IMPLEMENTATION OF STACKER ARCADE USING ARDUINO

Anurag Yadav
Electronics and Communication Engineering,
Indian Institute of Technology Guwahati
Assam, 781039
y.anurag@iitg.ac.in
anurag.yadav@alumni.iitg.ac.in
anuragfzd62@gmail.com

Brajendra Suman
Electronics and Communication Engineering,
Indian Institute of Technology Guwahati
Assam, 781039
s.brajendra@iitg.ac.in

Abstract:

In this project we implement Stacker Arcade Game using Arduino. We made minimal use of hardware and used only one button to take input. We used modular approach to program algorithms to the Arduino. In the end we were able to successfully implement the game with features like level selection and score display at the end of the game. This project is a demonstration of easy, wide and diverse application of microcontrollers like Arduino which can be easily programmed for such purposes.

1. INTRODUCTION

Stacker Stacker Arcade is a popular game known to test players focus, precision and reaction time. In this game user tries to increase the height of a stack of LEDs by pausing an array of moving LEDs right above the stack already built. User gives input with the help of an input device like a joystick or push button.

In this project we use Arduino Uno as microcontroller. It is an open-source electronic platform which provides both the hardware which contains all basic input output ports, processor, clock, RAM etc. Software which is very similar to C programming language makes programming the board easy.

MAX7219 is a led dot matrix which has 8x8 LEDs that can be used to display patterns or text on it. These 8x8 blocks can be cascaded further to make the display screen even larger. A drives controls which LED of the matrix to light-up. Various patterns can be achieved by lighting up various LEDs. To make specific patterns multiplexing is used.

Various LEDs of the pattern take turn to go on and off. But this process happens so fast that to human eyes this appears as permanent on pattern.

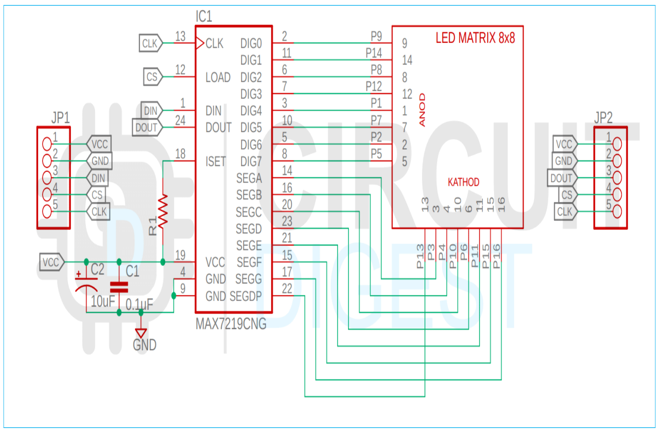

Figure 1: Schematic of MAX7219 8x8 LED dot matrix

Push button used in this project is a simple switch in which one can press the button to complete a circuit but circuit is broken as soon as one removes fingers from the button. This button is useful when one has to send a single bit information repeatedly.

2. MATERIAL NEEDED

1. Arduino Uno: to program the game.
2. Breadboard: as a board to connect various devices together.
3. Jumper wires: for connections.
4. Buzzer: to indicate button push, game end etc.
5. Push button: to take user input.
6. MAX7219 LED dot matrix: as display for the game.

3. ASSEMBLING HARDWARE

MAX7219 LED dot matrix has following five terminals:

1. Vcc: Connected to supply of +5V
2. GND: Connected to ground
3. DIN: Data In pin connected to digital pin 11 of Arduino.
4. CS: Chip select, also known as load, connected to pin 3 of Arduino.
5. CLK: Clock Pin connected to pin 13 of Arduino.

We connect one terminal of the pushbutton to +5V and another terminal to pin2 of Arduino. Also, we use a pull-down resistor to keep the digital pin as low unless button is pressed. Buzzer is connected to pin7 via a resistor and to ground line. To run the game, we have to supply Arduino with dc power for which we are using a 9 Volts battery.

