NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

IM3080 Design and Innovation Project

(AY2022/23 Semester 1)

Project Report

Title: Project Tetris (Interactive)

Supervisor: Prof Chua Hock Chuan

Submitted by: Group 4

Shawn Koh Jun Jie (Group Leader)	U2122707J
Sean Young Song Jie	U2122305F
Tan Khai Ming	U2122111B
Han Zhize	U2022119L
Zhang Wei	U2022549D
Yap Qi Long, Marcel	U2021667A
Teo Kai Xin, Hazel	U2021004C
Ding Man	U2022009E
Tan Zi Qi	U2021274E

PROJECT TETRIS

Abstract

Tetris is a game created by Soviet scientist Alexey Pajitnov in 1984. The aim of the game is to clear as many lines as possible using 7 different types of tiles also known as tetrominoes that are generated pseudo-randomly, without them stacking to the top. With tetris being a game that is highly compatible with the arcade-style of gameplay, a functional way to recreate the game would be to use a combination of LEDs and buttons.

PROJECT TETRIS

Content Page

Section	Section Name	Page
1	Project Background and Motivation	3
2	Objective	4
3	Review of Literature/Technology	4
4	Design and Implementation	18
5	Recommendation for Future Work	41
6	Conclusion	42
	References	43
	Appendices	44

PROJECT TETRIS

1 Project Background and Motivation

This interactive project was previously attempted by another group back before COVID. They used ICs and individual LEDs to control the flow of the tetrominoes based on certain analog inputs. However, the design was clunky and too many wires were required to make the project run smoothly. Based on the shortcomings of said project, we have decided to update the Tetris board by replacing the individual LEDs with addressable LED strips as well as using microcontrollers such as the Arduino Mega to assist with the program flow. The aim is to ensure that there will be a significant reduction in wiring as well as an improvement to the overall quality and feel of the game. Since we are using addressable LED strips, we would be required to apply certain levels of Data Structures and Algorithms to our project, allowing us to also apply what we learnt in school directly into the project.

1.1 Project Delegation

Team	**Main Job Scope***	**Members**
Hardware	In charge of wiring, soldering, circuit analysis and mounting of the board.	- Shawn Koh Jun Jie - Tan Khai Ming - Teo Kai Xin, Hazel
Software	In charge of writing the code for the Arduino which controls the overall flow of the game.	- Ding Man - Han Zhize - Zhang Wei - Sean Young Song Jie
Design	In charge of coming up with the design for the exterior of the board, as well as the overall theme	- Yap Qi Long, Marcel - Tan Zi Qi

*While the above states the main job scope of the various members in this team, members still assist in other job scopes should the need arise.

2 Objective

The aim of this project is to provide the user with a retro 80's arcade feel through our recreation of Tetris and have an immersive experience. Overall, the user should have an enjoyable time playing our recreation without feeling frustrated at the controls as well as having a dopamine boost when the user beats the high score.

3 Review of Literature/Technology

Before starting this project, background research has to be done on all fronts, including software, hardware and design. Some preparation regarding hardware also needed to be done. This section will cover the research and preparation done by the software and hardware team, as well as the design methodology.

3.1 Hardware Research and Preparation

This section will cover the research and preparation done in the hardware aspect of the project.

3.1.1 Electronic Circuits

We were able to accurately navigate the many circuits and wiring by applying fundamental knowledge on electronic devices and circuitry taught in IE2002 Analog Electronics. Precise analysis and design of electronic circuits, both discrete and integrated, were also performed. We calculated how many and what types of amplifiers and resistors could be connected to help with the circuit's power and current supply.

3.1.2 LEDs

We took things to the next level by upgrading our Tetris game screen from our predecessor's individual LEDs wiring configuration to using WS2812 addressable LED strips. As compared to its predecessor, the WS2812 addressable LED strips come with RGB capability that offers our software team the freedom to assign multiple colours for the tile shapes. By being addressable, our software team was able to treat each LED as an individual and code them based as such, which is much easier than using a non-addressable LED strip. In addition, the LED strip generates much less heat than its traditional Dual In-line Package (DIP) variant.

3.1.3 Hardware Preparation

The Hardware Team cut and measured the LED strips to the exact number of LEDs required per strip before soldering wires to each contact point, namely the 5V wire, data wire and ground wire. We colour coded each wire for easy future reference, using red for 5V, green for data and black for ground.

Following that, we applied a similar methodology to connect the buttons and wires. To ensure an efficient and effective connection, all components including by not limited to wires and LED strips were precisely measured and cut. We also added an insulation layer of hot glue and heat shrink case to prevent unnecessary disconnection and reduce the chance of the occurrence of shorting respectively.

3.2 Software Literature and Preparation

This section will cover the research and preparation done in the software aspect of the project.

3.2.1 Installation

This project requires the use of an Arduino. In order to implement the project, we would first require the Arduino IDE, which can be downloaded at <https://www.arduino.cc/en/software>.

3.2.2 Self-Learning

Following the EEE-Arduino_Online_Workshop on NTULearn, we started with basic concepts and syntax of Arduino language. Then, we tried some mini projects, such as LED blinking, Buzzer controlling, and the usage of resistor, sensor and DC motor.

To start off, we first visited Prof Chua Hock Chuan's website <https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaGame_Tetris.html> and learned about the basic logic behind the Tetris algorithm:

As Prof Chua's algorithm is written in Java and Arduino is mainly written in something more akin to C/C++, we used Prof Chua's algorithm as a reference and developed our own algorithm.

Before diving into programming the algorithm, we visited Arduino's official website<https://docs.arduino.cc/> to familiarise the syntax for declaring functions, variables and the overall structure of Arduino programming language, which helps us develop the features in the project.

3.2.3 Feature Collection

To become familiar with the Tetris game, we tried an online Tetris game <https://tetris.com/play-tetris/> and used that as a guideline to design our own set of rules and gameplay.

We listed down all features required in our project, including the different tile shapes, tile movement, tile rotation, soft drop, hard drop, stack, line clear, score system, difficulty level and background music with effects.

3.2.4 Simulation using TinkerCAD

Due to delivery time and the constant flux of our inventory during the first few weeks, there lacked a meaningful way to test our code using hardware. Thus we went to TinkerCAD, an online circuit simulator tool so we could visualise the output of our codes. <https://www.tinkercad.com/>

Since Tinkercad is an online simulator tool, it has its limitations. We need to figure out a way to import and use the LED Strips as it's not part of their basic components. It also has a maximum text section of about 500 lines. Till the limit is reached, we will make use of it to test our code.

3.2.5 Migration

After the hardware was successfully set up, we migrated the codes from the simulator into Arduino's IDE and uploaded it. After testing, there are some things that happened that the simulator didn't occur, which we fixed accordingly later on. From here on, we constantly tested our codes physically and debugged them live.

3.3 Design Methodology

This section will cover how the design team came up with the design and theme for the tetris box.

3.3.1 Brainstorming

The Design Team researched various different themes of Tetris Games, both virtual games and physical consoles. The team used Figma and compiled all relevant screenshots of possible themes. A mood board was created.

From this, we shortlisted some colours and fonts and after some deliberation, we decided the background was to be dark blue with white stars as well as some decorations related to the space theme like rocket ships and UFOs.

3.3.2 Creating Designs in Software

The font chosen was Space Obsessed as it had an arcade vibe and yet felt futuristic so it fit our design very nicely. The following image shows how we designed different font sizes on Figma and were printed to be cut out.

Figma was used to design the UFOs and rocket ships, as shown in the following figure. We wanted to keep the design simple yet colourful so it wouldn't distract from the main game but still give the box a bit of life. We asked the team what their favourite animal was and we put some of those animals into the UFOs and rocket ship designs.

PROJECT TETRIS

We decided to create a title card for the box to show everyone was the project name was. The background followed the dark blue space theme, while for the letters, we chose the primary colours of classic tetris blocks: Red, Orange, Yellow, Green, Blue. Purple. These colour values were directly sampled from classic tetris games. We decided to go for a more blocky letter design to attract attention while keeping in line with the tetris “block” theme. We made a few drafts in Figma but eventually decided to land on one.

We also had an idea to print, fold and glue 3D paper cubes together to form 3D tetris blocks and paste them on the box. Thus, a cutout for the cubes was designed in Illustrator and the colours still followed the theme. After much trial and error, we found out 3cm was the right length for the sides of the cubes and it would be easier to print a whole sheet of colour.

3.3.3 Implementation of Designs

The design team ordered paint and paint brushes to paint the Tetris Box: 4 bottles of black paint, 1 bottle of white paint and 2 bottles of blue paint. Blue and black paint were mixed to create the deep dark blue we needed and we dipped a paintbrush in white paint and tapped it with another paintbrush to create the star effect on the box.

PROJECT TETRIS

A white box was drawn around the 2 small holes to create a more futuristic effect. The letters for the “High score”, “Score” and “Next Tile” were printed and cut out and pasted on the box. The team then left it to dry over the weekend.

PROJECT TETRIS

The title card was printed out and pasted with glue onto a piece of plastic coated cardboard and glued with supports so it would be able to stand on its own on top of the box.

PROJECT TETRIS

3D paper cubes were printed, cut and pasted together to form a wide variety of different coloured cubes to be pasted on or on top of the box. They were pasted together.

3.3.4 Assembly

Once all the different components were completed, the team assembled the final design, along with the hard components.

4 Design and Implementation

This section will cover both software and hardware design and implementation of the project.

