1. Introduction

Snake Game has been very popular since the beginning of the Mobile phones. Initially it was come in Black and white cell phones, and soon became very famous. Then with the advancement of the Cellphones, this game has also changed a lot, and now many graphical and colorful versions of this game are available. The simple mechanism inside the Snake Game may make it so popular and prevalent among various devices. For good understanding of Atmega processor and AVR programming, I want to implement a LED version Snake Game, it going to be very interesting.

2. Features

- Represent the food as a dot
- Represent the snake as a line (at least two dots)
- 5 scores up every time eat a dot
- Speed up the snake every time eat a dot
- Output the score using integrated 7-seg
- Using 4 buttons to control the direction
- Using a button for pause/resume the game (on board button)

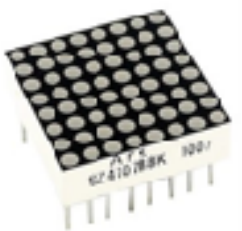
3. Working Explanation:

a. The game flow

To make this project, I would use an 8x8 red color Dot matrix display for displaying the snake and food dot, a 7-seg for displaying the points or score, 4 push buttons for giving directions and a on board button for pause/resume the game, finally an JKIT-128-1 as a MCU for processing the game logic and drive the led matrix. When the circuit power up, initiate the game and displaying an animation to indicate the initiation done. After this, 7-seg shows the score as zero and led matrix displays two dots as snake and a single dot as food. The user need to press the on-board button to start the game and snake start moving in upward direction by default. Then the user needs to give direction to snake by pressing the direction keys. Whenever the snake reaches to the food dot, the scores increases by 5 points each time and the Snake length is increased by one dot (LED) each time, and speed up the moving speed of the snake. Whenever snake's head hits its body, then game over. The user needs to start game again by pressing start key.

b. The hardware

max7219

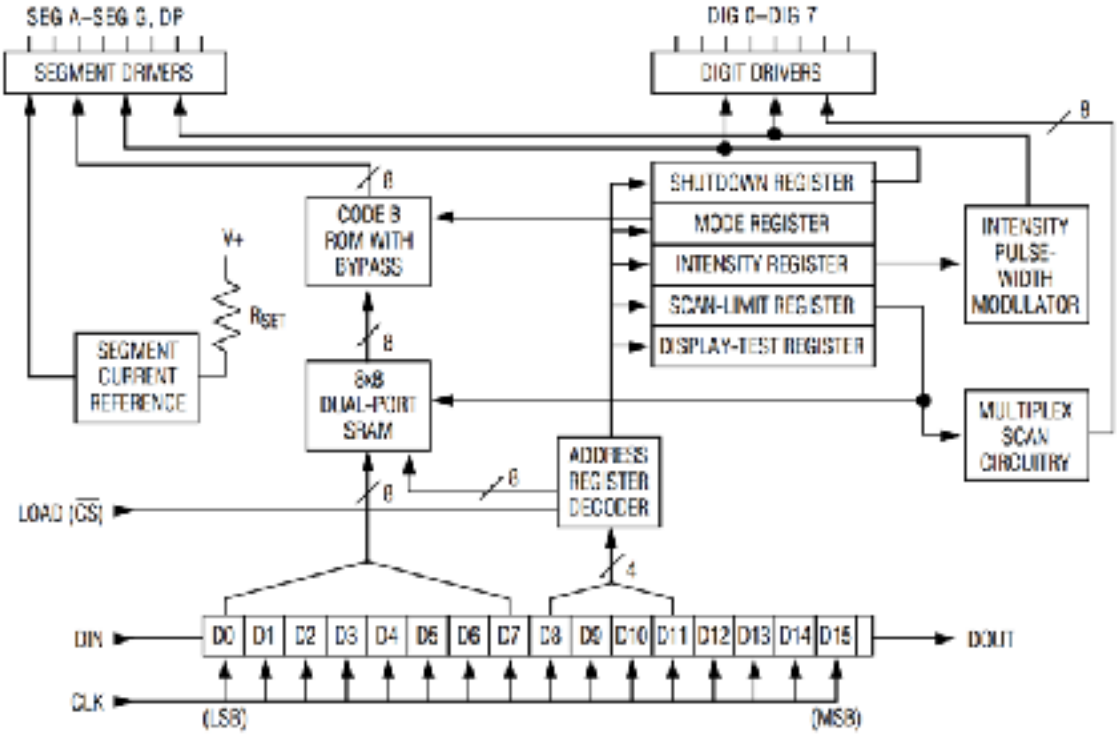


8x8 led matrix

Using a max7219 can drive 64 LEDs while I only need 4 wires to interface it to Jkit-128-1. It can save lots of pins and reduce the wiring complexity.
The max7219 has a four wire SPI interface - **clock, data, chip select and ground** - making it very simple to connect to a microcontroller.

