Image 1
Here I was having trouble with the dim LED at position (3,2) (top left is 0,0) as seen in the image. The fixed arduino code (and what the board SHOULD look like) can be seen in Image 2 – it turned out that I needed to slow down the loop function so the persistence of vision illusion would work better.

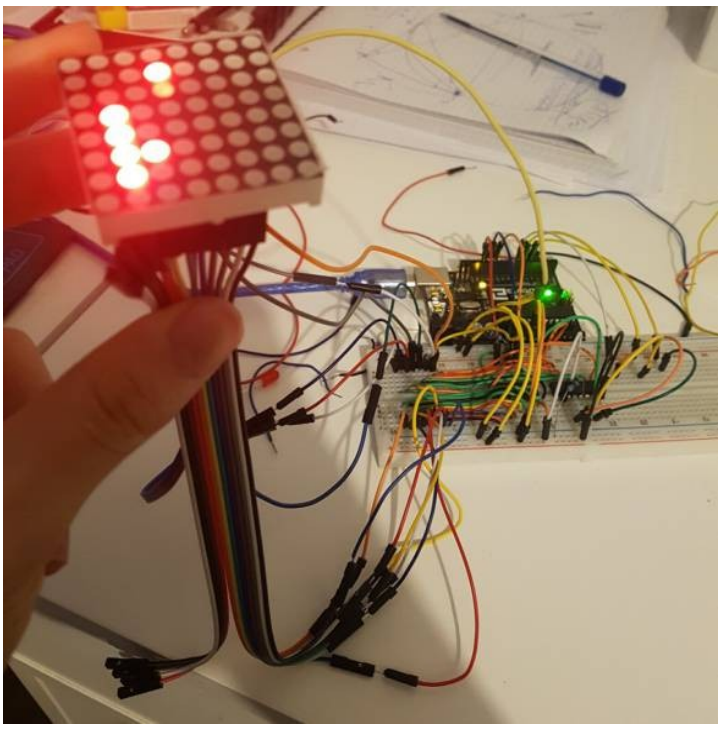
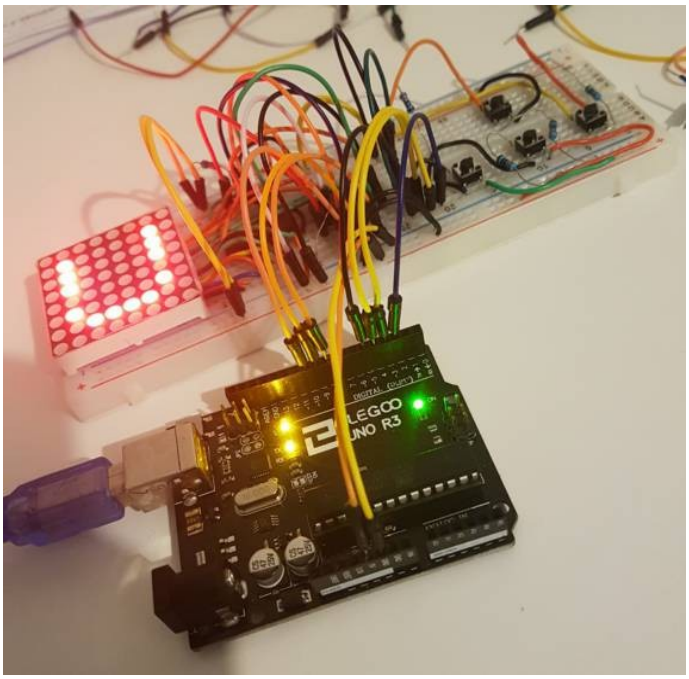