4.1 Important Hardware used in Project

	<u>Arduino Mega 2560 Rev3</u> - More processing power than an Arduino Uno which is more feasible to run the addressable LED strip - More pins than the Arduino Uno - Used as a microcontroller to program the LED strip
	<u>WS2812 Addressable LED Strip</u> - Requires only 5V to run - Being addressable allows the individual LEDs in the strip to be programmed

	<u>LED Lighted Arcade Push Button (60mm)</u> - Implemented 2 buttons. Both comes in white colour - One is used for Start Game while the other enables instant tile placement
	<u>LED Lighted Arcade Push Button (45mm)</u> - Implemented 4 buttons. Comes in red, blue, green and yellow colour. - Red(Up) - Rotate tile - Blue(Right) - Moves tile to the right - Green(Down) - Speeds up falling speed - Yellow(Left) - Moves tile to the left
	<u>Kingbright SA23-12SRWA 7-Segment Display</u> - Requires 8V to operate for number segments, 4V for decimal point - To display game scores for both current score and high score - Common Anode Configuration

PROJECT TETRIS

	<u>MicroSD Card Adapter</u> - Enables playback of Tetris game background music - Plays WAV files stored inside the Micro SD card - Utilises SPI interface via the file system driver to read and write MicroSD card files
	<u>LCN-210 Desktop Wireless Speaker</u> - Main audio generator for Tetris game background music - Works collaboratively with MicroSD Card Adapter - AUX mode connection (3.5mm audio jack to Arduino GPIO pin)
	<u>8ohm 0.5W Mini Speaker (40mm)</u> - Main audio generator for Tetris game score point sound - Implemented 2 speakers for both Left and Right channels for an immersive experience

	<u>PAM8403 Class-D Audio Amplifier Board</u> - Works collaboratively with both mini speakers to amplify its output audio - Volume of speakers can be adjusted or switched off with the in-built potentiometer - Operates on 5V

4.2 Design Theme

“Space” Theme with a variety of colours which stand out against the dark blue background of the box.

4.3 Software Implementation

This section will discuss each part of the code with regards to their use in the overall project.

4.3.1 Declaring and Initialising variables

Description	Sample code
We'll be using Adafruit Neopixel library to manipulate the addressable LED strips. Firstly, we have to declare the Adafruit Neopixel and assign the following arguments: →	`Adafruit_NeoPixel col1(NUMPIXELS, PIN, NEO_GRB + NEO_KH` Argument 1 = Number of pixels in NeoPixel strip Argument 2 = Arduino pin number (most are valid) Argument 3 = Pixel type flags: ● NEO_KHZ800 ○ 800 KHz bitstream (most NeoPixel products w/WS2812 LEDs) ● NEO_GRB ○ Pixels are wired for GRB bitstream (most NeoPixel products)

To simplify the colour assignment,
 	
We've declared colors as enum class and initialising it.
 	
It is for easy calling by using readable text instead of RGB/hexadecimals.
	`enum Colors : long`
	`{`
	`  BLACK = 0x000000,`
	`  WHITE = 0xFFFFFF,`
	`  RED = 0xFF0000,`
	`  BLUE = 0x00FFFF,`
	`  GREEN = 0x00FF00,`
	`  YELLOW = 0xFFFF00,`
	`  PURPLE = 0xFF00FF,`
	`};`
	
	`Colors colorL = Colors::RED;`

Same goes with this.
 	
Declaring audio state as enum and initialising it
 	
PROGMEM - store a table of strings in program memory (flash)
 	
This is for easy calling when further down the code.
	`enum State`
	`{`
	`  game_over,`
	`  add_point,`
	`  background,`
	`  increase,`
	`  decrease,`
	`  mute_bg,`
	`  unmute_bg,`
	`  stop_all,`
	`  default_null`
	`} audio_state;`
	`  You, 4 days ago • Implemented instant drop and Add`
	`static const char wav_gameOver[] PROGMEM = "go3.wav";`
	`static const char wav_backGround[] PROGMEM = "bgm1.wav";`
	
	`const char *wav_table[] = {`
	`  wav_gameOver,`
	`  wav_backGround,`
	`};`

We need to find a way to store the tiles somewhere to be able to keep track of the progress. Thus declaring 2D array to record the light and dark states for each pixel	`int lightArray[38][10];`

Considering we need to manually set which pins to on for the 7 segments to display the number we want. To make life easier, We've placed all the conditions on which LEDs to on and off to display specific numbers by declaring a 2D array and initialising for the 7-segment	

```
const int numbers[10][7] =
    {
        {0, 0, 0, 0, 0, 0, 1}, // 0
        {1, 1, 0, 0, 1, 1, 1}, // 1
        {0, 0, 1, 0, 0, 1, 0}, // 2
        {1, 0, 0, 0, 0, 1, 0}, // 3
        {1, 1, 0, 0, 1, 0, 0}, // 4
        {1, 0, 0, 1, 0, 0, 0}, // 5
        {0, 0, 0, 1, 0, 0, 0}, // 6
        {1, 1, 0, 0, 0, 0, 1}, // 7
        {0, 0, 0, 0, 0, 0, 0}, // 8
        {1, 0, 0, 0, 0, 0, 0}, // 9
};
```

4.3.2 Setup

Description	Sample code

Setting up specific PINS as an INPUT_PULLUP
 	
We decided to use INPUT_PULLUP instead of the normal INPUT is because we can reduce all the data pins into just 1.
	`pinMode(BUTTON_LEFT, INPUT_PULLUP);`
	`pinMode(BUTTON_RIGHT, INPUT_PULLUP);`
	
	`----NOTE----`
	`INPUT_PULLUP inverts the input of the buttons!!!`
	`LOW -> PRESSED`
	`HIGH -> RELEASED`

Initialize the pseudo-random sequence	`randomSeed(analogRead(0));`

Initialising the lightArray 2D array with 0	`for (int i = 0; i < 38; i++)`
	`{`
	`  for (int j = 0; j < 10; j++)`
	`  {`
	`  lightArray[i][j] = 0;`
	`  }`
	`}`

Setting up specific pins as OUTPUT for the 7 segments	`for (int x = 0; x < 7; x++)`
	`{`
	`  pinMode(ones_score_pin[x], OUTPUT);`
	`  pinMode(tens_score_pin[x], OUTPUT);`
	`  pinMode(ones_highscore_pin[x], OUTPUT);`
	`  pinMode(tens_highscore_pin[x], OUTPUT);`
	`}`

Setting up pins for speaker and buzzer for background music and sound effect.	`audio.speakerPin = 5; // 5,6,11 or 46 on Mega, 9 on Uno, Nano, etc`
	`pinMode(2, OUTPUT);  // Pin pairs: 9,10 Mega: 5-2,6-7,11-12,46-45`
	
	`pinMode(buzzer, OUTPUT);`
	
	`audio.setVolume(3);`
	`audio.quality(1);`

Set volume between 0 to 7 Quality = 1 for 2x oversampling	

Check if the SD card is properly accessible by the SD library	`if (!SD.begin(SD_ChipSelectPin))`
	`{`
	`  return;`
	`}`
	`else`
	`{`
	`  Serial.println("SD OK");`
	`  audio_state = default_null;`
	`}`

Before we can start using the Neopixel strip object, we have to first initialise it	`col1.begin();`

4.3.3 Loop

Description	Sample code
We'll chose to set the brightness to 10 within the range of 0 to 255 as we'll use the circuit power to determine the final brightness of it	`col1.setBrightness(10);`

To allow the player to get ready before starting the game, we've create 2 blocks to execute the specific action accordingly if the game is active or not
 	
Also once the game starts, the background music will start to play infinitely until game over.
	`if (!isGameStarted)`
	`{`
	`  initGame();`
	`  currentStateStart = digitalRead(BUTTON_START);`
	`  if (currentStateStart == LOW)`
	`  {`
	`  startGame();`
	`  randomShape(randNumber);`
	`  }`
	`}`
	`else`
	`{`
	`  if (!audio.isPlaying())`
	`  {`
	`  audio_state = background;`
	`  }`
	`  isGameOver = false;`
	
	`  randomShape(randNumber);`
	`}`

Constantly updates what audio needs to play and display the correct score and high score	`audioControl();`
	
	`displayScore(score);`
	`displayHighscore(highestScore);`

4.3.4 Initialise Game function

Description	Sample code

We've decided to light up all the LED in the main board to white as an indication of "game not started" or "game over"	`for (int i = 0; i < 38; i += 2)`
	`{`
	`  for (int j = 0; j < 10; j++)`
	`  {`
	`  switch (17 - j)`
	`  {`
	`  case PIN:`
	`  col1.setPixelColor(i, Colors::WHITE);`
	`  break;`
	`  ...`
	`  }`
	`  }`
	`}`

Likewise, this is to turn all the LED in the "next tile" board to white as an indication	`for (int i = 0; i <= 6; i += 2)`
	`{`
	`  colPL.setPixelColor(i, Colors::WHITE);`
	`  colPR.setPixelColor(i, Colors::WHITE);`
	`}`

Call the specific functions to update the strips	`showCol();` `showNextTileCol();`

4.3.5 Start Game function

Description	Sample code
Set the delay value to 500 milli seconds for the tetris speed Set the count_duplicate to 0 for counting the amount of same tiles is generated in a row	`DelayVal = 500;` `count_duplicate = 0;`

Before starting a new game, we need to clear and switch off every single LED that is on by turning all LED in the main board to black	`for (int i = 0; i < 38; i += 2)`
	`{`
	`  for (int j = 0; j < 10; j++)`
	`  {`
	`  switch (17 - j)`
	`  {`
	`  case PIN:`
	`  col1.setPixelColor(i, Colors::BLACK);`
	`  break;`
	`  ...`
	`  }`
	`  }`
	`}`

Likewise, to reset by turning all the LED on the “next tile” board to black	`for (int i = 0; i <= 6; i += 2)`
	`{`
	`  colPL.setPixelColor(i, Colors::BLACK);`
	`  colPR.setPixelColor(i, Colors::BLACK);`
	`}`

Clear and reset the lightArray 2D array back to 0	`for (int i = 0; i < 38; i++)`
	`{`
	`  for (int j = 0; j < 10; j++)`
	`  {`
	`  lightArray[i][j] = 0;`
	`  }`
	`}`

4.3.6 Generate Number function

Description	Sample code

We'll generate 2 tiles at the start of the game.
 	