- Data - **MOSI** - Master Output Serial Input. The 7219 is a slave device.
- Chip select - **Load (CSn)** - active low Chip select.
- Clock - **SCK**
- **Ground**.

max7219 Specification	
Power Supply:	4.0V ~ 5.5V
Supply Current:	330mA
Segment drive source current	-40mA
Scan rate	800Hz
CLK	max 10MHz

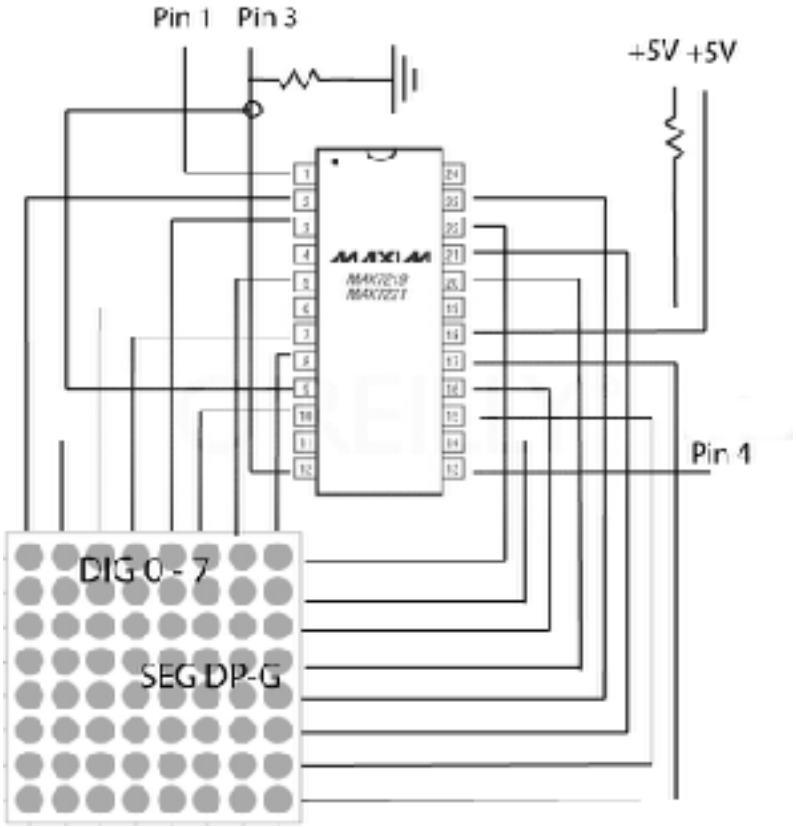


D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB	DATA						LSB

The MAX7219 input interface is a 16 bit serial input shift register. The first 8 bits define a command while the second 8 bits define the data for that command (see diagram below):

Serial Data Format

Only 4 address lines select the operation to perform, the other 4 are not used. Note that the MSB is sent first. The top 4 bits are redundant but still have to send them into the shift register to fill that register completely.
The most important point is that multiple MAX7219 devices can be cascaded together simply by feeding in the Dout from one to the Din of the next. Load and clock (CLK) are kept the same for all devices.
The load pulse (CSn) is a trigger that sends all the data from the shift register into each control register in the device, at the same time. We can change the shift register contents as much as we need to and when all done, finally transfer the data to the control registers using the CSn line by pulsing it low then high.



