Final Experiment
Snake Game Using Arduino

I. Experiment Objective

THE goal of this experiment is to program a game of snake on an Arduino UNO board using a C program.

II. Overview

Free Form snake game was very popular since the introduction of mobile gaming systems. So, we decided to build this game using an Arduino UNO and a LED matrix. The project involves both hardware and software, which makes it ideal for an embedded system design project. The classic game of 'Snake' was made popular through its inclusion in early cell phones. At the beginning the player starts with a small snake. As the snake moves around, he must eat 'fruits' to grow the snake in length. The player must make sure that the snake doesn't hit its own body, otherwise game is over. There are various variations of the game. In some instances, the snake is allowed to travel through the field wall and emerge from the opposite wall, while in others, the snake has to guide though mazes to find its way. In our case, the snake is allowed to move through the walls but can't hit its own body. Eating 1 fruit will increase the score by 1. After game is over, the score is displayed on the LED matrix.

A. Materials

We used the following materials to perform our experiment:
Arduino UNO board
Push Buttons
Breadboard
64 LED Matrix Display
Resistors

III. Implementation

We used a 64 LED dot matrix to display the game and an Arduino UNO to program it. A C program was written which is compiled and uploaded to the Arduino board using Arduino IDE. The LED matrix is initialized and implemented using the respective library. Three buttons are used: Up, Left and Reset. Here is a figure showing how the connections are made:

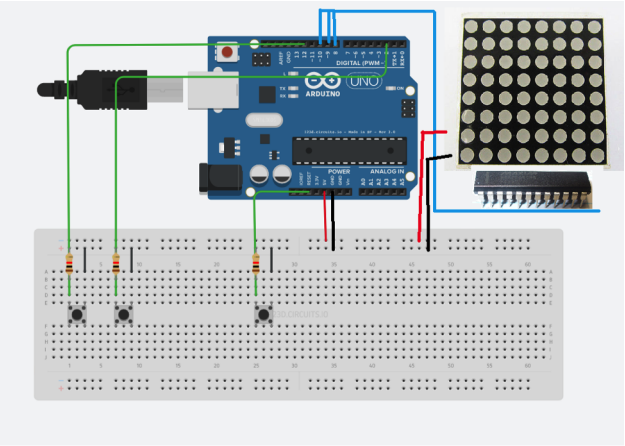

Fig. 1: Connections between Breadboard, Arduino and LED Matrix

A. Procedure

We wrote a C code to run the game in the Arduino IDE. The code consists of the all the algorithms required to run the snake game. The snake game starts with a unit length snake and a randomly generated fruit displayed on the LED board. The snake is controlled by two push buttons. Each time a player is able to eat the fruit, the length of the snake increases by one. This process continues each time the player eats the fruit and thus the length keeps growing. The snake dies once its head touches the tail of its own. A 20 by 8 a matrix array is used to control the length LED of the snake. Once the snake dies, the LED display shows the score and stops. After getting the score, we use a push button to reset the game and the game starts from beginning with a randomly generated snake and a fruit.

1) Initialization: Before we write any function, we need to initialize necessary variables for the whole program. Here is the code of Initialization part:

```
#include "LedControl.h"

int DIN = 8;
int CS = 9;
int CLK = 10;
LedControl lc = LedControl(DIN,CLK,CS,1);

byte numbers [20][8]  // Holds the necessary
    BCD code to display scores. Because of
    length of the code it is not shown here.

const int UP = 1;
const int DOWN = 2;
//const int RIGHT = 3;
const int LEFT = 4;
```

```
int snakeLength = 1;
int snakeX[25];
int snakeY[25];
int directions = 0;
int fruitX = 0;
int fruitY = 0;
int score=-1;
```

Here, we defined necessary pins of inputs and outputs for the LED matrix. We also defined necessary variables to keep track of the progress of the game and state of snake. Using these variables next, we will write our algorithm of the game. "LedControl.h" is the library to control behavior of the LED matrix.

2) Setup: This is the function that is being run only once when the program runs first time on the board. So this deals with all the setup and connections before the main algorithm takes place.

```
void setup(){
Serial.begin(9600);

lc.shutdown(0,false);
lc.setIntensity(0, 1);
lc.clearDisplay(0);

pinMode(2, INPUT);
//pinMode(3, INPUT);
pinMode(12, INPUT);
//pinMode(13, INPUT);
}
```

Here we can see that, we setup the LED matrix using function from the library. We also assigned input pins for the buttons.