Image 2
The arduino sketch file for v1.

```
const int latchPin1 = 12;
const int latchPin2 = 6;
const int clockPin1 = 11;
const int clockPin2 = 7;
const int dataPin1 = 13;
const int dataPin2 = 5;

byte first_shift_reg_data = 0;
byte second_shift_reg_data = 0;

int board[8][8] = {
  {0,0,0,0,0,0,0,0},
  {0,0,0,1,0,0,0,0},
  {0,0,0,0,0,0,0,0},
  {0,1,0,0,0,0,0,0},
  {0,1,0,0,0,0,0,0},
  {0,1,1,0,0,0,0,0},
  {0,1,0,0,0,0,0,0},
  {0,0,0,0,0,0,0,0}
};

// Q_A to Q_H of first register labelled 0-7, Q_A to Q_H of second labelled 8-15
// index n is row/col n, val at index n is the corresponding register
int row_mappings[8] = {15, 10, 1, 12, 0, 2, 7, 4};
int col_mappings[8] = {11, 6, 5, 14, 3, 13, 9, 8};

void setup()  {
  pinMode(latchPin1, OUTPUT);
  pinMode(dataPin1, OUTPUT);
  pinMode(clockPin1, OUTPUT);

  pinMode(latchPin2, OUTPUT);
  pinMode(dataPin2, OUTPUT);
  pinMode(clockPin2, OUTPUT);
}

void loop() {
    for(int y=0; y<8; y++) {
    first_shift_reg_data = 0;
    second_shift_reg_data = 0;
    // activate row
    if(row_mappings[y]>7){
      bitSet(second_shift_reg_data, row_mappings[y]-8);
    } else {
      bitSet(first_shift_reg_data, row_mappings[y]);
    }

    for(int x=0; x<8; x++) {
      if(board[y][x]==0){
        if(col_mappings[x]>7){
          bitSet(second_shift_reg_data, col_mappings[x]-8);
        } else {
          bitSet(first_shift_reg_data, col_mappings[x]);
        }
      }
    }

    latch();

    // deactivate row
    if(row_mappings[y]>7){
      bitClear(second_shift_reg_data, row_mappings[y]-8);
    } else {
      bitClear(first_shift_reg_data, row_mappings[y]);
    }
  }
}

void latch(){
  digitalWrite(latchPin1, LOW);
  digitalWrite(latchPin2, LOW);
  shiftOut(dataPin1, clockPin1, MSBFIRST, first_shift_reg_data);
  shiftOut(dataPin2, clockPin2, MSBFIRST, second_shift_reg_data);
  digitalWrite(latchPin1, HIGH);
  digitalWrite(latchPin2, HIGH);
}
```

Image 3
v1 is shown working



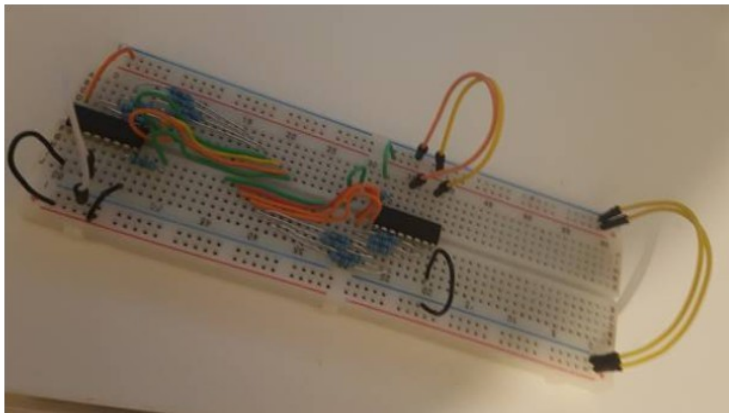


Image 4
v1 is shown in the process of being redesigned to v2 – better wire organization, added resistors to limit the current through the LEDs (previously could not because of the lack of space on the breadboard but this redesign allows it)

Image 5
v2 working – note that the buttons have been changed to be pulled high so no resistors are needed which simplifies the wiring.