4. SPECIFICATIONS & FEATURES OF GAME

After assembling the hardware, we made defined exactly what features and specifications we were going to have in the game. Following are details of the same.

I. In the beginning user can select level of game, which decides difficulty of the game.
II. For selecting the level user can press the key at desired level from a toggling menu.
III. After selecting game shall start.
IV. After game end user must be able to see their score, high score and level they were playing at.

5. DESIGNING ALGORITHM

Having decided features and specifications for our game we moved on to develop an algorithm that could implement these functionalities. We followed a top-down approach to develop our algorithm. That is, first we decided in what order what functions should be performed without worrying how that function should be performed. Then, we proceeded to define how those specific function needs to be performed. It took several iterations to make a fully functional algorithm. In each iteration we made algorithm more detailed delving into intricate details of maintaining various states and how-to of various functional entities. While, we designed the algorithm before programming the Arduino board there were some factors that we could not take into account beforehand and had to revisit and improve the algorithms once we noticed the faults after programming the Arduino.

In designing first, we decided the flow at topmost level. It was like this: first we should be able to select the level of the game, then play the game and then see the score. In next iteration we decided that a toggling menu should be there to select our level and that we need to be watchful for input through pushbutton to select appropriate level. Then in the game portion, there should be a running array of led, which on button press should stop and according to its alignment stack should grow, and once game is over score, high-score and level should be printed in binary.

In further iterations more details were added giving us a well-defined algorithm. We describe the final algorithm in appropriate section of this report.

6. PROGRAMMING ARDUINO

Having the algorithm ready our next task was to program it to the Arduino board to get final working game. To make the process easy to proceed and easy to code, improve and debug we broke programming task in multiple steps.

First, we wrote a program using led control library of Arduino to light the desired LED on MAX7219 LED dot matrix by giving input coordinates. The library used takes device number, x and y coordinate as input to light a LED. With little algebraic manipulations we could simplify the problem to giving x and y coordinates as input and treating four blocks of LED matrix as a single device with contagious coordinates.

Next, we build a running array of on-LEDs. To do this we looped a variable and set y-coordinates around that ON. Variable keeps incrementing but once it reaches a particular value it starts decreasing. This gives an appearance as if a led array is moving from one side to other side and bumping off the corners.
An interrupt function was declared to detect if button was pressed.
A variable to maintain x-coordinate was declared which incremented whenever a button press was detected x-coordinate improved by one. This made the moving array go one level up on the screen.
Next, we wrote a function to keep those LEDs on where LED array was when button was pressed. After this, a function to check alignment of LEDs, when player presses the button, was written.
This function compares stopping y-coordinate of the current array with that of previous level stopping array.
Based on the alignment we can end the game or decide the size of array at next level.
This completed the core of the game. So, next we wrote code for selecting level. To achieve this, we made a variable change from 0 to 7 waiting for 1 second at each value.

Whenever the button was pressed pointer came out of the loop and value of variable decided the level of the game selected. To realise change in the game we made speed of moving arrays linearly dependent on level.

This was done by changing the delay in the loop used in moving array function

7. DEBUGGING & IMPROVING

We could not foresee some of the problems that could arise in writing the code and thereafter. Once we had written the code, we found there were many problems in it. We were getting unexpected delays and glitches introduced by delays. Also, sometimes arrays sizes that move across the screen were not as intended due to miscalculations. We also face an error of multiple input being recorded at once due to imperfection and hardware limitation of pushbutton. We were able to debug and rectify these problems by making appropriate changes in algorithm and code thereby getting final and desirable results. This was obtained in multiple iterations of the above process.