3) Main Loop: This is the function which being run over and over. So this function will hold all the necessary functions which we want to be executed on each iteration of the loop.

```
void loop(){
  delay(150);
  checkButtonPress();
  draw();
  nextStep();
}
```

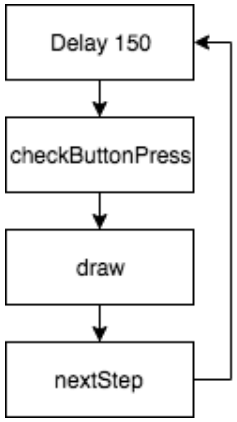

Fig. 2: Flowchart of main loop

4) CheckButtonPress Subroutine: This function checks which button has been pressed and save the result using variables declared in the initialization step. Here is the function:

```
void checkButtonPress(){
  if (digitalRead(2)==0){
    directions = UP;
  }

  else if (digitalRead(12)==0){
    directions = LEFT;
  }
}
```

As we use only two buttons, so we check two conditions.

5) Draw Subroutine: This subroutine draws the snake and fruit in the LED matrix.

```
void draw(){
  lc.clearDisplay(0);
  drawSnake();
  drawFruit();
}

void drawSnake(){

  for (int i=0; i<snakeLength; i++){
    lc.setLed(0,snakeX[i], snakeY[i], true);

  }
}

void drawFruit(){
  if (insidePlayField(fruitX, fruitY)){
    lc.setLed(0, fruitX, fruitY, true);
  }
}
boolean insidePlayField(int x, int y){
  return (x>=0) && (x<8) && (y>=0) && (y<8);
}
```

As we can see, this subroutine first clears the LED board to get rid of the previous draw. Then depending on the game state it draws the snake and fruit. The coordinate of the snake is saved in snakeX[] and snakeY[] arrays. Fruit coordinate can be found in fruitX and fruitY. The flowchart is shown below:

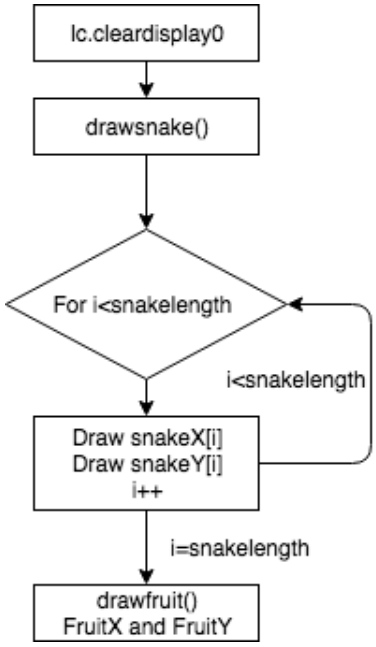

Fig. 3: Flowchart of CheckButtonPress

6) Nextstep Subroutine: The nextStep subroutine records the game state for the next iteration of the main loop. It updates the position of the snake depending on its position and last button pressed. Generates new fruit and updates scores if necessary. And if the game over condition is met, it executes the gameOver subroutine which displays the final score.

```
void nextStep(){
   if (directions == UP){           // 2
    snakeX[0] = snakeX[0]-1;
   }
   else if (directions == DOWN){    // Not
    Used anymore
    snakeX[0] = snakeX[0]+1;
   }
   else if (directions == LEFT){    //12
    snakeY[0] = snakeY[0]-1;
   }

    if (snakeX[0]<0){
      snakeX[0] = 7;
    }
    else if (snakeX[0]>7){
      snakeX[0] = 0;
    }

    if (snakeY[0]<0){
      snakeY[0] = 7;
    }
    else if (snakeX[0]>7){
      snakeY[0] = 0;
    }

    updateSnakeLength();
}


void updateSnakeLength(){

  if ((snakeX[0] == fruitX) && (snakeY[0] ==
    fruitY))
    {
      snakeLength++;
      createFruit();
      score++;

    }

      for (int i=snakeLength; i>0; i--){
      snakeX[i] = snakeX[i-1];
      snakeY[i] = snakeY[i-1];

      if (snakeX[0] !=0){
      if (snakeX[0] == snakeX[9]){
        gameOver();
      }
      }
      else if (snakeY[0] !=0){
        if (snakeY[0] == snakeY[9]){
          gameOver();
        }
      }

      }
}

void gameOver(){
```

Following this a small section of code is responsible for displaying the end score.

```
   for (int i=0; i<8; i++){
    lc.setRow(0, i, numbers[score][i]);
   }
   while (1){};
}
```

