GROUP NUMBER:

ROLL NUMBER 1:200260016 NAME 1: Dosapati Jayanth
ROLL NUMBER 2:200260056 NAME 2: Uppala Mukesh

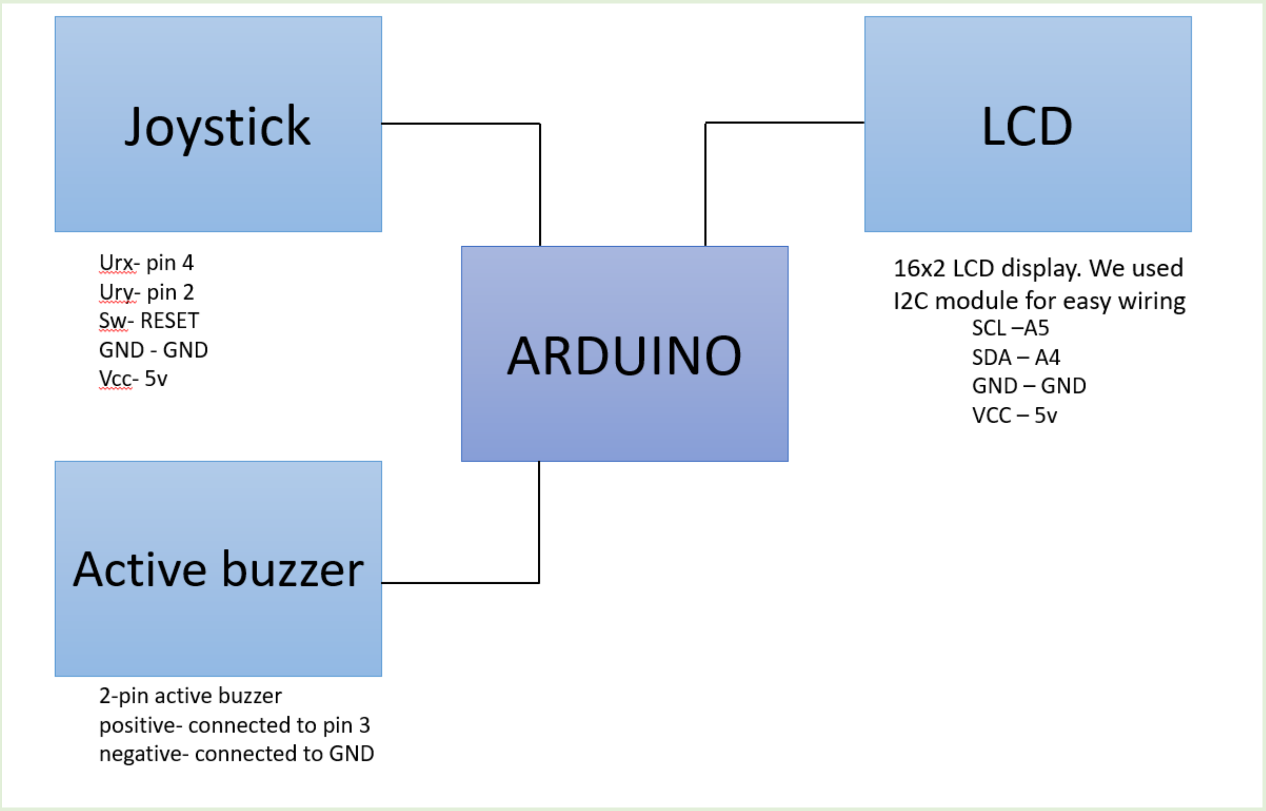
TITLE: Run and Jump

ABSTRACT:

Created a 16x2 LCD display game. The player can select different game modes. Random blocks will emerge from right moving to left, the man has to jump to escape them. Initially the man is in the 2nd

Row of the LCD. The score is calculated by how many seconds the player had played and is displayed at top right corner of LCD display