1 number for the current tile and the other number to the next tile.
 	
Subsequently, we'll use the pre-generated number of the next tile and set it to the current tile then generate a new tile for the next tile.
	`if (randNumber == -1)`
	`{`
	`  randNumber = random(0, 5);`
	`}`
	`else`
	`{`
	`  randNumber = randNumberNext;`
	`}`

Since we only got 5 tiles to randomise from, the probability of getting the same number more than twice or even thrice is very high.
 	
Thus number for next tile will keep regenerating if it is the same tile for more than 2 times in a row.
	`do`
	`{`
	`  randNumberNext = random(0, 5);`
	`  if (randNumber == randNumberNext)`
	`  {`
	`  count_duplicate++;`
	`  }`
	`  if (randNumber != randNumberNext)`
	`  {`
	`  count_duplicate = 0;`
	`  }`
	`} while (count_duplicate >= 2);`

4.3.7 Display Next Tile function

Description	Sample code

Before displaying the next tile, we have to switch off all the LEDs in the “next tile” board to prevent overlapping.	`for (int i = 0; i <= 6; i += 2)`
	`{`
	`  colPL.setPixelColor(i, Colors::BLACK);`
	`  colPR.setPixelColor(i, Colors::BLACK);`
	`}`

After switching the LEDs off, we’ll call the specific function to light the correct shape for the next tile	`switch (randNumberNext)`
	`{`
	`case 0:`
	`  lightOPre();`
	`  break;`
	`...`
	`}`

4.3.8 Update Speed function

Description	Sample code

We have to first detect the “soft drop” input button pressed and then reduce the delay by 300.
 	
But since we have the difficulty level where we’ll constantly reduce the delay, we need to make sure the delay doesn’t exceed 0.
	`if (SpeedUp == LOW)`
	`{`
	`  if (DelayVal - 300 > 0)`
	`  {`
	`  delay(DelayVal - 300);`
	`  }`
	`  else`
	`  {`
	`  delay(DelayVal);`
	`  }`
	`}`
	`else`
	`{`
	`  delay(DelayVal);`
	`}`

4.3.9 Update Score function

Description	Sample code
Increase the current score by 1	`score += 1;`

We've implemented a difficulty feature where we'll
- Decrease the Tetris current delay by 50 when score = 4
- Continueusly decrease Tetris current delay by 40 every single score after 8.
	`if (score == 4)`
	`{`
	`  DelayVal -= 50;`
	`}`
	`else if (score >= 8)`
	`{`
	`  if (DelayVal - 40 >= 10)`
	`  {`
	`  DelayVal -= 40;`
	`  }`
	`}`

4.3.10 Random Shape function

Description	Sample code

Before allowing the current tile to be displayed, we need to check the boundaries to make sure there are available space for the specific tile.
 	
If the conditions are not met, game over will be set to true
	
	switch (currentShape)
	{
	case 0:
	if (lightArray[2][4] == 0 && lightArray[2][5] == 0 &&
	lightArray[0][0] == 0 && lightArray[0][1] == 0 &&
	lightArray[0][2] == 0 && lightArray[0][3] == 0 &&
	lightArray[0][6] == 0 && lightArray[0][7] == 0 &&
	lightArray[0][8] == 0 && lightArray[0][9] == 0)
	{
	shape0();
	}
	else
	{
	isGameOver = true;
	}
	break; You, 4 days ago • Fixed warnings …
	...
	}

If game over is set to true, the endGame function will be called instead of displaying the shape	
	if (isGameOver)
	{
	endGame();
	}

4.3.11 End Game function

Description	Sample code

Stoping all audio and play the game over sound as an indication of game over	`audio_state = stop_all;`
	`audioControl();`
	`audio_state = game_over;`
	`audioControl();`

Re-initialise the board to prepare for the next game	`initGame();`

Comparing if the current score is higher than the high score before replacing high score	`if (highestScore <= score)`
	`{`
	`highestScore = score;`
	`}`

4.3.12 Shape Function

Description	Sample code

Overview of the entire function	`void shapeL()`
	`{`
	`  centery1b = centery1;`
	`  centery2b = centery2;`
	`  centery3b = centery3;`
	`  centery4b = centery4;`
	`  lightL(centerx);`
	`  showCol();`
	`  currentStateL = digitalRead(BUTTON_LEFT);`
	`  currentStateR = digitalRead(BUTTON_RIGHT);`
	`  currentRotateState = digitalRead(BUTTON_ROTATE);`
	`  currentStateDrop = digitalRead(BUTTON_DROP);`
	
	`  if (currentL == 0)`
	`  {`
	`  <move left>`
	`  <move right>`
	`  <rotate>  You, 1 second ago • Uncommitte`
	`  }`
	`  else if (currentL == 1)`
	`  {`
	`  .....`
	`  }`
	`  .....`
	`  <change speed>`
	`  if (currentL == 0)`
	`  {`
	`  <automatically drop>`
	`  <instant drop>`
	`  <stack>`
	`  }`
	`  else if (currentL == 1)`
	`  {`
	`  .....`
	`  }`
	`  .....`
	`}`

Use centerx and centery1, etc, to track the position of the current shape.
 	
Each shape starts from centerx = 0.
 	
Centerx is incremented by the value of speed.
	`// move left`
	`if (currentStateL == LOW &&`
	`  (lightArray[centerx + SPEED][17 - centery3 - 1] == 0) &&`
	`  (lightArray[centerx + 2 + SPEED][17 - centery3 - 1] == 0) &&`
	`  (lightArray[centerx + 4 + SPEED][17 - centery3 - 1] == 0))`
	`{  You, 4 days ago • Implemented rotate and speed down shape`
	
	`  if (centery3 < PIN_LEFT3)`
	`  {`
	`  centery1b = centery1;`
	`  centery2b = centery2;`
	`  centery3b = centery3;`
	`  centery4b = centery4;`
	`  centery1 += 1;`
	`  centery2 += 1;`
	`  centery3 += 1;`
	`  centery4 += 1;`
	`  }`
	`}`

Detect the boundaries by checking the lightArray to move left or right if the moving button is triggered and change the centery.	
If the rotate button is triggered, detect the boundaries by checking lightArray to rotate to the next state of the shape.	
Stack at the bottom or on the previous shapes by detecting the lower boundaries according to lightArray and the lightArray is updated.	
If the speedUp button is triggered, the delay will decrease and the shape will move downwards faster.	

```
// rotate
if ((currentRotateState == LOW) &&
    (centery2 > PIN_RIGHT3) &&
    (lightArray[centerx + 2][17 - centery2] == 0) &&
    (lightArray[centerx + 2][17 - centery1] == 0))
{
    // turnoff current state
    turnoffL(centerx);
    // light next state
    currentL = (currentL + 1) % 4;
    lightL(centerx);
}
```

```
if ((centerx < NUMPIXELS - 6) &&
    (lightArray[centerx + 4 + SPEED][17 - centery2] == 0) &&
    (lightArray[centerx + 4 + SPEED][17 - centery3] == 0))
{
    turnoffL(centerx);
    centerx = centerx + SPEED;
    showCol();
}       You, 4 days ago • Implemented rotate and speed down sh
else
{
    lightArray[centerx][17 - centery3] = 1;
    lightArray[centerx + 2][17 - centery3] = 1;
    lightArray[centerx + 4][17 - centery3] = 1;
    lightArray[centerx + 4][17 - centery2] = 1;

    centerx = 0;
    centery1 = PIN;
    centery2 = PIN_2;
    centery3 = PIN_3;
    centery4 = PIN_4;
    checkLine();

    showCol();
    generateNumber();
    currentL = 0;
}
```

```
updateSpeed();
```

If the instantDrop button is triggered, we repeat the movingDown function without actually showing the light on the board until it stops.	

```
while ((centerx < NUMPIXELS - 6) &&
        (lightArray[centerx + 4 + SPEED][17 - centery2] == 0) &&
        (lightArray[centerx + 4 + SPEED][17 - centery3] == 0))
{
    centerx = centerx + SPEED;
}       You, 4 days ago • Implemented instant drop and Added curre
lightArray[centerx][17 - centery3] = 1;
lightArray[centerx + 2][17 - centery3] = 1;
lightArray[centerx + 4][17 - centery3] = 1;
lightArray[centerx + 4][17 - centery2] = 1;
lightL(centerx);
centerx = 0;
centery1 = PIN;
centery2 = PIN_2;
centery3 = PIN_3;
centery4 = PIN_4;
checkLine();

showCol();
generateNumber();

currentL = 0;
```

4.3.13 Light On/Off Function

Description	Sample code
Ulitize setPixelColor function to control the light condition and color of the shape. turnoff function is to change the color0 into black	

```
switch (centery1)
{
case PIN:
    col2.setPixelColor(centerx, color0);
    col2.setPixelColor(centerx + 2, color0);
    col3.setPixelColor(centerx, color0);
    col3.setPixelColor(centerx + 2, color0);
    break;
case PIN_2:
    ...
    break;
...
}
```

4.3.14 Check Line function

Description	Sample code

Since the value 0 indicates empty, any product with a 0 will results as 0.
 	
Thus, we calculate the product of the lightArray value of 1 line and update the product array.
 	