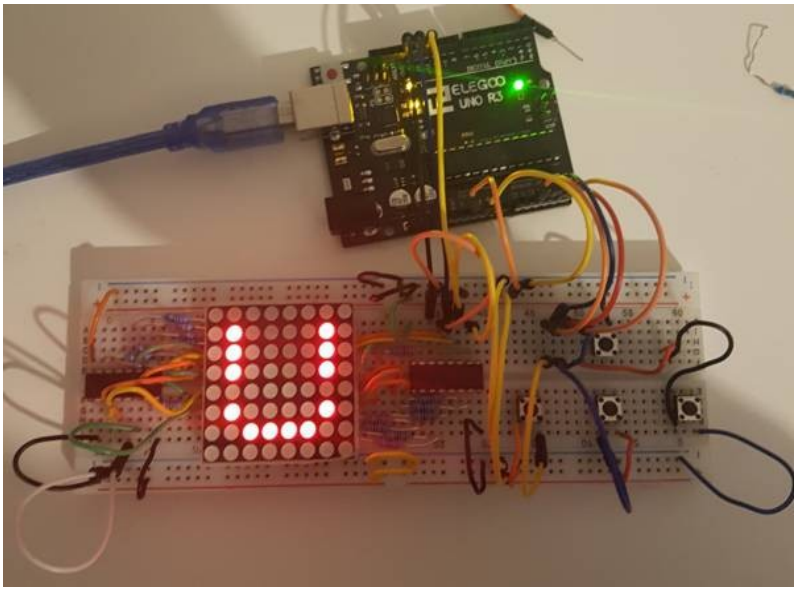


Image 6
The code for the LED_matrix_8_8 class (.cpp file only) for v3.

```cpp
#include "LED_matrix_8_8.h"

#include "Arduino.h"

LED_matrix_8_8::LED_matrix_8_8(int data, int latch, int clock) : dataPin(data), latchPin(latch), clockPin(clock) {
    pinMode(latchPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
}

void LED_matrix_8_8::setPixel(int x, int y, int val) {
    grid[y][x] = val;
}

void LED_matrix_8_8::set(int grid[8][8]) {
    for (int y = 0; y < 8; y++) {
        for (int x = 0; x < 8; x++) {
            this->grid[y][x] = grid[y][x];
        }
    }
}

void LED_matrix_8_8::draw() {
    // scan row by row, activating needed columns
    for (int y = 0; y < 8; y++) {
        // activate row
        bitSet(reg_data[row_mappings[y] > 7], row_mappings[y] % 8);

        // activate columns
        for (int x = 0; x < 8; x++) {
            if (grid[y][x]) {
                bitClear(reg_data[col_mappings[x] > 7], col_mappings[x] % 8);
            } else {
                bitSet(reg_data[col_mappings[x] > 7], col_mappings[x] % 8);
            }
        }

        latch();
        delay(1);

        // deactivate row
        bitClear(reg_data[row_mappings[y] > 7], row_mappings[y] % 8);

        // deactivate columns
        for (int x = 0; x < 8; x++) {
            bitSet(reg_data[col_mappings[x] > 7], col_mappings[x] % 8);
        }
    }
}

void LED_matrix_8_8::latch() {
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, MSBFIRST, reg_data[0]);
    shiftOut(dataPin, clockPin, MSBFIRST, reg_data[1]);
    digitalWrite(latchPin, HIGH);
}
```

Image 7
A snippet of the code from the sketch file for v3 is shown below.

```cpp
void logicUpdate(){
  // process input
  if(inputQueueLen>0){
    // if direction to change to isn't opposite current dir
    if(dir != inputQueue[0]-2 && dir != inputQueue[0]+2){
      dir = inputQueue[0];
    }
    inputQueueLen--;
    for(int i=0; i<inputQueueLen; i++){
      inputQueue[i] = inputQueue[i+1];
    }
  }

  // check if dead
  int nextX = snake[0].x + dx[dir];
  int nextY = snake[0].y + dy[dir];
  if(nextX<0 || nextY<0 || nextX>7 || nextY>7){
    die();
    return;
  }
  for(int i=1; i<len; i++){
    if(snake[i].x == nextX && snake[i].y == nextY){
      die();
      return;
    }
  }

  // check food
  boolean grow = nextX == foodPos[0] && nextY == foodPos[1];
  if(grow){
    len++;
    // speed up game
    tickstep-=0.5;
  }

  // update snake pos
  if(!grow) {
    screen.setPixel(snake[len-1].x, snake[len-1].y, 0);
  }
  for(int i=len-1; i>0; i--){
    snake[i].x = snake[i-1].x;
    snake[i].y = snake[i-1].y;
  }
  snake[0].x = nextX;
  snake[0].y = nextY;
  screen.setPixel(snake[0].x, snake[0].y, 1);

  // update food pos
  if(grow){
    repositionFood();
  }

}
```

Image 8
The completed project (v3)