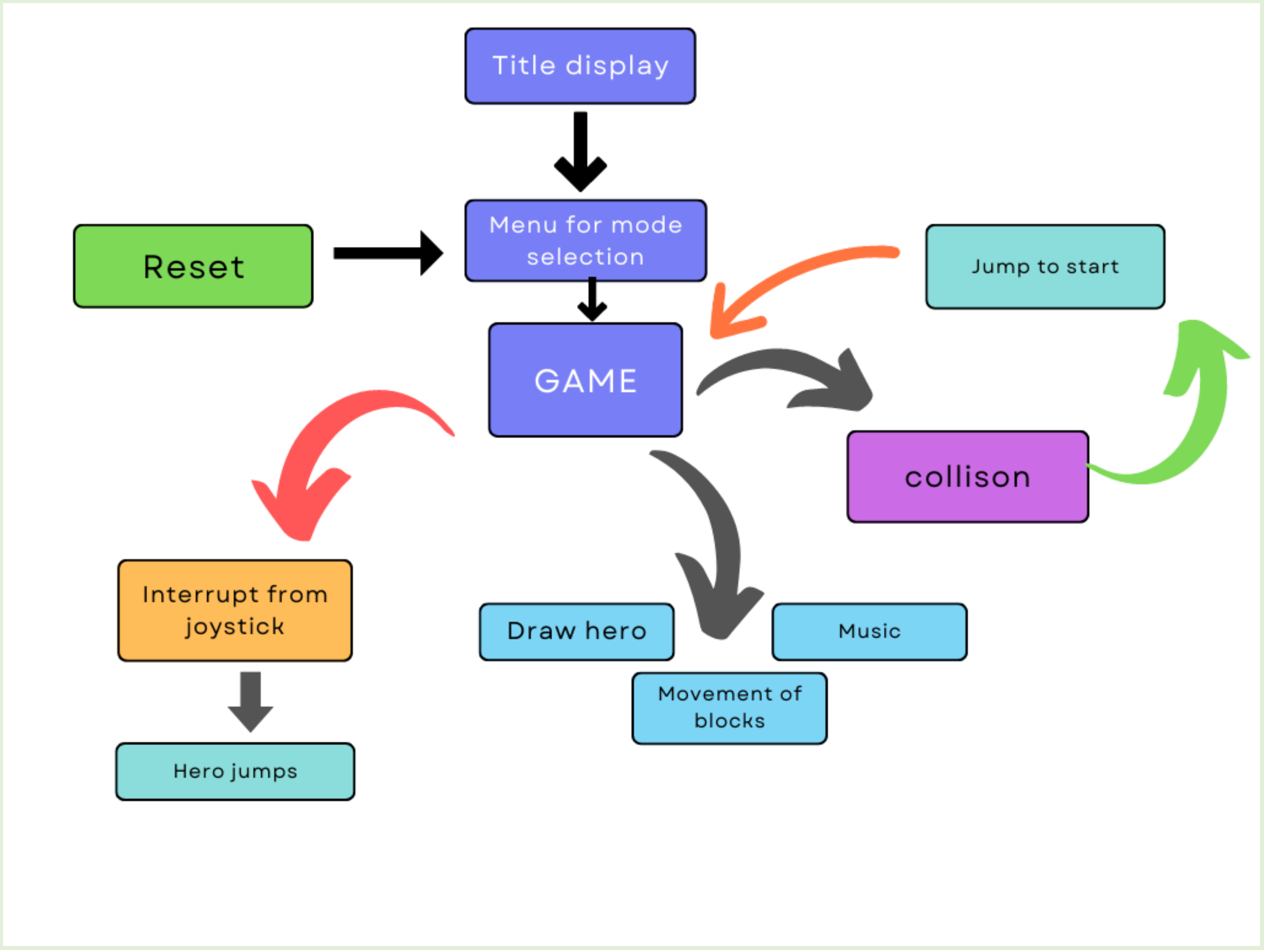
PROJECT DETAILS:



algorithm flowchart

Drawhero, Interrupt, menu function- done by Dosapati Jayanth

Music, movement of blocks, circuit- done by Uppala Mukesh



MAIN COMPONENTS NEEDED TO BUILD THE PROJECT:

1)16x2 LCD display

2)I2c module

3)Joystick

4)Active buzzer

RESULTS:

Demo link: https://drive.google.com/drive/folders/1E7LDxF1vJbLbzfei113s1SaETIu-c389

APPENDIX:

Program code is to be put here in the following fixed width font. Note that the code must be well commented and self-explanatory. The following is a shining example of well-written code:
(the color coding of functions is just a cosmetic add-on for readability)

```
// GND and Pin 3 - Buzzer

// GND and Reset - Sw from joystick

// GND and pin 4 - Urx of joystick

// GND and pin 2 - Ury of joystick

// SCL - A5

// SDA - A4

// GND - GND

// VCC - 5v

#include <LiquidCrystal_I2C.h>

#include<avr/interrupt.h>

volatile int game_mode; //total game modes

volatile int select_mode; // selected  game mode

#define VRy 2 // joystick Y-axis for hero to jump

#define SPRITE_RUN1 1 // Hero running posture with open hands and open legs

#define SPRITE_RUN2 2 // Hero running posture with closed hands

#define SPRITE_JUMP 3 // Hero posture while jumping up

#define SPRITE_JUMP_UPPER '.'   //

#define SPRITE_JUMP_LOWER 4   // Hero position while jumping to ground

#define SPRITE_TERRAIN_EMPTY ' ' // Empty block (5x8 pixels)

#define SPRITE_TERRAIN_SOLID 5  // Full (5x8 pixels)

#define SPRITE_TERRAIN_SOLID_RIGHT 6 // (right half filled block)

#define SPRITE_TERRAIN_SOLID_LEFT 7 // left half filled block

#define HERO_HORIZONTAL_POSITION 1   // same as

#define TERRAIN_WIDTH 16  // complete length of lcd screen (16x2)

/* variables for different orientations of terrain */

#define TERRAIN_EMPTY 0

#define TERRAIN_LOWER_BLOCK 1

#define TERRAIN_UPPER_BLOCK 2



#define HERO_POSITION_OFF 0

#define HERO_POSITION_RUN_LOWER_1 1

#define HERO_POSITION_RUN_LOWER_2 2

#define HERO_POSITION_JUMP_1 3

#define HERO_POSITION_JUMP_2 4

#define HERO_POSITION_JUMP_3 5

#define HERO_POSITION_JUMP_4 6

#define HERO_POSITION_JUMP_5 7

#define HERO_POSITION_JUMP_6 8
```

```
#define HERO_POSITION_JUMP_7 9
#define HERO_POSITION_JUMP_8 10
#define HERO_POSITION_RUN_UPPER_1 11
#define HERO_POSITION_RUN_UPPER_2 12

LiquidCrystal_I2C lcd(0x27, 16, 2);
static char terrainUpper[TERRAIN_WIDTH + 1]; // array for blocks of upper lcd matrix
static char terrainLower[TERRAIN_WIDTH + 1]; // array for blocks of upper lcd matrix
static bool buttonPushed = false; // variable for interrupts


void initializegraphics(){
  static byte graphics[] = {
    /* different character pixel data for hero positions and blocks*/
    //Hero run position 1
    B01100,
    B01100,
    B00000,
    B01110,
    B11100,
    B01100,
    B11010,
    B10011,
    //Hero run position 2
    B01100,
    B01100,
    B00000,
    B01100,
    B01100,
    B01100,
    B01100,
    B01110,
    // Jump
    B01100,
    B01100,
    B00000,
    B11110,
    B01101,
    B11111,
    B10000,
    B00000,
    // Jump lower
    B11110,
    B01101,
    B11111,
    B10000,
    B00000,
    B00000,
    B00000,
    B00000,
    // full block
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
    B11111,
    //  right block
    B00011,
    B00011,
    B00011,
    B00011,
    B00011,