8. ALGORITHM

Algorithm

Keep listening for buttonPress:

```
1. If button is pressed set buttonPressed=true
```

Algorithm in loop:

```
1. Call the function set_level().
2. Call the function game().
3. Call the function status_display().
4. Repeat from step 1.
```

Algorithm for set_level():

```
1. Set integer variable i to 0.
2. While i is less than 32 and buttonPressed is false, do the following:
  a. For integer variable j between 1 and 6, do the following:
    i. For integer variable k between 0 and 2, do the following:
      1. Set the led at position (i/8, j, 7-((i+k)%8)) to true using lc.setLed().
  b. Delay for 1000 milliseconds.
  c. Increment i by 4.
3. If buttonPressed is true, set game_level to i/4.
4. Else if game_level is less than 8, increment game_level by 1.
5. Else set game_level to 8.
6. Set buttonPressed to false.
7. Set DELAY_MS to 450 minus game_level multiplied by 50.
8. Call the function clearAll().
9. Return.
```

Algorithm for clearAll():

```
1. For all coordinates set led state to 0;
```

Algorithm for game():

```
1. Set the boolean variable game_on to true.
2. While game_on is true, do the following:
  a. Call the function coordinate_update().
  b. If the variable first_time is equal to 0, do the following:
    i. Set the variable reference_Y to the value returned by the function runArray().
    ii. Set the variable first_time to 1.
    iii. Set the variable stopping_Y to the value of reference_Y.
    iv. Call the function keepon().
  c. Else, do the following:
    i. Set the variable stopping_Y to the value returned by the function runArray().
    ii. Call the function check_alignment().
    iii. Call the function keepon().
  d. Delay for 400 milliseconds.
  e. If the variable x_pos is equal to 31, do the following:
    i. Call the function playMusic() with the argument 1.
    ii. Call the function reset().
      Return.
```

Algorithm for coordinate_update()

```
1. Set the variable x to 7 minus the remainder of x_pos divided by 8.
2. Set the variable blc_num to the integer division of x_pos by 8.
3. Increment the variable x_pos by 1.
4. Set the variable buttonPressed to 0.
5. Return.
```

Algorithm for runArray():

```
1. Set the integer variable y to a random value between 0 and 8 minus the value of the variable size.
2. Set the integer variable yStep to 1.
3. Set the boolean variable down to true.
4. Set the boolean variable stop to false.
5. Set the integer variable lowestY to 0.
6. While stop is false, do the following:
  a. For integer variable i between 0 and 7, do the following:
    i. If i is greater than or equal to y and less than y plus the value of the variable size, set the led at position
(blc_num, i, x) to true using lc.setLed().
    ii. Otherwise, set the led at position (blc_num, i, x) to false using lc.setLed().
  b. If down is true, do the following:
    i. Increment y by yStep.
    ii. If y is greater than or equal to 7 minus size plus 1, set y to 7 minus size plus 1 and set down to false.
  c. Else, do the following:
    i. Decrement y by yStep.
    ii. If y is less than or equal to 0, set y to 0 and set down to true.
  d. Delay for the amount of time specified by the variable DELAY_MS.
  e. If buttonPressed is true, do the following:
    i. Set stop to true.
    ii. Set lowestY to the value of y.
    iii. Set buttonPressed to false.
 1.   Return the value of the variable lowestY.
```

Algorithm for reset():

```
1. Call the function clearAll() to clear the display.
2. Calculate the score by multiplying the value of x_pos by the value of game_level and subtracting 1,
and store the result in the variable score.
3. If the value of score is greater than the value of high_score, set high_score to the value of score.
4. Set the value of x_pos to 0.
5. Set the value of first_time to 0.
6. Set the value of reference_Y to 9.
7. Set the value of stopping_Y to 9.
8. Set the value of size to 4.
9. Set the value of sign to 0.
10. Set the value of game_on to false.
11. Call the function playMusic() with the argument 0 to stop playing music.
12. Return.
```

Algorithm for keepon():

```
1. For each row (i) in the current block, call the function setLed() with the arguments blc_num (block
number), i (row number), and x (column number) set to false, to turn off all LEDs in that row.
2. For each row (i) in the range from reference_Y to reference_Y + size (exclusive), call the function
setLed() with the arguments blc_num (block number), i (row number), and x (column number) set to
true, to turn on the LEDs in that row that correspond to the moving array.
3. Return.
```