If the product > 0, means that line is filled
If the product = 0, means that line have empty slots.
	`for (int i = 36; i > 0; i = i - 2)`
	`{`
	`  product = 1;`
	`  for (int j = 0; j < 10; j = j + 1)`
	`  {`
	`  product = product * lightArray[i][j];`
	`  }`
	`  if (product != 0)`
	`  {`
	`  productArray[i] = product;`
	`  }`
	`}`

Call the blink function to do the blink effect of the non zero line	`blink(productArray);`

This is to prevent any cases of audio overlapping if more than 1 line is cleared
 	
Update the lightArray
	`updateScore();`
	`if (!playAudioOnce)`
	`{`
	`  audio_state = add_point;`
	`  audioControl();`
	`  playAudioOnce = true;`
	`}`
	
	`arrayUpdate(i);`

4.3.15 Blink function

Description	Sample code
To allow the player to see which line(s) cleared, we implemented a blink feature to slow down the line clear and make it more visually appealing. It will blink 2 times of the line that has a product of a non 0 value	

```
for (int blinktime = 0; blinktime < 4; blinktime++)
{
    for (int i = 0; i < 38; i = i + 2)
    {
        if (productArray[i] != 0)
        {
            if (blinktime % 2 == 0)
            {
                col1.setPixelColor(i, Colors::BLACK);
                ...
            }
            else
            {
                col1.setPixelColor(i, Colors::WHITE);
                ...        You, 1 second ago • Uncommit
            }
        }
    }
    showCol();
}
```

4.3.16 Array Update function

Description	Sample code
After blinking, we need to replace the line with product a non-zero value, with everything above. By doing so, we need to move all the tiles on the board down by replacing the non-zero line	
To complete the shift down, we need to reset the above back to 0	

```
for (int i = line; i > 0; i = i - 2)
{
    for (int j = 0; j < 10; j = j + 1)
    {
        lightArray[i][j] = lightArray[i - 2][j];
    }
}
```

```
for (int j = 0; j < 10; j = j + 1)
{
    lightArray[0][j] = 0;
}
```

4.3.17 Displaying Score/High Score function

Description	Sample code

Since the 7-segment can only display a single digit, we need to split the value in 2 especially if it's a double digit.
 	
Assume the max is only 99, a 2 digit number:
 	
By modulating 10, we'll be able to extract the ones position out.
 	
By dividing 10, we'll be able to extract the tens position out
	`int ones = 0;`
	`int tens = 0;`
	`if (score <= 9)`
	`{`
	`  ones = score;`
	`}`
	`else if (score > 9)`
	`{`
	`  ones = score % 10;`
	`  tens = score / 10;`
	`}`

After extraction, we'll plug it into the 7 segments array pre-initialised at the very top to turn on the specific 7 segments pin to display the ones and tens	`for (int j = 0; j < 7; j++)`
	`{`
	`  digitalWrite(ones_score_pin[j], numbers[ones][j]);`
	`  digitalWrite(tens_score_pin[j], numbers[tens][j]);`
	`}`

4.3.18 Audio Control function

Description	Sample code

This is how the specific audio is played by detecting what state it's supposed to be in.
 	
To play the specific audio, we need to first extract the audio PROGMEM string from the table pre-initialised at the very top and copy it into a char array.
 	
Pass the char array as a parameter to play the audio.
	`char wavFile[33];`
	`switch (audio_state)`
	`{ You, 4 days ago • Added backg`
	`case game_over:`
	`  strcpy_P(wavFile, wav_table[0]);`
	`  audio.play(wavFile);`
	`  break;`
	`case background:`
	`  ...`
	`  break;`
	`}`
	`audio_state = default_null;`

5 Recommendation for Future Work

Due to the lack of time, we were unable to finish up certain features that were planned at the start of the project. Such features can be implemented for future work to improve on the project which includes but not limited to:

- Saving a tile for future use within the game
- Using a Kinect sensor to control the tiles via machine learning to read the body posture of player in order to determine the movement of the tetris tiles
- Using a drum and pressure sensor to control the movement of the tiles when hit
- Making the design of the hardware more modular for easy maintenance
- Minor bug fixes with the code as well as improving the responsiveness of the game

6 Conclusion

In conclusion, this DIP project has allowed us to apply what we have learnt so far within the curriculum of IEM in Nanyang Technological University. By engaging in this project, we have found a way to apply both our knowledge in hardware and software in a practical capacity to create a game for everyone to enjoy.

References

https://edition.cnn.com/style/article/tetris-video-game-history/index.html#:~:text=Like%20many%20of%20history's%20greatest,a%20puzzle%20from%20his%20childhood

https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaGame_Tetris.html

https://docs.arduino.cc/

Appendices

A. Bill of Materials (BOM) – “All spending” AND “for the project prototype”

BOM - All Spending

S/N	ITEM	QTY	UNIT COST (SGD)	TOTAL COST (SGD)	REMARKS
1	Arduino Mega 2560 (Rev3)	1	$72.16	**$72.16 (incl. shipping)**	
2	Arduino 45-in-1 Sensor Starter Kit	1	$33.00	**$33.00 (incl. shipping)**	
3	MicroSD Card (32GB)	1	$6.99	**S$8.48 (incl. shipping)**	
4	LED Strip WS2812 (5m)	1	$13.50	**$14.50 (incl. shipping)**	
5	Round Push Button (Big)	2	$2.26	**$5.52 (incl. shipping)**	
6	Round Push Button (Small)	4	$4.50	**$19.49 (incl. shipping)**	
7	7-Segment LED Display (SA23-12SRWA)	4	$15.59	**$62.99 (incl. shipping)**	
8	Wireless Speaker	1	$15.00	**$16.49 (incl. shipping)**	
9	Small Speaker	4	$0.60	**$3.40 (incl. shipping)**	

10	PAM8403 Audio Amplifier	5	$0.75	**$4.74 (incl. shipping)**	
11	Corrugated Board	2	$8.15	**$16.30 (incl. shipping)**	
12	Paintbrush	4	$1.35	**$5.40 (incl. shipping)**	
13	Paint	1	$50.68	**$50.68 (incl. shipping)**	
14	Acrylic (Opal white)	1	$8.30	**$8.30 (incl. shipping)**	
15	Acrylic (Matt Barley)	1	$12.45	**$12.45 (incl. shipping)**	

BOM – For the Project Prototype

S/N	ITEM	PURPOSE	QTY	UNIT COST (SGD)	TOTAL COST (SGD)	REMARKS
1	Arduino Mega 2560 (Rev3)	MCU Board	1	$72.16	**$72.16 (incl. shipping)**	
2	Arduino 45-in-1 Sensor Starter Kit	Familiarise ourselves with Arduino	1	$33.00	**$33.00 (incl. shipping)**	Only certain parts are used from this kit
3	MicroSD Card Module	Read microSD card	1	-	**-**	Taken from 45-in-1 Sensor Starter Kit
4	MicroSD Card (32GB)	Store audio files	1	$6.99	**S$8.48 (incl. shipping)**	
5	LED Strip WS2812 (5m)	Game main and next tile screen display	2	$13.50	**$14.50 (incl. shipping)**	1 provided by school 1 purchased

BOM – For the Project Prototype

6	Round Push Button (Big)	Start game and instant drop tile button	2	$2.26	**$5.52 (incl. shipping)**	
7	Round Push Button (Small)	Move tiles left, right, rotate and speedup respectively	4	$4.50	**$19.49 (incl. shipping)**	
8	7-Segment LED Display (SA23-12SRWA)	Display 2-digit current and high score numerically	4	$15.59	**$62.99 (incl. shipping)**	
9	9V External Power Supply Plug	Power up 7-segment displays	1	-	**-**	Provided by school
10	5V External Power Supply Plug	Power up LED WS2812 strips	1	-	**-**	Provided by self
11	Wireless Speaker	Output audio for BGM and game over	1	$15.00	**$16.49 (incl. shipping)**	
12	Small Speaker	Output tone for scoring point	2	$1.20	**$2.20 (incl. shipping)**	
13	PAM8403 Audio Amplifier	Amplify small speaker audio	1	$0.75	**$4.74 (incl. shipping)**	
14	Breadboard (Small)	Affix components and build circuits	3	-	**-**	Provided by school
15	Breadboard (Big)	Affix components and build circuits	1	-	**-**	Provided by school
16	Jumper Wires	To connect circuit connections	100 +/-	-	**-**	Provided by school
17	1-inch Masking Tape (roll)	Label components and secure wirings	1	-	**-**	Provided by self
18	Corrugated Board	For mounting the LEDs to be displayed	2	$8.15	**$16.30 (incl. shipping)**	

BOM – For the Project Prototype

19	Paintbrush	Used to paint the Tetris Box	4	$1.35	**$5.40 (incl. shipping)**	
20	Paint	Used to paint the Tetris Box	1	$50.68	**$50.68 (incl. shipping)**	
21	Acrylic (Opal white)	Used as a cover for the LEDs to diffuse the light and fill up a square of light instead of a point	1	$8.30	**$8.30 (incl. shipping)**	
22	Acrylic (Matt Barley)	Used as a cover for the LEDs to diffuse the light and fill up a square of light instead of a point	1	$12.45	**$12.45 (incl. shipping)**	Only a small portion of the acrylic was used due to limited transparency

B. Design Diagrams

Schematic Diagram:

Pinout Diagram:

3D Modelling

Front View:

Side View:

PROJECT TETRIS

Plan View:

Isometric View:

C. Flowchart

Setup
Initialise Game
Start Game Button pressed?
No
Yes
Start Game
Generate random number
Sufficient space for tile?
No
Game Over
Is Score>Highscore?
Yes
No
Update Highscore
Yes
Display tile
is rotate button pressed?
is left button pressed?
is right button pressed?
is soft drop button pressed?
is hard drop button pressed?
Collision Detected?
No
Rotate
Move left
Move right
Soft drop
Hard drop
Reached the bottom?
No
Yes
Update array
is there line to clear?
No
Yes
Blink
Clear line
Update score