Schematic of 8x8 LED Matrix and max7219

c. The software

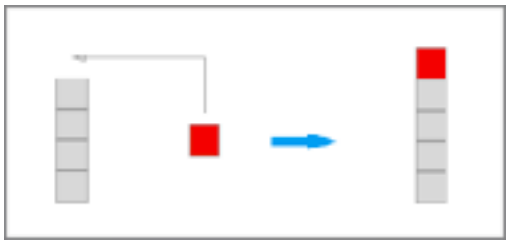
The snake is constructed with nodes, I represents the node with a struct in c.

```
typedef struct Node{
    uint8_t x;
    uint8_t y;

    struct Node* next;
}node_t;
```

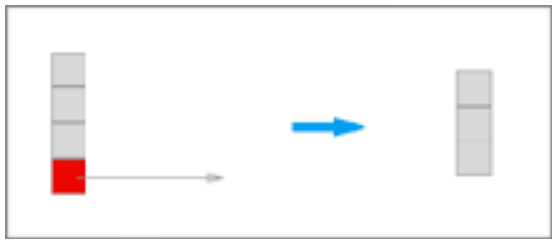
The snake is represented by a linked list, as an ADT, I defined two method among the ADT.

```
void push_head(node_t* head2go)
{
   node_t* current_head = head;
   head2go->next = current_head;
   head = head2go;
}
void pop_tail()
{
    node_t* current_head = head;

    while(current_head->next->next != NULL){
        current_head = current_head->next;
    }
    free(current_head->next);
    current_head->next = NULL;
}
```

The push_head is for adding a new node to the head, the pop_tail if for removing the last node(tail) in the linked list.



push_head



pop_tail

To make the snake move, just push a head and pop a tail, it can make a illusion of moving. Changing the direction is also by pushing a head to the linked list, before we push the

node to the linked list, we determine the proper coordinate of the node, then push it. If the position of the new node is a food, we only push a head, don't pop a tail. I tied these stuff together to a move function.

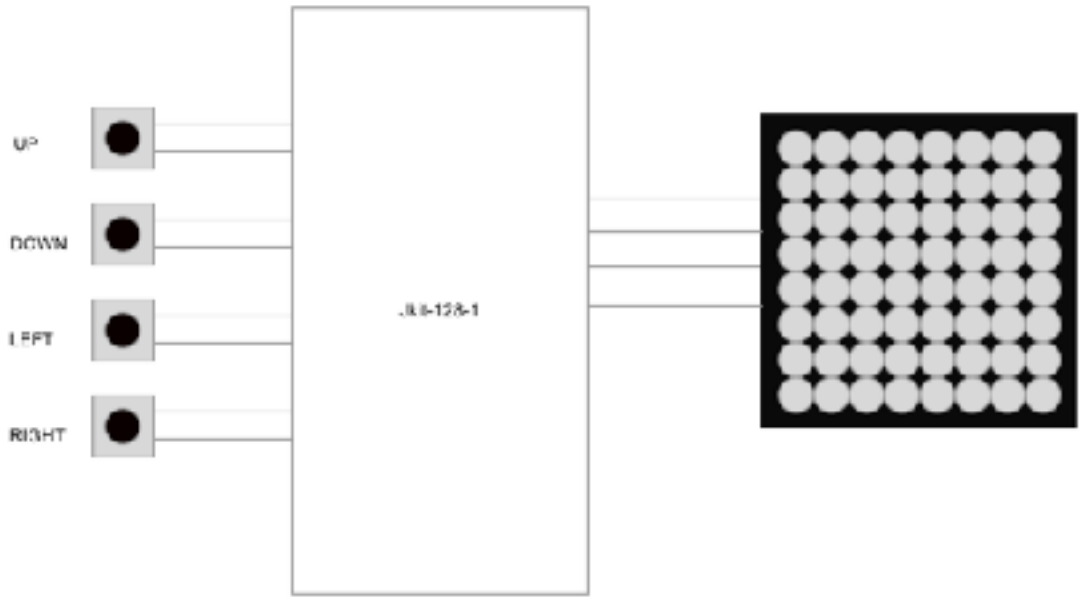
```
void move(void)
{
  node_t* temp_head = (node_t*)malloc(sizeof(node_t));
  int x = head->x; //current head x
  int y = head->y; //current head y

  switch(direction){
    case up:
      x += 1;
      break;
    case down:
      x -= 1;
      break;
    case left:
      y -= 1;
      break;
    case right:
      y += 1;
      break;
    default:
      break;
  }
  if(x > 7) x = 0; //there is no wall in the game, this is for
  if(y > 7) y = 0; //crossing the wall.
  if(x < 0) x = 7;
  if(y < 0) y = 7;
  temp_head->x = x;
  temp_head->y = y;
  push_head(temp_head);

  if(head->x == food->x && head->y == food->y){
    generate_food();
    scores += 5;
    speed -= 3;
    if(speed < 3) speed = 3;
  }
  else{
    pop_tail();
  }
}
```