Algorithm for checkAllignment():

```
1. Check if block number is 1 and size is 4
2. If yes, set size to 3
3. Check if block number is 2 and size is 3
4. If yes, set size to 2
5. Check if block number is 3 and size is 2
6. If yes, set size to 1
7. Check if absolute difference between reference_Y and stopping_Y is less than size
8. If yes, subtract absolute difference from size
9. If reference_Y is less than stopping_Y, set reference_Y to stopping_Y
10. Play music with ID 5
11. If not, reset the game
12. Return.
```

Algorithm for `status_display()` function:

```
1. Call `print_binary()` function with `high_score`, `score`, and `game_level` as arguments.
2. Delay for 2000 milliseconds.
3. Clear all LEDs on the matrix display.
4. Return.
```

Algorithm for `print_binary(int div, int number)` function:

```
1. Create a boolean array `binaryArray` with size 8.
2. Use a loop to convert the `number` parameter to binary and store the binary digits in `binaryArray`.
   - Start the loop from 7 and go down to 0.
   - If `number` is greater than or equal to 2 to the power of `i`, set `binaryArray[7-i]` to true and subtract 2 to the power of `i` from `number`.
   - Otherwise, set `binaryArray[7-i]` to false.
3. Use nested loops to display the binary digits on the matrix display.
   - Start the outer loop from 0 and go up to 7.
   - Start the inner loop from 1 and go up to 6.
   - If `binaryArray[i]` is true, set the LED at `(div, 7-i, j)` to true, otherwise set it to false.
4. Return.
```