    B00011,
    B00011,
    B00011,
    // left block
    B11000,
    B11000,
    B11000,
    B11000,
    B11000,
    B11000,
    B11000,
    B11000,
  };


// creating pixels for different orientations mentioned graphics array
  for (int i = 0; i < 7; ++i) {
    lcd.createChar(i + 1, &graphics[i * 8]);
  }
  // initialising arrays to empty blocks
  for (int i = 0; i < TERRAIN_WIDTH; ++i) {
    terrainUpper[i] = SPRITE_TERRAIN_EMPTY;
    terrainLower[i] = SPRITE_TERRAIN_EMPTY;
  }
}
// function for movement of blocks
void advanceTerrain(char* terrain, byte newTerrain){
  for (int i = 0; i < TERRAIN_WIDTH; ++i) {
    char current = terrain[i];
    char next ;
    if(i == TERRAIN_WIDTH-1){
      next =newTerrain ;}
    else{
        next=terrain[i+1];
      }
    switch (current){
      case SPRITE_TERRAIN_EMPTY:
        if(next == SPRITE_TERRAIN_SOLID){
          terrain[i] =SPRITE_TERRAIN_SOLID_RIGHT;
        }
        else {
          terrain[i] =SPRITE_TERRAIN_EMPTY;
        }
        break;
      case SPRITE_TERRAIN_SOLID:
        if(next == SPRITE_TERRAIN_EMPTY){
          terrain[i] =SPRITE_TERRAIN_SOLID_LEFT;
          }
         else{
          terrain[i] =SPRITE_TERRAIN_SOLID;
         }
        break;
      case SPRITE_TERRAIN_SOLID_RIGHT:
        terrain[i] = SPRITE_TERRAIN_SOLID;
        break;
      case SPRITE_TERRAIN_SOLID_LEFT:
        terrain[i] = SPRITE_TERRAIN_EMPTY;
        break;
    }
  }
}
// function for sounds
void music (int music_delay, int freq){
tone(3, freq); // buzzer pin 3
delay(music_delay);
noTone(3);
}
```

```
//Function for drawing hero postures, checking for collisions and score position
bool drawHero(byte position, char* terrainUpper, char* terrainLower, unsigned int score) {
  bool collide = false;
  char upperSave = terrainUpper[HERO_HORIZONTAL_POSITION];
  char lowerSave = terrainLower[HERO_HORIZONTAL_POSITION];
  byte upper, lower;
  switch (position) {
    case HERO_POSITION_OFF:
      upper = lower = SPRITE_TERRAIN_EMPTY;
      break;
    case HERO_POSITION_RUN_LOWER_1:
      upper = SPRITE_TERRAIN_EMPTY;
      lower = SPRITE_RUN1;
      break;
    case HERO_POSITION_RUN_LOWER_2:
      upper = SPRITE_TERRAIN_EMPTY;
      lower = SPRITE_RUN2;
      break;
    case HERO_POSITION_JUMP_1:
    case HERO_POSITION_JUMP_8:
      upper = SPRITE_TERRAIN_EMPTY;
      lower = SPRITE_JUMP;
      break;
    case HERO_POSITION_JUMP_2:
    case HERO_POSITION_JUMP_7:
      upper = SPRITE_JUMP_UPPER;
      lower = SPRITE_JUMP_LOWER;
      break;
    case HERO_POSITION_JUMP_3:
    case HERO_POSITION_JUMP_4:
    case HERO_POSITION_JUMP_5:
    case HERO_POSITION_JUMP_6:
      upper = SPRITE_JUMP;
      lower = SPRITE_TERRAIN_EMPTY;
      break;
    case HERO_POSITION_RUN_UPPER_1:
      upper = SPRITE_RUN1;
      lower = SPRITE_TERRAIN_EMPTY;
      break;
    case HERO_POSITION_RUN_UPPER_2:
      upper = SPRITE_RUN2;
      lower = SPRITE_TERRAIN_EMPTY;
      break;
  }
  if (upper != ' ') {
    terrainUpper[HERO_HORIZONTAL_POSITION] = upper;
    if(upperSave == SPRITE_TERRAIN_EMPTY){
      collide = false;
    }
    else {