I use a on-board timer to control the moving speed.

```
ISR(TIMER1_OVF_vect)
{
```

```
    tot_overflow++; // 30 overflows = 1 seconds delay (approx.)

    if (tot_overflow >= speed)
    {
        if(running) move();
        tot_overflow = 0;   // reset overflow counter
    }
}
```

4. Circuit

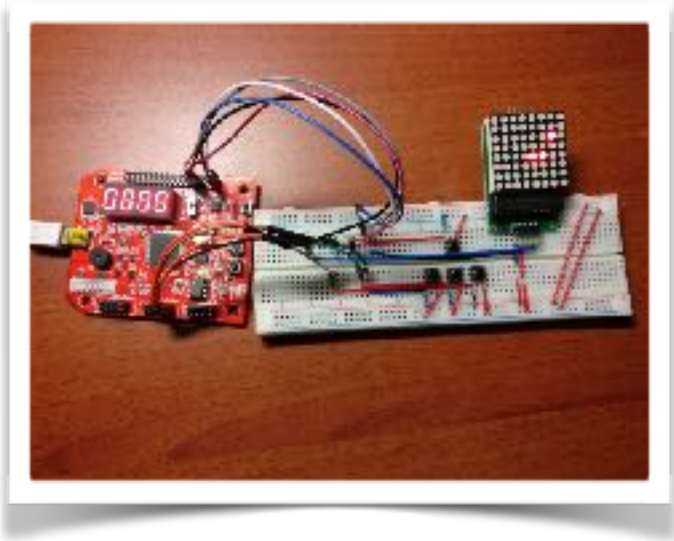


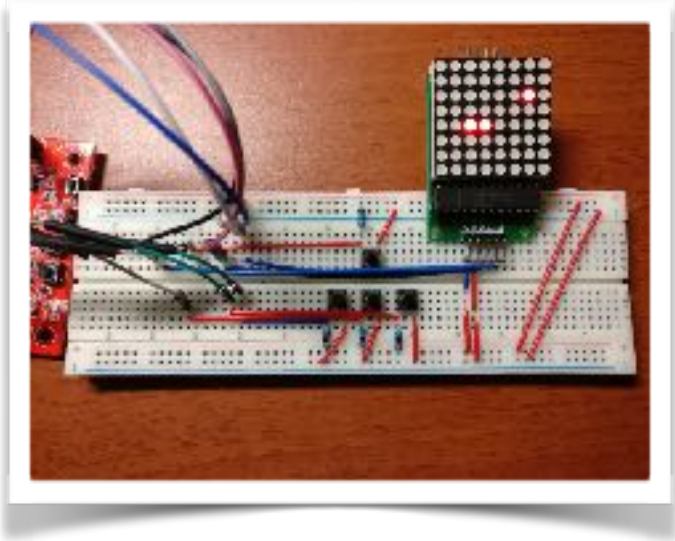
5. Components

- JKIT-128-1
- 8x8 LED Dot Matrix
- max7219
- Push Buttons

- Bread Board
- Connecting wires

6. The photos of the project





7. Source Code

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include "max7219.h"
#define F_CPU 16000000UL

typedef unsigned char uchar;
typedef struct Node{
    uint8_t x;
    uint8_t y;

    struct Node* next;
}node_t;

typedef struct Food{
       uint8_t x;
       uint8_t y;
}food_t;

typedef enum {up, down, left, right} Directions;
uint8_t scrolls_index1 = 0;
uint8_t screen_buffer1[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t buffer_row1;

volatile uint8_t tot_overflow;
volatile uint8_t speed = 40;
volatile uint8_t collision = 0;
volatile uint8_t running = 0;
uint8_t scores = 0;
```