D. Project Workflow using Trello

E. Source Code

```
#include <Adafruit_NeoPixel.h>
#include <SD.h>
#include <SPI.h>
#include <TMRpcm.h>
#include <Beacon.h>
#include <toneAC.h>

#ifdef __AVR__
#include <avr/power.h> // Required for 16 MHz Adafruit Trinket
#endif
#define PIN 11
#define NUMPIXELS 38
#define SPEED 2
#define PIN_2 12
#define PIN_3 13
#define PIN_4 14
#define PIN_LEFT1 15
#define PIN_LEFT2 16
#define PIN_LEFT3 17
#define PIN_RIGHT1 10
```

```
#define PIN_RIGHT2 9
#define PIN_RIGHT3 8
#define BUTTON_LEFT 19
#define BUTTON_RIGHT 20
#define BUTTON_ROTATE 21
#define BUTTON_SPEEDUP 18
#define BUTTON_START A0
#define BUTTON_DROP 3
// new pins for shape prediction
#define PINPREL A6
#define PINPRER A7

#define SD_ChipSelectPin 53 // example uses hardware SS pin 53
on Mega2560
#define buzzer 6

Adafruit_NeoPixel col1(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel col2(NUMPIXELS, PIN_2, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel col3(NUMPIXELS, PIN_3, NEO_GRB + NEO_KHZ800);
Adafruit_NeoPixel col4(NUMPIXELS, PIN_4, NEO_GRB + NEO_KHZ800);

Adafruit_NeoPixel colL1(NUMPIXELS, PIN_LEFT1, NEO_GRB +
NEO_KHZ800);
Adafruit_NeoPixel colL2(NUMPIXELS, PIN_LEFT2, NEO_GRB +
NEO_KHZ800);
Adafruit_NeoPixel colL3(NUMPIXELS, PIN_LEFT3, NEO_GRB +
NEO_KHZ800);
Adafruit_NeoPixel colR1(NUMPIXELS, PIN_RIGHT1, NEO_GRB +
NEO_KHZ800);
Adafruit_NeoPixel colR2(NUMPIXELS, PIN_RIGHT2, NEO_GRB +
NEO_KHZ800);
Adafruit_NeoPixel colR3(NUMPIXELS, PIN_RIGHT3, NEO_GRB +
NEO_KHZ800);
// new cols for shape prediction
Adafruit_NeoPixel colPL(NUMPIXELS, PINPREL, NEO_GRB +
NEO_KHZ800);
Adafruit_NeoPixel colPR(NUMPIXELS, PINPRER, NEO_GRB +
NEO_KHZ800);
// L rgb(220,0,0) 1
// I rgb(0,0,255) 2
// O rgb(0,128,0) 3
// Z rgb(255,255,0) 4
// T rgb(255,0,255) 5

enum Colors : long
```

```
{
    BLACK = 0x000000,
    WHITE = 0xFFFFFF,
    RED = 0xFF0000,
    BLUE = 0x00FFFF,
    GREEN = 0x00FF00,
    YELLOW = 0xFFFF00,
    PURPLE = 0xFF00FF,
};

Colors colorL = Colors::RED;
Colors colorI = Colors::BLUE;
Colors colorO = Colors::GREEN;
Colors colorZ = Colors::YELLOW;
Colors colorT = Colors::PURPLE;

int centerx = 0;
int centery1 = PIN;
int centery2 = PIN_2;
int centery3 = PIN_3;
int centery4 = PIN_4;
int highestScore = 0;
int DelayVal = 500; // Time (in milliseconds) to pause between
pixels
// centery1b=centery1before
int centery1b = PIN;
int centery2b = PIN_2;
int centery3b = PIN_3;
int centery4b = PIN_4;

int currentStateL;
int currentStateR; // the current reading from the input pin
int currentRotateState;
int SpeedUp;
int currentStateStart;
int currentStateDrop;

int lightArray[38][10]; // record the light pixel
bool currentLanded = true;
long currentShape = -1;
long randNumber = -1;
long randNumberNext = -1;
int score = 0;
int currentL = 0;
int currentT = 0;
```

```
int currentZ = 0;
int currentI = 0;

const int ones_score_pin[] = {22, 23, 24, 25, 26, 27, 28};
const int tens_score_pin[] = {29, 30, 31, 32, 33, 34, 35};

const int ones_highscore_pin[] = {36, 37, 38, 39, 40, 41, 42};
const int tens_highscore_pin[] = {43, 44, 45, 46, 47, 48, 49};

uint8_t current_button_state;
uint8_t previous_button_state;

const int numbers[10][7] =
    {
        {0, 0, 0, 0, 0, 0, 1}, // 0
        {1, 1, 0, 0, 1, 1, 1}, // 1
        {0, 0, 1, 0, 0, 1, 0}, // 2
        {1, 0, 0, 0, 0, 1, 0}, // 3
        {1, 1, 0, 0, 1, 0, 0}, // 4
        {1, 0, 0, 1, 0, 0, 0}, // 5
        {0, 0, 0, 1, 0, 0, 0}, // 6
        {1, 1, 0, 0, 0, 0, 1}, // 7
        {0, 0, 0, 0, 0, 0, 0}, // 8
        {1, 0, 0, 0, 0, 0, 0}, // 9
};
int incomingByte = 0;

bool isGameStarted = false;
bool isGameOver = false;

int count_duplicate = 0;

TMRpcm audio; // create an object for use in this sketch

enum State
{
    game_over,
    add_point,
    background,
    increase,
    decrease,
    mute_bg,
    unmute_bg,
    stop_all,
    default_null
```

```
} audio_state;

static const char wav_gameOver[] PROGMEM = "go3.wav";
static const char wav_backGround[] PROGMEM = "bgm1.wav";

const char *wav_table[] = {
    wav_gameOver,
    wav_backGround,
};

void setup()
{
    Serial.begin(9600);
    pinMode(BUTTON_LEFT, INPUT_PULLUP);
    pinMode(BUTTON_RIGHT, INPUT_PULLUP);
    pinMode(BUTTON_ROTATE, INPUT_PULLUP);
    pinMode(BUTTON_SPEEDUP, INPUT_PULLUP);
    pinMode(BUTTON_START, INPUT_PULLUP);
    pinMode(BUTTON_DROP, INPUT_PULLUP);
    /*
        ----NOTE----
        INPUT_PULLUP inverts the input of the buttons!!!
        LOW -> PRESSED
        HIGH -> RELEASED

    */
    randomSeed(analogRead(0));

    for (int i = 0; i < 38; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            lightArray[i][j] = 0;
        }
    }

    // put your setup code here, to run once:
    for (int x = 0; x < 7; x++)
    {
        pinMode(ones_score_pin[x], OUTPUT);
        pinMode(tens_score_pin[x], OUTPUT);
        pinMode(ones_highscore_pin[x], OUTPUT);
        pinMode(tens_highscore_pin[x], OUTPUT);
    }
```

```
    audio.speakerPin = 5; // 5,6,11 or 46 on Mega, 9 on Uno,
Nano, etc
    pinMode(2, OUTPUT);   // Pin pairs: 9,10 Mega:
5-2,6-7,11-12,46-45

    pinMode(buzzer, OUTPUT);

    audio.setVolume(3);
    audio.quality(1);

    if (!SD.begin(SD_ChipSelectPin))
    {
        return;
    }
    else
    {
        Serial.println("SD OK");
        audio_state = default_null;
    }

    col1.begin();
    col2.begin();
    col3.begin();
    col4.begin();
    colL1.begin();
    colL2.begin();
    colL3.begin();
    colR1.begin();
    colR2.begin();
    colR3.begin();
    colPL.begin();
    colPR.begin();
}

void loop()
{
    col1.setBrightness(10);
    col2.setBrightness(10);
    col3.setBrightness(10);
    col4.setBrightness(10);
    colL1.setBrightness(10);
    colL2.setBrightness(10);
    colL3.setBrightness(10);
    colR1.setBrightness(10);
    colR2.setBrightness(10);
```

```
    colR3.setBrightness(10);
    colPL.setBrightness(10);
    colPR.setBrightness(10);

    if (!isGameStarted)
    {
        initGame();
        currentStateStart = digitalRead(BUTTON_START);
        if (currentStateStart == LOW)
        {
            startGame();
            randomShape(randNumber);
        }
    }
    else
    {
        if (!audio.isPlaying())
        {
            audio_state = background;
        }
        isGameOver = false;

        randomShape(randNumber);
    }
    audioControl();

    displayScore(score);
    displayHighscore(highestScore);
}

void initGame()
{
    for (int i = 0; i < 38; i += 2)
    {
        for (int j = 0; j < 10; j++)
        {
            switch (17 - j)
            {
            case PIN:
                col1.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_2:
                col2.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_3:
```

```
                col3.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_4:
                col4.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_LEFT1:
                colL1.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_LEFT2:
                colL2.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_LEFT3:
                colL3.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_RIGHT1:
                colR1.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_RIGHT2:
                colR2.setPixelColor(i, Colors::WHITE);
                break;
            case PIN_RIGHT3:
                colR3.setPixelColor(i, Colors::WHITE);
                break;
            }
        }
    }

    for (int i = 0; i <= 6; i += 2)
    {
        colPL.setPixelColor(i, Colors::WHITE);
        colPR.setPixelColor(i, Colors::WHITE);
    }
    showCol();
    showNextTileCol();
}

void startGame()
{
    audio_state = stop_all;
    DelayVal = 500;
    count_duplicate = 0;

    for (int i = 0; i < 38; i += 2)
    {
        for (int j = 0; j < 10; j++)
```

```
        {
            switch (17 - j)
            {
            case PIN:
                col1.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_2:
                col2.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_3:
                col3.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_4:
                col4.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_LEFT1:
                colL1.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_LEFT2:
                colL2.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_LEFT3:
                colL3.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_RIGHT1:
                colR1.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_RIGHT2:
                colR2.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_RIGHT3:
                colR3.setPixelColor(i, Colors::BLACK);
                break;
            }
        }
    }

    for (int i = 0; i <= 6; i += 2)
    {
        colPL.setPixelColor(i, Colors::BLACK);
        colPR.setPixelColor(i, Colors::BLACK);
    }
    showCol();
    showNextTileCol();
```

```
    generateNumber();

    for (int i = 0; i < 38; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            lightArray[i][j] = 0;
        }
    }
    score = 0;
    isGameStarted = true;
}

// show all col
void showCol()
{
    col1.show();
    col2.show();
    col3.show();
    col4.show();
    colL1.show();
    colL2.show();
    colL3.show();
    colR1.show();
    colR2.show();
    colR3.show();
}

void showNextTileCol()
{
    colPR.show();
    colPL.show();
}

void generateNumber()
{
    if (randNumber == -1)
    {
        randNumber = random(0, 5);
    }
    else
    {
        randNumber = randNumberNext;
    }
```

```
    do
    {
        randNumberNext = random(0, 5);
        if (randNumber == randNumberNext)
        {
            count_duplicate++;
        }
        if (randNumber != randNumberNext)
        {
            count_duplicate = 0;
        }
    } while (count_duplicate >= 2);

    displayNextTile();
}

void displayNextTile()
{
    for (int i = 0; i <= 6; i += 2)
    {
        colPL.setPixelColor(i, Colors::BLACK);
        colPR.setPixelColor(i, Colors::BLACK);
    }
    showNextTileCol();

    switch (randNumberNext)
    {
    case 0:
        lightOPre();
        break;
    case 1:
        lightZPre();
        break;
    case 2:
        lightLPre();
        break;
    case 3:
        lightTPre();