9. ALGORITHM FLOW CHART

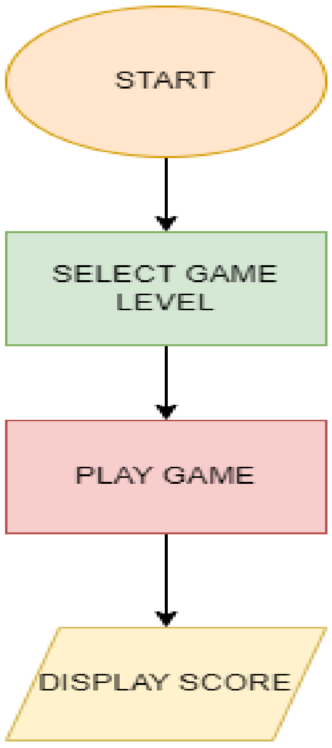

Figure 2: Main algorithm top level abstraction

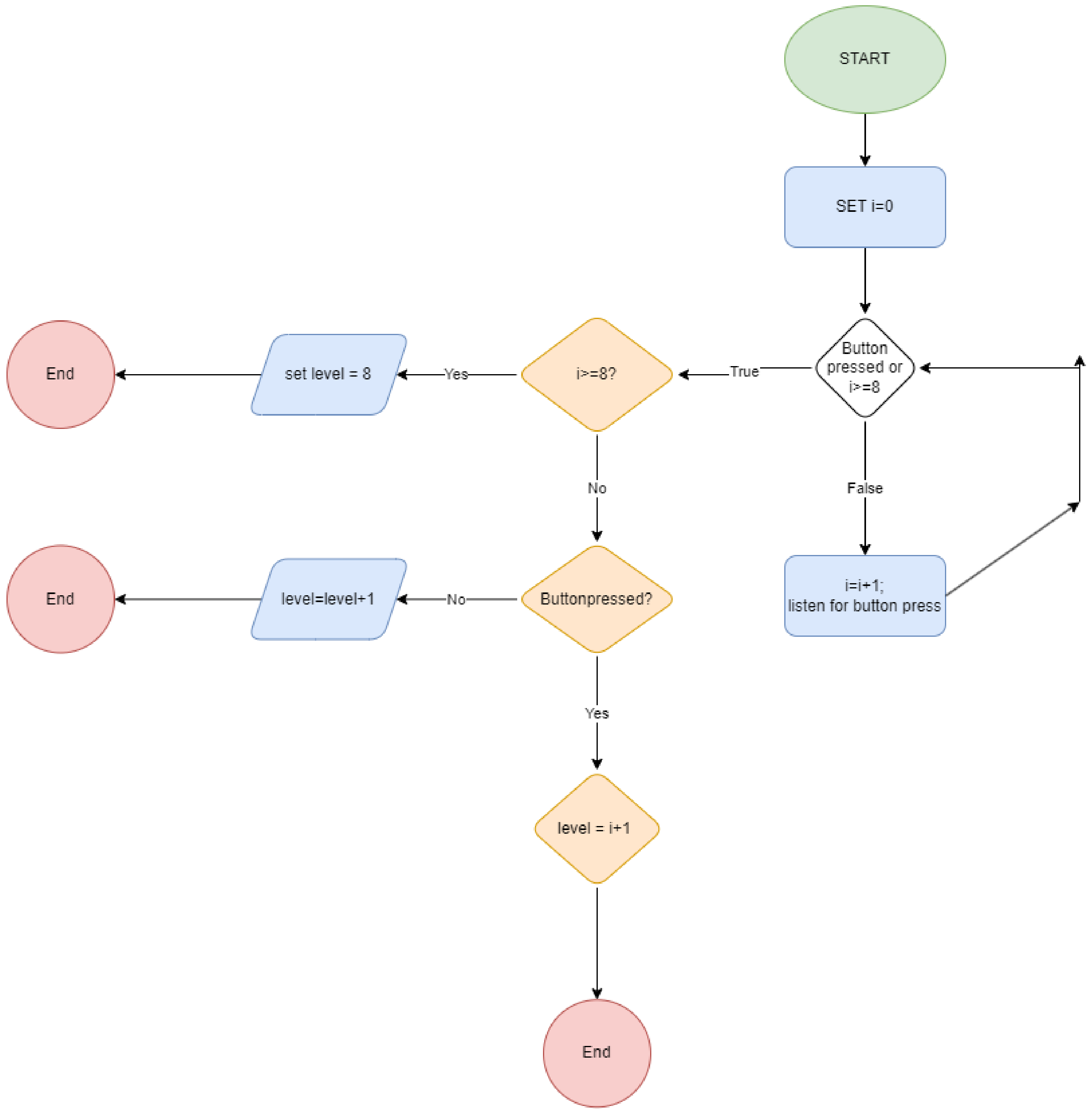

Figure 3: Algorithm to set game level

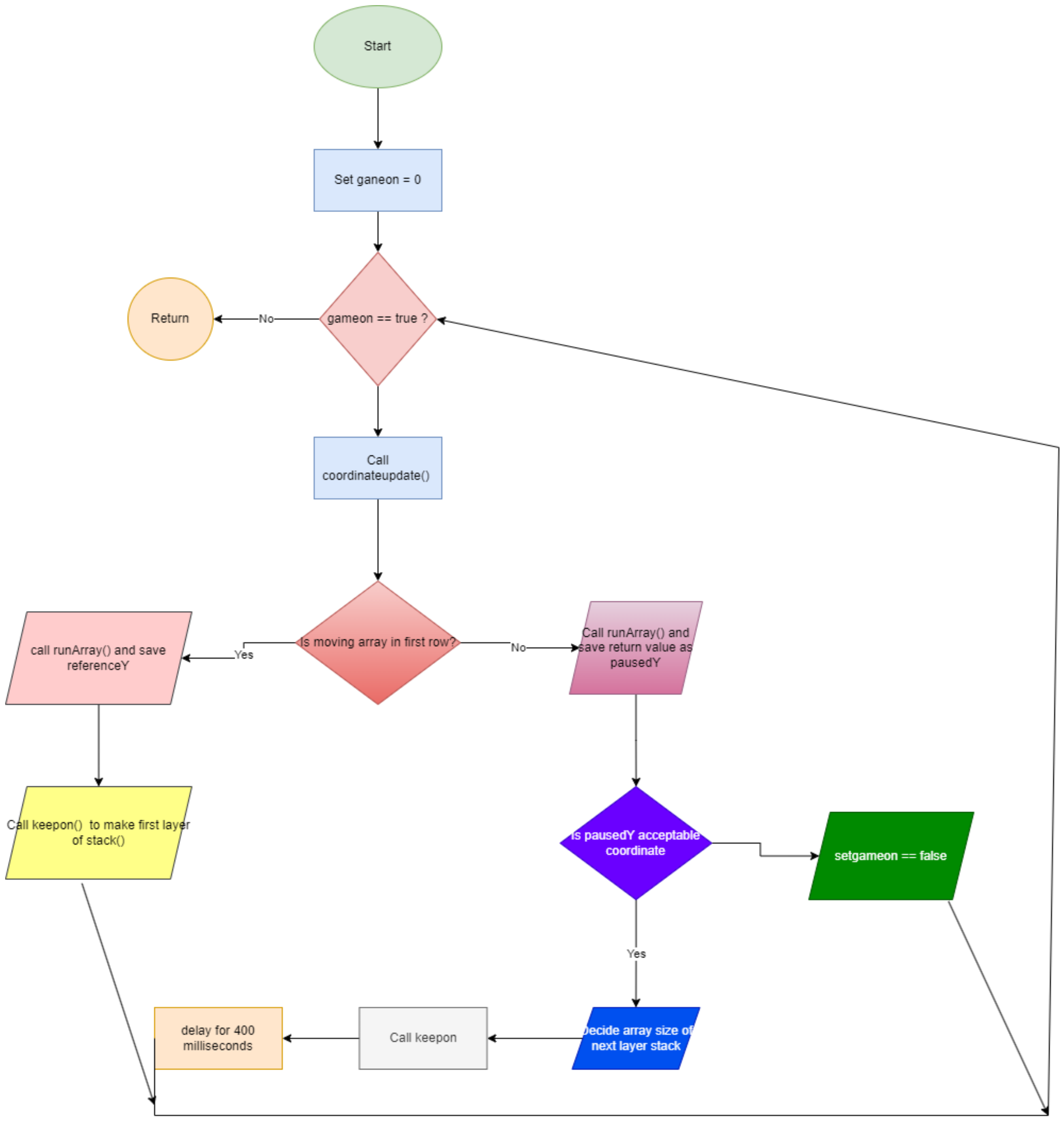
Start
Set ganeon = 0
gameon == true ?
No
Return
Call coordinateupdate()
Is moving array in first row?
Yes
call runArray() and save referenceY
No
Call runArray() and save return value as pausedY
Call keepon() to make first layer of stack()
Is pausedY acceptable coordinate
setgameon == false
Yes
Decide array size of next layer stack
Call keepon
delay for 400 milliseconds

Figure 4: Main algorithm for gameplay

10. RESULTS & DISCUSSION

We implemented the game successfully on Arduino UNO successfully and were able to achieve a very smooth and enjoyable multi-level gaming experience. Players can select their desired level and enjoy the game. How high can one score depends on focus, watchfulness, strategy and reaction time.

Results of this project suggest that many basic games can be implemented on Arduino board and MAX7219 LED screen using similar development methodology. Some other games that can be easily implemented by slightly modifying and using the functions of this program are Bubble Shooter, Tetris, snake eating bubbles, etc.

This also gives motivation to develop temporary and permanent interactive consoles to automate and ease certain processes like allocation of seats based on some id.

11. CONCLUSION

From results of our project, we can conclude that we can easily build and deploy various simple games using minimal hardware. With a little bit of creativity and ingenuity it is possible to develop simple games with minimal use of hardware. Working on the project was fun and challenging. Along the way we needed to troubleshoot many unforeseen errors and difficulties mainly due to introduction of delays which affected unintended parts of the program. The project was a great learning experience for us in the field of electronics, programming and game development.

12. ACKNOWLEDGEMENT

We are grateful to our instructor Dr Ribhu who guided us through the project. Also, we are thankful to modern technological advances which made gathering knowledge, debugging and procuring components very easy.

13. REFERENCES

I. Arduino documentations, https://www.arduino.cc/reference/en/
II. https://lastminuteengineers.com/max7219-dot-matrix-arduino-tutorial/