      collide=true;
    }
  }
  if (lower != ' ') {
    terrainLower[HERO_HORIZONTAL_POSITION] = lower;
    if(lowerSave == SPRITE_TERRAIN_EMPTY){
      collide|=false;
    }
    else {
      collide |=true;
    }
  }

  byte digits = (score > 9999) ? 5 : (score > 999) ? 4 : (score > 99) ? 3 : (score > 9) ? 2 : 1;

  // Draw the scene
  terrainUpper[TERRAIN_WIDTH] = '\0';
  terrainLower[TERRAIN_WIDTH] = '\0';
  char temp = terrainUpper[16-digits];
  terrainUpper[16-digits] = '\0';
  lcd.setCursor(0,0);
  lcd.print(terrainUpper);
  terrainUpper[16-digits] = temp;
  lcd.setCursor(0,1);
  lcd.print(terrainLower);

  lcd.setCursor(16-digits,0);
  lcd.print(score);

  terrainUpper[HERO_HORIZONTAL_POSITION] = upperSave;
  terrainLower[HERO_HORIZONTAL_POSITION] = lowerSave;
  return collide;
}

// Handle the button push as an interrupt
void buttonPush() {
  buttonPushed = true;
```

```
}

// function for title display and startup music

void Title(){

  lcd.setCursor(0,0);

  lcd.print("RUN");

  lcd.setCursor(5,1);

  lcd.print("AND JUMP");

  HERO_POSITION_RUN_LOWER_1 ;

  music(100,400);

  music(100,300);

  music(100,200);

  music(100,300);

  music(100,400);

  music(100,600);

  delay(4000);

  lcd.clear();


}

// Actual game function, calling functions defined earlier and controlling game speed based on
mode selected

void game(int Jack_speed){

  static byte heroPos = HERO_POSITION_RUN_LOWER_1;

  static byte newTerrainType = TERRAIN_EMPTY;

  static byte newTerrainDuration = 1;

  static bool playing = false;

  static bool blink = false;

  static unsigned int distance = 0;

  if (!playing) {

    drawHero((blink) ? HERO_POSITION_OFF : heroPos, terrainUpper, terrainLower, distance >> 3);

    if (blink) {

      lcd.clear();

      lcd.setCursor(0,0);

      lcd.print("Press  JUMP");

      playing=false;

    }

    delay(250);

    blink = !blink;

    if (buttonPushed) {

      music(100,400);

      music(100,600);

      music(100,400);

      initializegraphics();

      heroPos = HERO_POSITION_RUN_LOWER_1;
      playing = true;

      buttonPushed = false;

      distance = 0;


    }

    return;

  }


  // Shift the terrain to the left

  advanceTerrain(terrainLower, newTerrainType == TERRAIN_LOWER_BLOCK ? SPRITE_TERRAIN_SOLID :
SPRITE_TERRAIN_EMPTY);

  advanceTerrain(terrainUpper, newTerrainType == TERRAIN_UPPER_BLOCK ? SPRITE_TERRAIN_SOLID :
SPRITE_TERRAIN_EMPTY);


  // Make new terrain to enter on the right

  if (--newTerrainDuration == 0) {

    if (newTerrainType == TERRAIN_EMPTY) {

      newTerrainType = (random(3) == 0) ? TERRAIN_UPPER_BLOCK : TERRAIN_LOWER_BLOCK;

      newTerrainDuration = 2 + random(10);

    } else {

      newTerrainType = TERRAIN_EMPTY;

      newTerrainDuration = 10 + random(10);

    }

  }


  if (buttonPushed) {


    if (heroPos <= HERO_POSITION_RUN_LOWER_2)

    music(100,300);

    heroPos = HERO_POSITION_JUMP_1;

    buttonPushed = false;


  }


  if (drawHero(heroPos, terrainUpper, terrainLower, distance >> 3)) {

    playing = false; // The hero collided with something. Too bad.

    music(100,500);