```
node_t* head = NULL;
food_t* food = NULL;
Directions direction = up;
const uchar digit[10] = { 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7c, 0x07, 0x7f, 0x67 };

const uint8_t scrolls[100] PROGMEM = {
  SCROLL________,
  SCROLL________,
  SCROLL________,
  SCROLL________,
  SCROLLX______X,
  SCROLLX______X,
  SCROLLXX____XX,
  SCROLLXX____XX,
  SCROLLXXX___XX,
  SCROLLXXX___XX,
  SCROLLXX____XX,
  SCROLLX______X,
  SCROLL________,
  SCROLL________,
  SCROLL_______X,
  SCROLL______XX,
  SCROLL_____XXX,
  SCROLL____XXXX,
  SCROLL___XXXXX,
  SCROLLX___XXXX,
  SCROLLXX___XXX,
  SCROLLXXX___XX,
  SCROLLXXXX___X,
  SCROLLXXXXX___,
  SCROLLXXXX___X,
  SCROLLXXX___XX,
  SCROLLXX___XXX,
  SCROLLX___XXXX,
  SCROLL___XXXXX,
  SCROLLX___XXXX,
  SCROLLXX___XXX,
  SCROLLXX____XX,
  SCROLLX______X,
  SCROLLX______X,
  SCROLL________,
  SCROLL________,
  SCROLL________,
  SCROLL____X___,
  SCROLL____X___,
  SCROLL____X___,
  SCROLL___XX___,
  SCROLL___XX___,
  SCROLL___XX___,
  SCROLL____X___,
  SCROLL____X___,
  SCROLL____X___,
  SCROLL________,
  SCROLL________,
  SCROLL________,
  SCROLLX______X,
  SCROLLX______X,
  SCROLLXX____XX,
  SCROLLXX____XX,
  SCROLLX______X,
  SCROLLX______X,
  SCROLL________,
  SCROLL________,
  SCROLL________,
  SCROLL___X____,
  SCROLL___X____,
  SCROLL___X____,
  SCROLL___XX___,
  SCROLL___XX___,
  SCROLL___XX___,
  SCROLL___X____,
  SCROLL___X____,
```

```
  SCROLL___X___X,
  SCROLL___X___X,
  SCROLL___X___X,
  SCROLL___X____,
  SCROLL____X___,
  SCROLL____X___,
  SCROLL____X___,
  SCROLL________,
  SCROLL________,
  SCROLLX______X,
  SCROLLXX____XX,
  SCROLLXX___XXX,
  SCROLLX____XXX,
  SCROLL____XXXX,
  SCROLL___XXXXX,
  SCROLL___XXXXX,
  SCROLL___XXXXX,
  SCROLLX___XXXX,
  SCROLLXX___XXX,
  SCROLLXX____XX,
  SCROLLX______X,
  SCROLL________,
  SCROLL________,
  SCROLLXXXXXXXX,
  SCROLL________,
  SCROLLXXXXXXXX,
  SCROLL________,
  SCROLLXXXXXXXX,
  SCROLL________,
  SCROLLXXXXXXXX,
  SCROLL________,
  SCROLLXXXXXXXX,
  SCROLL________,
  SCROLLXXXXXXXX
};

void init_snake(int init_x, int init_y);
void push_head(node_t* head2go);
void pop_tail();
void get_direction();
void init_food(void);
void generate_food(void);
void move(void);
void output(void);
void timer1_init();
void check_collision(void);
void button_init();
void control_fnd(uchar *nums);
void keep_fnd(uchar *nums, int sec);
void init_fnd();
void print_scores(int cnt);
void reset_game();
void screen_saver();

ISR(TIMER1_OVF_vect)
{
    tot_overflow++;

    // 30 overflows = 1 seconds delay (approx.)
    if (tot_overflow >= speed)
    {
        if(running) move();
        tot_overflow = 0;   // reset overflow counter
    }
}
ISR(INT5_vect)
{
    if(running == 1) running = 0;
    else running = 1;
}
int main(void)
{
```

```
    init_fnd();
    button_init();
    max7219_init();
    timer1_init();

    init_snake(rand()%8, rand()%8);
    init_food();
    while(1){
        if(running){
          output();
          print_scores(scores);
          get_direction();
          _delay_ms(2);
          check_collision();
          while(collision){
              reset_game();
              running = 0;
          }
        }
        if(!running){
          screen_saver();
          print_scores(scores);
        }
    }
}
void print_scores(int cnt) {
  int i, tmp = cnt;
  uchar nums[4] = {0,};
  for(i=3;i>=0;i--) {
    nums[i] = tmp % 10;
    tmp /= 10;
  }

  keep_fnd(nums, 1);
}
void control_fnd(uchar *nums) {
  int i = 0x08;
  int j = 0;

  for(; i>0; i= i >>1, j++) {
    PORTG = i;
    PORTC = digit[nums[j]];
    _delay_ms(1);
  }
}

void keep_fnd(uchar *nums, int sec) {
  int cnt = sec*250;
  while(cnt--) {
    control_fnd(nums);
  }
}
void init_fnd() {
  DDRC = 0xff;
  DDRG = 0x0f;