        break;
    case 4:
        lightIPre();
        break;
    }
    showNextTileCol();
}
```

```
void updateSpeed()
{
    SpeedUp = digitalRead(BUTTON_SPEEDUP);
    // Change Speed
    if (SpeedUp == LOW)
    {
        if (DelayVal - 300 > 0)
        {
            delay(DelayVal - 300);
        }
        else
        {
            delay(DelayVal);
        }
    }
    else
    {
        delay(DelayVal);
    }
}

void updateScore()
{
    score += 1;
    if (score == 4)
    {
        DelayVal -= 50;
    }
    else if (score >= 8)
    {
        if (DelayVal - 40 >= 10)
        {
            DelayVal -= 40;
        }
    }
}

void randomShape(int currentShape)
{
    bool isGameOver = false;
    switch (currentShape)
    {
    case 0:
        if (lightArray[2][4] == 0 && lightArray[2][5] == 0 &&
```

```
lightArray[0][0] == 0 && lightArray[0][1] == 0 &&
lightArray[0][2] == 0 && lightArray[0][3] == 0 &&
lightArray[0][6] == 0 && lightArray[0][7] == 0 &&
lightArray[0][8] == 0 && lightArray[0][9] == 0)
        {
            shapeO();
        }
        else
        {
            isGameOver = true;
        }

        break;
    case 1:
        if ((lightArray[0][5] == 0) && lightArray[2][4] == 0 &&
lightArray[2][5] == 0 && lightArray[4][4] == 0 &&
lightArray[0][0] == 0 && lightArray[0][1] == 0 &&
lightArray[0][2] == 0 && lightArray[0][3] == 0 &&
lightArray[0][6] == 0 && lightArray[0][7] == 0 &&
lightArray[0][8] == 0 && lightArray[0][9] == 0)
        {
            shapeZ();
        }
        else
        {
            isGameOver = true;
        }

        break;
    case 2:
        if (lightArray[0][5] == 0 && lightArray[2][5] == 0 &&
lightArray[4][5] == 0 && lightArray[4][4] == 0 &&
lightArray[0][0] == 0 && lightArray[0][1] == 0 &&
lightArray[0][2] == 0 && lightArray[0][3] == 0 &&
lightArray[0][6] == 0 && lightArray[0][7] == 0 &&
lightArray[0][8] == 0 && lightArray[0][9] == 0)
        {
            shapeL();
        }
        else
        {
            isGameOver = true;
        }

        break;
```

```
    case 3:
        if (lightArray[0][4] == 0 && lightArray[2][4] == 0 &&
lightArray[2][5] == 0 && lightArray[4][4] == 0 &&
lightArray[0][0] == 0 && lightArray[0][1] == 0 &&
lightArray[0][2] == 0 && lightArray[0][3] == 0 &&
lightArray[0][6] == 0 && lightArray[0][7] == 0 &&
lightArray[0][8] == 0 && lightArray[0][9] == 0)
        {
            shapeT();
        }
        else
        {
            isGameOver = true;
        }

        break;
    case 4:
        if (lightArray[0][5] == 0 && lightArray[2][5] == 0 &&
lightArray[4][5] == 0 && lightArray[6][5] == 0 &&
lightArray[0][0] == 0 && lightArray[0][1] == 0 &&
lightArray[0][2] == 0 && lightArray[0][3] == 0 &&
lightArray[0][6] == 0 && lightArray[0][7] == 0 &&
lightArray[0][8] == 0 && lightArray[0][9] == 0)
        {
            shapeI();
        }
        else
        {
            isGameOver = true;
        }

        break;
    }

    if (isGameOver)
    {
        endGame();
    }

    currentLanded = false;
}

// Game Over function
// all pixels set to white
void endGame()
```

```
{
    audio_state = stop_all;
    audioControl();
    audio_state = game_over;
    audioControl();

    initGame();

    for (int i = 0; i < 38; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            lightArray[i][j] = 0;
        }
    }
    // 10.20新加
    centerx = 0;
    if (highestScore <= score)
    {
        highestScore = score;
    }

    isGameStarted = false;
    isGameOver = true;
}

void shapeO()
{
    centery1b = centery1;
    centery2b = centery2;
    centery3b = centery3;
    centery4b = centery4;
    lightO(centerx);
    showCol();

    currentStateL = digitalRead(BUTTON_LEFT);
    currentStateR = digitalRead(BUTTON_RIGHT);

    currentStateDrop = digitalRead(BUTTON_DROP);

    if (currentStateL == LOW && (lightArray[centerx + SPEED][17
- centery3 - 1] == 0) && (lightArray[centerx + 2 + SPEED][17 -
centery3 - 1] == 0))
    {
```

```
        if (centery3 < PIN_LEFT3)
        {
            centery1b = centery1;
            centery2b = centery2;
            centery3b = centery3;
            centery4b = centery4;
            centery1 += 1;
            centery2 += 1;
            centery3 += 1;
            centery4 += 1;
        }
    }
    if ((currentStateR == LOW) && (lightArray[centerx +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx + SPEED
+ 2][17 - centery2 + 1] == 0))
    {

        if (centery2 > PIN_RIGHT3)
        {
            centery1b = centery1;
            centery2b = centery2;
            centery3b = centery3;
            centery4b = centery4;
            centery1 -= 1;
            centery2 -= 1;
            centery3 -= 1;
            centery4 -= 1;
        }
    }
    // Change Speed (Called in the void loop)
    updateSpeed();

    if (currentStateDrop == LOW)
    {
        turnoffO(centerx);
        while ((centerx < NUMPIXELS - 4) && (lightArray[centerx
+ 2 + SPEED][17 - centery2] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3] == 0))
        {
            centerx = centerx + SPEED;
        }

        lightArray[centerx][17 - centery2] = 3;
        lightArray[centerx + 2][17 - centery2] = 3;
        lightArray[centerx][17 - centery3] = 3;
```

```
        lightArray[centerx + 2][17 - centery3] = 3;
        lightO(centerx);
        centerx = 0;
        centery1 = PIN;
        centery2 = PIN_2;
        centery3 = PIN_3;
        centery4 = PIN_4;
        checkLine();

        showCol();
        generateNumber();
    }

    else if ((centerx < NUMPIXELS - 4) && (lightArray[centerx +
2 + SPEED][17 - centery2] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3] == 0))
    {

        turnoffO(centerx);

        centerx = centerx + SPEED;
        showCol();
    }
    else
    {

        lightArray[centerx][17 - centery2] = 3;
        lightArray[centerx + 2][17 - centery2] = 3;
        lightArray[centerx][17 - centery3] = 3;
        lightArray[centerx + 2][17 - centery3] = 3;

        centerx = 0;
        centery1 = PIN;
        centery2 = PIN_2;
        centery3 = PIN_3;
        centery4 = PIN_4;
        checkLine();

        showCol();
        generateNumber();
    }
}

// shapeL
void shapeL()
```

```
{
    centery1b = centery1;
    centery2b = centery2;
    centery3b = centery3;
    centery4b = centery4;
    lightL(centerx);

    showCol();

    currentStateL = digitalRead(BUTTON_LEFT);
    currentStateR = digitalRead(BUTTON_RIGHT);
    currentRotateState = digitalRead(BUTTON_ROTATE); //记录是否按
下rotate button

    currentStateDrop = digitalRead(BUTTON_DROP);
    // need to add more conditions 不同形态的
    if (currentL == 0)
    {
        // move left
        if (currentStateL == LOW && (lightArray[centerx +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery3 - 1] == 0))
        {

            if (centery3 < PIN_LEFT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        // move right
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED][17 - centery3 + 1] == 0) && (lightArray[centerx + SPEED
+ 4][17 - centery2 + 1] == 0))
        {

            if (centery2 > PIN_RIGHT3)
            {
```

```
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (centery2 >
PIN_RIGHT3) && (lightArray[centerx + 2][17 - centery2] == 0) &&
(lightArray[centerx + 2][17 - centery1] == 0))
        {
            // turnoff current state
            turnoffL(centerx);
            // light next state
            currentL = (currentL + 1) % 4;
            lightL(centerx);
        }
    }
    else if (currentL == 1)
    {
        if (currentStateL == LOW && (lightArray[centerx +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0))
        {

            if (centery3 < PIN_LEFT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED][17 - centery1 + 1] == 0) && (lightArray[centerx +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx +
SPEED][17 - centery3 + 1] == 0) && (lightArray[centerx + 2 +
```

```
SPEED][17 - centery3 + 1] == 0))
        {

            if (centery1 > PIN_RIGHT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (lightArray[centerx
+ 2][17 - centery2] == 0) && (lightArray[centerx + 4][17 -
centery2] == 0) && (lightArray[centerx + 6][17 - centery2] ==
0))
        {
            // turnoff current state
            turnoffL(centerx);
            // light next state
            currentL = (currentL + 1) % 4;
            lightL(centerx);
        }
    }
    else if (currentL == 2)
    {
        if (currentStateL == LOW && (lightArray[centerx +
SPEED][17 - centery2 - 1] == 0) && (lightArray[centerx +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery2 - 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery2 - 1] == 0))
        {

            if (centery3 < PIN_LEFT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 += 1;
                centery2 += 1;
```

```
                centery3 += 1;
                centery4 += 1;
            }
        }
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery2 + 1] == 0))
        {

            if (centery2 > PIN_RIGHT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (centery2 >
PIN_RIGHT3) && (lightArray[centerx + 2][17 - centery1] == 0) &&
(lightArray[centerx + 4][17 - centery1] == 0) &&
(lightArray[centerx + 4][17 - centery3] == 0))
        {
            // turnoff current state
            turnoffL(centerx);
            // light next state
            currentL = (currentL + 1) % 4;
            lightL(centerx);
        }
    }
    else if (currentL == 3)
    {
        if (currentStateL == LOW && (lightArray[centerx + 2 +
SPEED][17 - centery1 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery2 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0))
        {

            if (centery3 < PIN_LEFT3)
            {
```

```
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED][17 - centery1 + 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery1 + 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 + 1] == 0))
        {

            if (centery1 > PIN_RIGHT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (lightArray[centerx
+ 4][17 - centery3] == 0) && (lightArray[centerx + 6][17 -
centery3] == 0) && (lightArray[centerx + 6][17 - centery2] ==
0))
        {
            // turnoff current state
