snake should be static. The enumeration variable 'dir' is set to the value of STOP, waiting for a key press by the user.

Fruit Constructor - The Fruit should also be assigned a random position in the map. This has been done using the rand() function. The srand() function ensures that the same sequence of random numbers aren't generated at every compilation of the program.

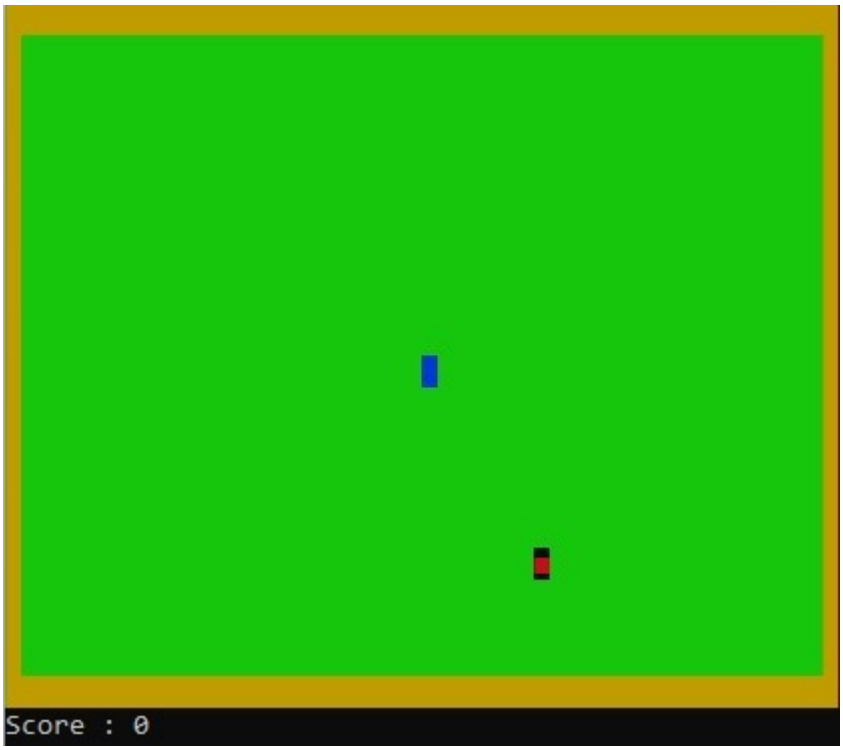

Fig 3.1

3.2 DRAW(God* g, Snake* s, Map* m, Fruit* f)

The Draw function is a function with a return type of void. It starts with using a member function from God class cursorReset() to reset the cursor at {0,0} so that the user experiences a continuous gameplay for successive iterations of the program. Further, it proceeds in printing the elements of the game like the boundaries, head, fruit, etc. using the concepts of nested for loops.

The first for loop repetitively prints 'YELLOW' charachter width + 2 times to make the upper wall of the map. This is done by using the Draw_WALL function from the map

class. The 'cout' function with the keyword of 'endl' moves the cursor to the next line for further printing. The successive for loop which is a combination of three nested for loops can be visualised as the cursor printing the elements in a row wise fashion. First it checks if the column number is the first one, i.e., = 0 and prints a Wall to depict a part of the left wall. Then it checks if the row and column coordinates coincide with that of the head and prints the snake's head using the Draw_HEAD function from the snake's class. Further it inspects if the row and column coordinates are equal to that of the fruit and uses the Draw_FRUIT function from the fruit class to draw the fruit.

This is followed by the printing of the Tail and the Lower Wall. A Boolean variable print is used which helps in confirming that the printing of the tail has taken place. The third nested for loop ensures that the no. of tail characters printed are equal to the length of the tail. The tail is printed using the Draw_TAIL function from the snake class.

We have implemented these things using conditional statements (if and if else-if ladders). If none of the above conditions hold true, an empty space character is printed to depict the empty space in the map, using the Draw_EMPTY_SPACE function from the map class.