The flowchart of nextStep subroutine is given below:

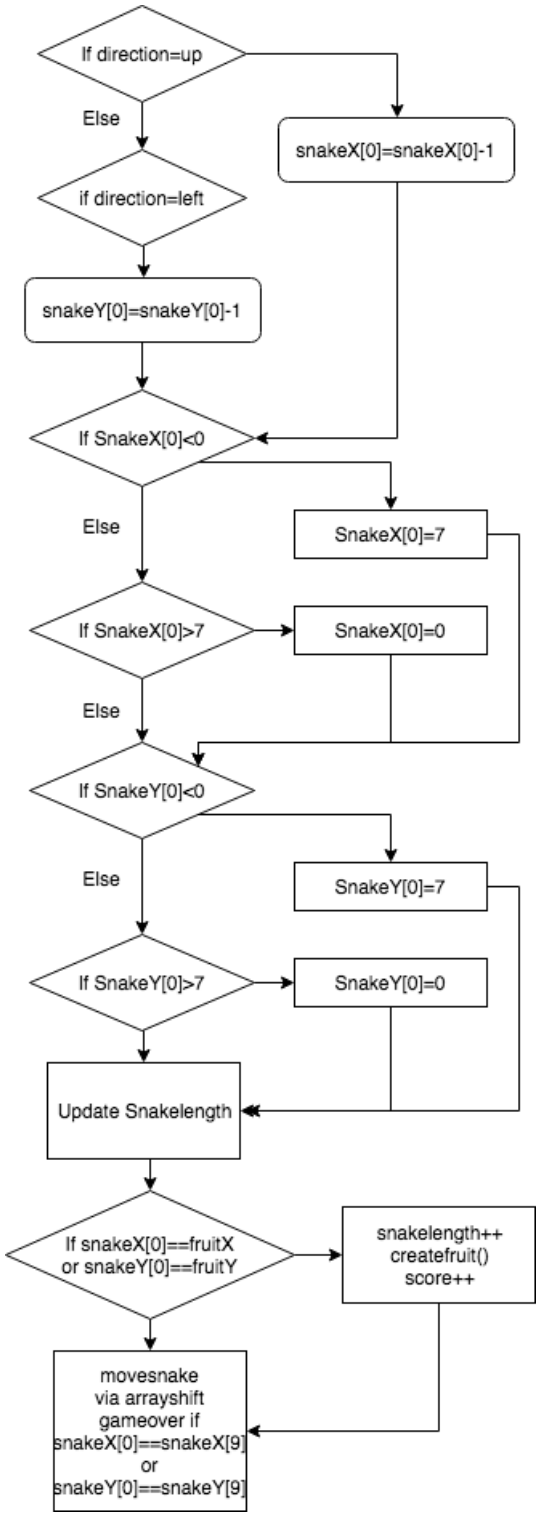

Fig. 4: Flowchart of nextStep

Here is a picture of the whole setup while the game is being run:

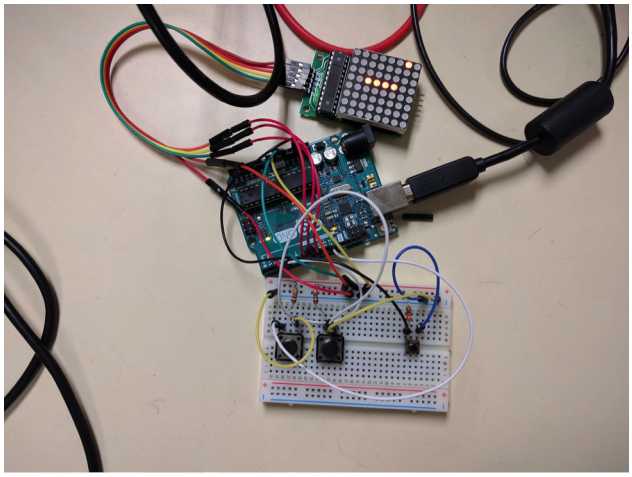

Fig. 5: Snake game being run on Arduino and LED board

IV. Conclusion

The goals of the experiment were reached. We were able to successfully implement the circuit using the breadboard and program code to run the game. The code allowed to a basic game of snake to be played, allowing the player to move up or left while collecting fruit. No errors were found. A basic system of showing the end score was implemented. Improvements can be achieved by using another 64 bit LED matrix to allow for a larger play area. Furthermore a system of moving all directions and also incorporating sound could also be possible improvements.