            turnoffL(centerx);
            // light next state
            currentL = (currentL + 1) % 4;
            lightL(centerx);
        }
    }
    // Change Speed (called in void loop)
    updateSpeed();
```

```
    // Drop
    // need to add more conditions 不同形态
    // 0
    if (currentL == 0)
    {
        if (currentStateDrop == LOW)
        {
            turnoffL(centerx);
            while ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + 4 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 4 + SPEED][17 - centery3] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery3] = 1;
            lightArray[centerx + 2][17 - centery3] = 1;
            lightArray[centerx + 4][17 - centery3] = 1;
            lightArray[centerx + 4][17 - centery2] = 1;
            lightL(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();

            currentL = 0;
        }
        else if ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + 4 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 4 + SPEED][17 - centery3] == 0))
        {
            turnoffL(centerx);
            centerx = centerx + SPEED;
            showCol();
        }
        else
        {
            lightArray[centerx][17 - centery3] = 1;
            lightArray[centerx + 2][17 - centery3] = 1;
            lightArray[centerx + 4][17 - centery3] = 1;
            lightArray[centerx + 4][17 - centery2] = 1;
```

```
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentL = 0;
        }
    }
    // 1
    if (currentL == 1)
    {
        if (currentStateDrop == LOW)
        {
            turnoffL(centerx);
            while ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + SPEED][17 - centery1] == 0) &&
(lightArray[centerx + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery1] = 1;
            lightArray[centerx][17 - centery2] = 1;
            lightArray[centerx][17 - centery3] = 1;
            lightArray[centerx + 2][17 - centery3] = 1;
            lightL(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentL = 0;
        }
        else if ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + SPEED][17 - centery1] == 0) &&
(lightArray[centerx + SPEED][17 - centery2] == 0) &&
```

```
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
        {
            turnoffL(centerx);
            centerx = centerx + SPEED;
            showCol();
        }
        else
        {
            lightArray[centerx][17 - centery1] = 1;
            lightArray[centerx][17 - centery2] = 1;
            lightArray[centerx][17 - centery3] = 1;
            lightArray[centerx + 2][17 - centery3] = 1;

            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentL = 0;
        }
    }
    if (currentL == 2)
    {
        if (currentStateDrop == LOW)
        {
            turnoffL(centerx);
            while ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + SPEED][17 - centery3] == 0) &&
(lightArray[centerx + 4 + SPEED][17 - centery2] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery3] = 1;
            lightArray[centerx][17 - centery2] = 1;
            lightArray[centerx + 2][17 - centery2] = 1;
            lightArray[centerx + 4][17 - centery2] = 1;
            lightL(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
```

```
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentL = 0;
        }
        else if ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + SPEED][17 - centery3] == 0) &&
(lightArray[centerx + 4 + SPEED][17 - centery2] == 0))
        {
            turnoffL(centerx);
            centerx = centerx + SPEED;
            showCol();
        }
        else
        {
            lightArray[centerx][17 - centery3] = 1;
            lightArray[centerx][17 - centery2] = 1;
            lightArray[centerx + 2][17 - centery2] = 1;
            lightArray[centerx + 4][17 - centery2] = 1;

            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentL = 0;
        }
    }
    if (currentL == 3)
    {
        if (currentStateDrop == LOW)
        {
            turnoffL(centerx);
            while ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + 2 + SPEED][17 - centery1] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
            {
                centerx = centerx + SPEED;
```

```
            }
            lightArray[centerx][17 - centery1] = 1;
            lightArray[centerx + 2][17 - centery1] = 1;
            lightArray[centerx + 2][17 - centery2] = 1;
            lightArray[centerx + 2][17 - centery3] = 1;
            lightL(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentL = 0;
        }
        else if ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + 2 + SPEED][17 - centery1] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
        {
            turnoffL(centerx);
            centerx = centerx + SPEED;
            showCol();
        }
        else
        {
            lightArray[centerx][17 - centery1] = 1;
            lightArray[centerx + 2][17 - centery1] = 1;
            lightArray[centerx + 2][17 - centery2] = 1;
            lightArray[centerx + 2][17 - centery3] = 1;

            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentL = 0;
        }
    }
```

```
}

// shape Z
void shapeZ()
{
    centery1b = centery1;
    centery2b = centery2;
    centery3b = centery3;
    centery4b = centery4;
    lightZ(centerx);

    showCol();

    currentStateL = digitalRead(BUTTON_LEFT);
    currentStateR = digitalRead(BUTTON_RIGHT);
    currentRotateState = digitalRead(BUTTON_ROTATE);
    currentStateDrop = digitalRead(BUTTON_DROP);

    if (currentZ == 0)
    {
        if (currentStateL == LOW && (lightArray[centerx +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery2 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery2 - 1] == 0))
        {

            if (centery3 < PIN_LEFT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED + 2][17 - centery2 + 1] == 0) && (lightArray[centerx +
SPEED + 4][17 - centery2 + 1] == 0))
        {
            // clear(centerx-SPEED);
            if (centery2 > PIN_RIGHT3)
```

```
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }

        if ((currentRotateState == LOW) && (centery2 >
PIN_RIGHT3) && (lightArray[centerx + 4][17 - centery3] == 0) &&
(lightArray[centerx + 2][17 - centery1] == 0))
        {
            // turnoff current state
            turnoffZ(centerx);
            // light next state
            currentZ = (currentZ + 1) % 2;
            lightZ(centerx);
        }
    }
    else if (currentZ == 1)
    {
        if (currentStateL == LOW && (lightArray[centerx + 2 +
SPEED][17 - centery2 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0))
        {

            if (centery3 < PIN_LEFT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED + 2][17 - centery3 + 1] == 0) && (lightArray[centerx +
SPEED + 2][17 - centery2 + 1] == 0) && (lightArray[centerx +
```

```
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx +
SPEED][17 - centery1 + 1] == 0))
        {
            // clear(centerx-SPEED);
            if (centery1 > PIN_RIGHT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }

        if ((currentRotateState == LOW) && (lightArray[centerx
+ 4][17 - centery3] == 0) && (lightArray[centerx + 4][17 -
centery2] == 0) && (lightArray[centerx + 6][17 - centery2] ==
0))
        {
            // turnoff current state
            turnoffZ(centerx);
            // light next state
            currentZ = (currentZ + 1) % 2;
            lightZ(centerx);
        }
    }

    updateSpeed();

    if (currentZ == 0)
    {
        if (currentStateDrop == LOW)
        {
            turnoffZ(centerx);
            while ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + 4 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx + 2][17 - centery2] = 4;
            lightArray[centerx + 4][17 - centery2] = 4;
```

```
            lightArray[centerx][17 - centery3] = 4;
            lightArray[centerx + 2][17 - centery3] = 4;
            lightZ(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentZ = 0;
        }
        else if ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + 4 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
        {

            turnoffZ(centerx);

            centerx = centerx + SPEED;
            showCol();
        }
        else
        {

            lightArray[centerx + 2][17 - centery2] = 4;
            lightArray[centerx + 4][17 - centery2] = 4;
            lightArray[centerx][17 - centery3] = 4;
            lightArray[centerx + 2][17 - centery3] = 4;

            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentZ = 0;
        }
    }
    if (currentZ == 1)
```

```
    {
        if (currentStateDrop == LOW)
        {
            turnoffZ(centerx);
            while ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + SPEED][17 - centery1] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx + 2][17 - centery3] = 4;
            lightArray[centerx][17 - centery2] = 4;
            lightArray[centerx + 2][17 - centery2] = 4;
            lightArray[centerx][17 - centery1] = 4;
            lightZ(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();
            showCol();
            generateNumber();
            currentZ = 0;
        }
        else if ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + SPEED][17 - centery1] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
        {

            turnoffZ(centerx);

            centerx = centerx + SPEED;
            showCol();
        }
        else
        {

            lightArray[centerx + 2][17 - centery3] = 4;
            lightArray[centerx][17 - centery2] = 4;
            lightArray[centerx + 2][17 - centery2] = 4;
            lightArray[centerx][17 - centery1] = 4;
```

```
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentZ = 0;
        }
    }
}
// shapeT
void shapeT()
{
    centery1b = centery1;
    centery2b = centery2;
    centery3b = centery3;
    centery4b = centery4;
    lightT(centerx);

    showCol();

    currentStateL = digitalRead(BUTTON_LEFT);
    currentStateR = digitalRead(BUTTON_RIGHT);
    currentRotateState = digitalRead(BUTTON_ROTATE);
    currentStateDrop = digitalRead(BUTTON_DROP);

    if (currentT == 0)