Finally, we put a condition to print the right wall of the map followed by the keyword 'endl' (out of the 2 nested for loops) which moves the cursor to the next line after the desired printing of the elements has taken place for a row.

After this we print the lower boundary of the map using another for loop running width+2 times calling the Draw_WALL function. When the lower boundary of the map is printed, we move to a new line using the same way mentioned above.

The function's body ends with printing the score of the player which is done by using the display_SCORE function from the God class.

3.3 LOGIC(God* g, Snake* s, Map* m, Fruit* f)

The logic function is now employed which is again having the return type of void. First, the logic_TAIL function is called from the snake class. To understand the logic of how the tail follows the head, lets us delve deeper in what the code actually says. Firstly we

store the value of the x and y coordinates of the first fragment of the tail (tail_x_coordinate[0] & tail_y_coordinate[0]) in the variables prev_x and prev_y respectively. Now the 0th coordinates of the tail (indexing of the array starts with 0 and hence a similar indexing for the tail coordinates) are updated to the coordinates of the head giving us the notion of a portion of the snake moving. We declare two more variables prev_2x and prev_2y.

Now comes the part where we move the rest of the tail coordinates. To achieve this, we use a for loop. The no of iterations of the loop depends on the length of the tail so that every fragment of the tail can move in the desired direction and follow the head. Before this loop we had moved the 0th coordinates of the tail to that of the head. Now we use the extra variables that we declared earlier. In the loop, we first assign the initial values of the tail coordinates starting from the 1st to the (length of the tail - 1)th to the declared variables of prev_2x and prev_2y respectively. Then we update the values of the tail coordinates using prev_x and prev_y. Finally, we update the values of prev_x and prev_y to make them the desired coordinates for the next iteration.

This can be better envisioned with the help of an example of an iteration. Suppose the length of the tail is 3 (Length_of_Tail = 3) and the the coordinates of head, tail 0, tail 1 and tail 2 are (2,1), (3,1), (4,1) and (5,1) respectively. If the Snake moves left, the coordinates would have to be shifted to (1,1), (2,1), (3,1) and (4,1) resepctively.

Before the for loop being executed, we store the value of the initial coordinates of the tail 0 i.e. (3,1) into prev_x and prev_y respectively. This implies that prev_x stores 3 and prev_y stores the value of 1. Now we update the values of the tail 0 to that of the head. This makes the tail 0 X as 2 and tail 0 Y as 1. Thereafter the loop is executed. As the loop starts running from tail 1, we store the initial 1st coordinates of the tail into prev_2x and prev_2y respectively. This means that prev_2x and prev_2y store the value of 4 and 1 respectively. Now the values of tail 1 are updated to the earlier coordinates of tail 0. This is done by assigning the values of prev_x and prev_y to tail 1. Tail 1 now becomes (3,1) which was earlier tail 0. Finally, the prev_x and prev_y values are updated by prev_2x and prev_2y making them store the earlier coordinates of tail 1 i.e. (4,1) so that the tail 2 coordinates can be replaced by the tail 1 coordinates in the next iteration. In the same

way, we shift all the coordinates successively and therefore we see the snake moving in a direction.

Then, the logic_MOVE function is called from the snake class. On the basis of the input given by the user the Input function will give a value to the variable ‘dir’ (UP, DOWN, LEFT, RIGHT) and on the basis of that value the direction of the snake's movement will be decided. The switch case statement will compare the value of ‘dir’ which we got from the input function to the value LEFT, RIGHT, UP or DOWN and if a match is found then that case is executed.

If the value of the ‘dir’ is

(i) LEFT then the x coordinate of the snake’s head will be decremented by 1.

(ii) RIGHT then the x coordinate of the snake’s head will be incremented by 1.

(iii) UP then the y coordinate of the snake’s head will be decremented by 1.

(iv) DOWN then the y coordinate of the snake’s head will be incremented by 1.

(v) For any other input we will exit the switch case.