}
void button_init()
{
    DDRF = 0x00;
    EICRB = 0x0A;
    EIMSK = 0x30;
}
void pop_tail()
{
    node_t* current_head = head;

    while(current_head->next->next != NULL){
        current_head = current_head->next;
    }
    free(current_head->next);
    current_head->next = NULL;
```

```
}

void push_head(node_t* head2go)
{
   node_t* current_head = head;
   head2go->next = current_head;
   head = head2go;
}

void init_snake(int init_head_x, int init_head_y)
{
   head = (node_t*)malloc(sizeof(node_t));
   head->x = init_head_x;
   head->y = init_head_y;
   head->next = NULL;
   node_t* tail = (node_t*)malloc(sizeof(node_t));
   tail->x = init_head_x - 1;
   tail->y = init_head_y;
   head->next = tail;
   tail->next = NULL;
}

void get_direction()
{
  if((!(PINF & (1<<PINF0))) && direction != down){
            direction = up;
        }
        if((!(PINF & (1<<PINF1))) && direction != up){
            direction = down;
        }
        if((!(PINF & (1<<PINF2))) && direction != right){
            direction = left;
        }
        if((!(PINF & (1<<PINF3))) && direction != left){
            direction = right;
        }
}

void init_food(void)
{
  food = (food_t*)malloc(sizeof(node_t));
  food->x = 5;
  food->y = 6;
}

void generate_food(void)
{
  food->x = rand()%8;
  food->y = rand()%8;
}
void reset_game()
{
  node_t* temp_head;
  scores = 0;
  while(head != NULL){
    temp_head = head;
    head = head->next;
    free(temp_head);
  }
  init_snake(rand()%8, rand()%8);
  init_food();
  speed = 40;
  collision = 0;
}
void move(void)
{
  node_t* temp_head = (node_t*)malloc(sizeof(node_t));
  int x = head->x;
  int y = head->y;

  switch(direction){
    case up:
```

```
      x += 1;
      break;
    case down:
      x -= 1;
      break;
    case left:
      y -= 1;
      break;
    case right:
      y += 1;
      break;
    default:
      break;
  }
  if(x > 7) x = 0;
  if(y > 7) y = 0;
  if(x < 0) x = 7;
  if(y < 0) y = 7;
  temp_head->x = x;
  temp_head->y = y;
  push_head(temp_head);

  if(head->x == food->x && head->y == food->y){
    generate_food();
    scores += 5;
    speed -= 3;
    if(speed < 3) speed = 3;
  }
  else{
    pop_tail();
  }
}

void output(void)
{
  max7219b_out();
  node_t* current_head = head;
  int x, y;

  for(int i = 0; i < 8; i++){
    for(int j = 0; j < 8; j++){
      max7219b_clr(i, j);
    }
  }
  while(current_head != NULL){
    max7219b_set(current_head->x, current_head->y);
    current_head = current_head->next;
  }
    max7219b_set(food->x, food->y);
}
void timer1_init()
{
    TCCR1B |= (1 << CS11);

    // initialize counter
    TCNT1 = 0;

    // enable overflow interrupt
    TIMSK |= (1 << TOIE1);

    // enable global interrupts
    sei();

    // initialize overflow counter variable
    tot_overflow = 0;
}

void check_collision(void)
{
    node_t* current_head = head;
    uint8_t x = current_head->x;
    uint8_t y = current_head->y;
```

```
    do{
        current_head = current_head->next;
        if(current_head != NULL && current_head->x == x && current_head->y == y) collision = 1;
    }while(current_head != NULL);
}

void screen_saver(){

  for (uint8_t screen_row = 8; screen_row >= 1; screen_row--) {
      buffer_row1 = screen_row - 1;
      // Output the buffer
      max7219_row(screen_row, screen_buffer1[buffer_row1]);

      // Scroll the row down
      if (buffer_row1 > 0) {
        screen_buffer1[buffer_row1] = screen_buffer1[buffer_row1 - 1];
      } else {
        screen_buffer1[buffer_row1] = pgm_read_byte(&scrolls[scrolls_index1++]);
        if (scrolls_index1 >= sizeof(scrolls) - 1) scrolls_index1 = 0;
      }
    }

    _delay_ms(10);

}
```