    music(100,400);

    music(120,300);

    music(160,400);

    lcd.clear();

  } else {

    if (heroPos == HERO_POSITION_RUN_LOWER_2 || heroPos == HERO_POSITION_JUMP_8) {

      heroPos = HERO_POSITION_RUN_LOWER_1;
    } else if ((heroPos >= HERO_POSITION_JUMP_3 && heroPos <= HERO_POSITION_JUMP_5) &&
terrainLower[HERO_HORIZONTAL_POSITION] != SPRITE_TERRAIN_EMPTY) {

      heroPos = HERO_POSITION_RUN_UPPER_1;

    } else if (heroPos >= HERO_POSITION_RUN_UPPER_1 && terrainLower[HERO_HORIZONTAL_POSITION] ==
SPRITE_TERRAIN_EMPTY) {

      heroPos = HERO_POSITION_JUMP_5;

    } else if (heroPos == HERO_POSITION_RUN_UPPER_2) {

      heroPos = HERO_POSITION_RUN_UPPER_1;

    } else {

      ++heroPos;

    }

    ++distance;


    //digitalWrite(PIN_AUTOPLAY, terrainLower[HERO_HORIZONTAL_POSITION + 2] ==
SPRITE_TERRAIN_EMPTY ? HIGH : LOW);

  }

  delay(Jack_speed);

  }


void setup(){

  pinMode(4, INPUT_PULLUP);

  pinMode(3, OUTPUT);

  pinMode(VRy, INPUT);

  digitalWrite(VRy, HIGH);

  select_mode = 1;

  attachInterrupt(0/*VRy*/, buttonPush, FALLING);//checking when joystick is pushed

  initializegraphics();


  lcd.begin();

  lcd.backlight();

   Title();

   do{menu();}while(digitalRead(2) == HIGH);//display menu until pin 2 is made low




}


// running game based on functions

void loop() {

switch (game_mode) {

case 1:

game(70);

break;

case 2:

game(50);

break;
```

```
case 3:
game(15);
break;
case 4:
game(1);
break;
    }
  }
//function for mode display based on interrupts
void menu() {

if (select_mode > 4 || select_mode < 0) {select_mode == 0;}
if (digitalRead(4) == LOW) { select_mode++;}
  switch (select_mode) {

    case 1:
    lcd.setCursor(0,0);
  lcd.print("Select mode:");
  lcd.setCursor(0,1);
  lcd.print("<     Easy     >");
  delay(500);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Select mode:");
  lcd.setCursor(0,1);
  lcd.print("<              >");
  delay(500);
  if(digitalRead(2) == LOW) {
    game_mode = 1;
    game_mode;}

  break;
    case 2:
lcd.setCursor(0,0);
  lcd.print("Select mode:");
  lcd.setCursor(0,1);
  lcd.print("<    Medium    >");
  delay(500);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Select mode:");
  lcd.setCursor(0,1);
  lcd.print("<              >");
  delay(500);
```

```
if(digitalRead(2) == LOW) {
    game_mode = 2;
    game_mode;}
    break;

    case 3:
lcd.setCursor(0,0);
  lcd.print("Select mode:");
  lcd.setCursor(0,1);
  lcd.print("<     Hard     >");
  delay(500);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Select mode:");
  lcd.setCursor(0,1);
  lcd.print("<              >");
  delay(500);
  if(digitalRead(2) == LOW) {
    game_mode = 3;
    game_mode;}

    break;
    case 4:
lcd.setCursor(0,0);
  lcd.print("Select mode:");
  lcd.setCursor(0,1);
  lcd.print("<  impossible  >");
  delay(500);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Select mode:");
  lcd.setCursor(0,1);
  lcd.print("<              >");
  delay(500);
  if(digitalRead(2) == LOW) {
    game_mode = 4;
    game_mode;}
    break;

  default:
select_mode = 0;
  break;
    }
  }
```