The game ends when the head of the snake touches the wall. When the x-coordinate of the head of the snake is greater than width or it is less than 0 or if the y coordinate of the head of the snake is greater than height or less than 0 the game terminates due to the value of the Boolean variable Game_Over becoming true. When the head of the snake touches its tail then also the game should end so if at any point the x and y coordinate of the head of the snake become equal to the x and y coordinate of any element of the tail the value of the Boolean variable Game_Over becomes true and the game ends. The EndGame function from the god class takes care of this. When the snake eats the fruit, the length of the snake's tail increases by 1, the score increases by 10 and the fruit is relocated to a new position. This is marked by the situation when the x and y coordinate of the head and the tail coincide with the x and y coordinate of the position of the fruit. Hence the scoreIncrease, logic_FRUIT increment_TAIL functions are called from the respective classes. As mentioned, we give a new position to the fruit by using the rand function. The modulo operator ensures that the coordinates of the position of the fruit come in the range of [0, width] for the x-coordinate and [0, height-1] for the y coordinate.

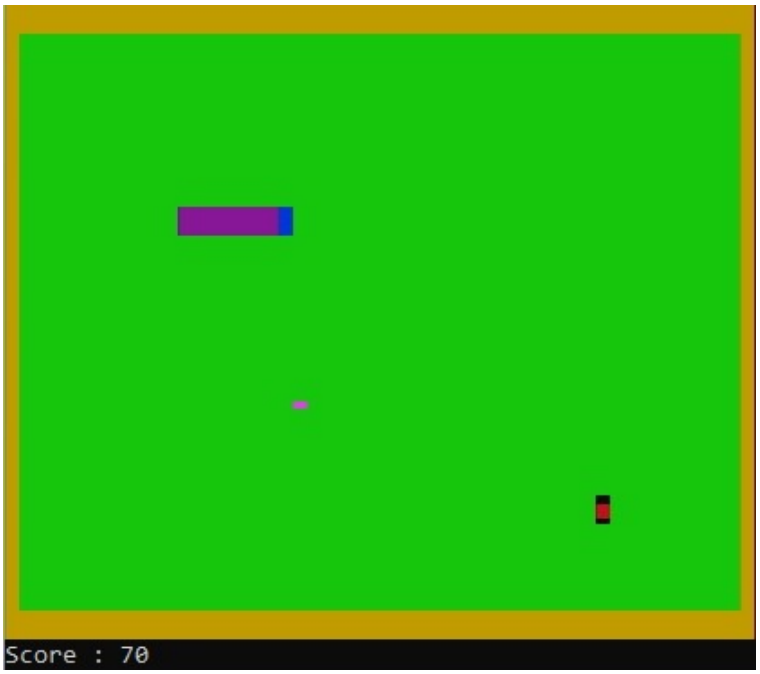
Score : 70

Fig 3.2

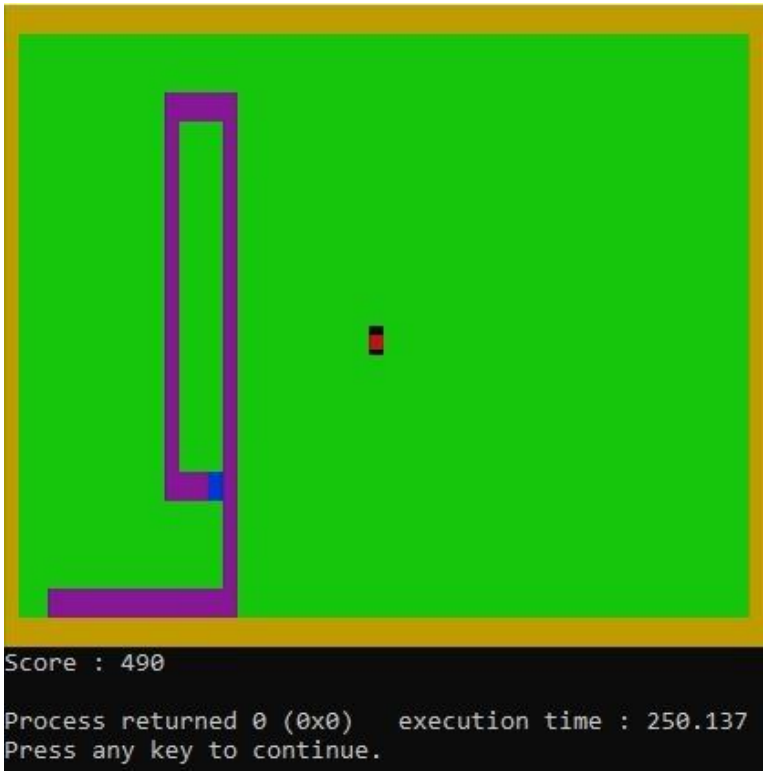
Score : 490
Process returned 0 (0x0) execution time : 250.137
Press any key to continue.

Fig 3.3

3.4 MAIN()-

In the main function, first class pointers are created. (God *g, Map *m, Snake*s , Fruit *f).

Setup Function is called to initialise the class objects with their respective constructors. Now using a while loop, till the Game_Over is not true, Draw function, Input function from the snake class and Logic function are called. The Sleep function is used to control the speed of the game. The output of this loop is the game which we see in the terminal window!

3.5 COLOR(int x)-

A console window comprises of buffers. A buffer is a 2-D Array which stores the the text character and its colour attribute. Buffers of a console window can be modified by handles. We use the SetConsoleTextAttribute function from the windows.h header file to set the text attribute with an integer which corresponds to a specific colour as shown below.

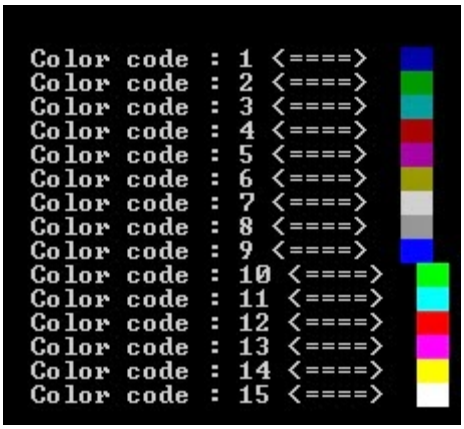

Fig 3.4

Chapter 4

GETTING INPUT FROM THE PLAYER

The input function from the snake class is a void function having no arguments, a function having a void return type does not return a value. The role of the input function is to take input- 'W' /'A' /'S' / 'D' from the user to decide the direction of the snake. We have used two functions, both in the header file “conio.h”.

(i) _kbhit() - this function returns a non-zero value if a character is pressed and returns zero if no character is pressed.

(ii) _getch() - this function returns the ASCII value of the character pressed.

So, if the user gives us an input then the statements inside the if statement will be executed else the statements will not be executed. Using switch case, a comparison is made between the ASCII (ASCII, abbreviated from American Standard Code for Information Interchange, is a character encoding standard for electronic communication. ASCII codes represent text in computers, telecommunications and other devices. ASCII is a character encoding that uses numeric codes to represent characters. These include upper and lowercase English letters, numbers and punctuation symbols.) values of the input and the ASCII values of 'a', 'd', 'w', 's', 'x'. If both values are same then that case will be executed. If the ascii value of the input is same as that of

(i) 'a' (97) the value of ‘dir’ becomes LEFT

(ii) 'w' (119) the value of the ‘dir’ becomes UP

(iii) 's' (115) the value of ‘dir’ becomes DOWN

(iv) 'd' (100) the value of ‘dir’ becomes RIGHT

(v) 'x' (120) the value of the Boolean variable Game_Over becomes true and the game ends.

SCOPE OF IMPROVEMENT

- IMPROVING GRAPHICS OF THE GAME

  While depicting things with colours of a same character is easier to implement, the players of a game love accurate representations of the items. Hence, using shapes, we can improve the game.

- DATABASE OF PLAYER'S SCORE

  We can also look for storing the player's score in a database, so that the player can know his/her personal best and can also compete with other players.

BIBLIOGRAHY

[1] www.youtube.com

[2] www.geeksforgeeks.org

[3] www.stackoverflow.com

[4] www.journaldev.com