    {
        // move left
        if ((currentStateL == LOW) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery2 - 1] == 0))
        {
            if (centery3 < PIN_LEFT3)
            {
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        // move right
```

```
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery2 + 1] == 0))
        {
            if (centery2 > PIN_RIGHT3)
            {
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (centery3 <
PIN_LEFT3) && (lightArray[centerx + 4][17 - centery3] == 0) &&
(lightArray[centerx + 4][17 - centery4] == 0))
        {
            // turnoff current state
            turnoffT(centerx);
            // light next state
            currentT = (currentT + 1) % 4;
            lightT(centerx);
        }
    }
    else if (currentT == 1)
    {
        // move left
        if ((currentStateL == LOW) && (lightArray[centerx + 2 +
SPEED][17 - centery2 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery4 - 1] == 0))
        {
            if (centery4 < PIN_LEFT3)
            {
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        // move right
        if ((currentStateR == LOW) && (lightArray[centerx + 2 +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 + 1] == 0) && (lightArray[centerx + 2 +
```

```
SPEED][17 - centery4 + 1] == 0))
        {
            if (centery2 > PIN_RIGHT3)
            {
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (lightArray[centerx
+ 4][17 - centery3] == 0) && (lightArray[centerx + 4][17 -
centery2] == 0) && (lightArray[centerx + 6][17 - centery3] ==
0))
        {
            // turnoff current state
            turnoffT(centerx);
            // light next state
            currentT = (currentT + 1) % 4;
            lightT(centerx);
        }
    }
    else if (currentT == 2)
    {
        // move left
        if ((currentStateL == LOW) && (lightArray[centerx +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery3 - 1] == 0))
        {
            if (centery3 < PIN_LEFT3)
            {
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        // move right
        if ((currentStateR == LOW) && (lightArray[centerx + 2 +
SPEED][17 - centery3 + 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery3 + 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery2 + 1] == 0))
        {
```

```
            if (centery2 > PIN_RIGHT3)
            {
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (centery3 <
PIN_LEFT3) && (lightArray[centerx + 2][17 - centery4] == 0))
        {
            // turnoff current state
            turnoffT(centerx);
            // light next state
            currentT = (currentT + 1) % 4;
            lightT(centerx);
        }
    }
    else if (currentT == 3)
    {
        // move left
        if ((currentStateL == LOW) && (lightArray[centerx +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx +
SPEED][17 - centery4 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0))
        {
            if (centery4 < PIN_LEFT3)
            {
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        // move right
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED][17 - centery3 + 1] == 0) && (lightArray[centerx +
SPEED][17 - centery2 + 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 + 1] == 0))
        {
            if (centery2 > PIN_RIGHT3)
            {
                centery1 -= 1;
                centery2 -= 1;
```

```
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (lightArray[centerx
+ 2][17 - centery2] == 0) && (lightArray[centerx + 4][17 -
centery2] == 0) && (lightArray[centerx + 6][17 - centery2] ==
0) && (lightArray[centerx + 4][17 - centery3] == 0))
        {
            // turnoff current state
            turnoffT(centerx);
            // light next state
            currentT = (currentT + 1) % 4;
            lightT(centerx);
        }
    }

    // Change Speed
    updateSpeed();

    if (currentT == 0)
    {
        if (currentStateDrop == LOW)
        {
            turnoffT(centerx);
            while ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + 4 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery2] = 5;
            lightArray[centerx + 2][17 - centery2] = 5;
            lightArray[centerx + 4][17 - centery2] = 5;
            lightArray[centerx + 2][17 - centery3] = 5;
            lightT(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
```

```
            generateNumber();
            currentT = 0;
        }
        else if ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + 4 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0))
        {
            turnoffT(centerx);
            centerx = centerx + SPEED;
            showCol();
        }
        else
        {
            lightArray[centerx][17 - centery2] = 5;
            lightArray[centerx + 2][17 - centery2] = 5;
            lightArray[centerx + 4][17 - centery2] = 5;
            lightArray[centerx + 2][17 - centery3] = 5;
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentT = 0;
        }
    }
    if (currentT == 1)
    {
        if (currentStateDrop == LOW)
        {
            turnoffT(centerx);
            while ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + 2 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery4] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery3] = 5;
            lightArray[centerx + 2][17 - centery2] = 5;
            lightArray[centerx + 2][17 - centery3] = 5;
            lightArray[centerx + 2][17 - centery4] = 5;
```

```
            lightT(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentT = 0;
        }
        else if ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + 2 + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery4] == 0))
        {
            turnoffT(centerx);
            centerx = centerx + SPEED;
            showCol();
        }
        else
        {
            lightArray[centerx][17 - centery3] = 5;
            lightArray[centerx + 2][17 - centery2] = 5;
            lightArray[centerx + 2][17 - centery3] = 5;
            lightArray[centerx + 2][17 - centery4] = 5;
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentT = 0;
        }
    }
    if (currentT == 2)
    {
        if (currentStateDrop == LOW)
        {
            turnoffT(centerx);
            while ((centerx < NUMPIXELS - 6) &&
```

```
(lightArray[centerx + 4 + SPEED][17 - centery3] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery2] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery3] = 5;
            lightArray[centerx + 2][17 - centery3] = 5;
            lightArray[centerx + 4][17 - centery3] = 5;
            lightArray[centerx + 2][17 - centery2] = 5;
            lightT(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentT = 0;
        }
        else if ((centerx < NUMPIXELS - 6) &&
(lightArray[centerx + 4 + SPEED][17 - centery3] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery2] == 0))
        {
            turnoffT(centerx);
            centerx = centerx + SPEED;
            showCol();
        }
        else
        {
            lightArray[centerx][17 - centery3] = 5;
            lightArray[centerx + 2][17 - centery3] = 5;
            lightArray[centerx + 4][17 - centery3] = 5;
            lightArray[centerx + 2][17 - centery2] = 5;
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentT = 0;
```

```
        }
    }
    if (currentT == 3)
    {
        if (currentStateDrop == LOW)
        {
            turnoffT(centerx);
            while ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0) &&
(lightArray[centerx + SPEED][17 - centery4] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery2] = 5;
            lightArray[centerx][17 - centery3] = 5;
            lightArray[centerx][17 - centery4] = 5;
            lightArray[centerx + 2][17 - centery3] = 5;
            lightT(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentT = 0;
        }
        else if ((centerx < NUMPIXELS - 4) &&
(lightArray[centerx + SPEED][17 - centery2] == 0) &&
(lightArray[centerx + 2 + SPEED][17 - centery3] == 0) &&
(lightArray[centerx + SPEED][17 - centery4] == 0))
        {
            turnoffT(centerx);
            centerx = centerx + SPEED;
            showCol();
        }
        else
        {
            lightArray[centerx][17 - centery2] = 5;
            lightArray[centerx][17 - centery3] = 5;
            lightArray[centerx][17 - centery4] = 5;
            lightArray[centerx + 2][17 - centery3] = 5;
```

```
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentT = 0;
        }
    }
}

// shape I
void shapeI()
{
    centery1b = centery1;
    centery2b = centery2;
    centery3b = centery3;
    centery4b = centery4;
    lightI(centerx);

    showCol();

    currentStateL = digitalRead(BUTTON_LEFT);
    currentStateR = digitalRead(BUTTON_RIGHT);
    currentRotateState = digitalRead(BUTTON_ROTATE);
    currentStateDrop = digitalRead(BUTTON_DROP);

    if (currentI == 0)
    {
        if (currentStateL == LOW && (lightArray[centerx +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 2 +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 4 +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx + 6 +
SPEED][17 - centery3 - 1] == 0))
        {
            if (centery3 < PIN_LEFT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 += 1;
```

```
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED + 2][17 - centery3 + 1] == 0) && (lightArray[centerx +
SPEED + 4][17 - centery3 + 1] == 0) && (lightArray[centerx +
SPEED + 6][17 - centery3 + 1] == 0) && (lightArray[centerx +
SPEED + 6][17 - centery3 + 1] == 0))
        {
            if (centery3 > PIN_RIGHT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (centery3 >
PIN_RIGHT2) && (centery3 < PIN_LEFT3) && (lightArray[centerx +
2][17 - centery4] == 0) && (lightArray[centerx + 2][17 -
centery2] == 0) && (lightArray[centerx + 2][17 - centery1] ==
0))
        {
            // turnoff current state
            turnoffI(centerx);
            // light next state
            currentI = (currentI + 1) % 2;
            lightI(centerx);
        }
    }
    else if (currentI == 1)
    {
        if (currentStateL == LOW && (lightArray[centerx +
SPEED][17 - centery1 - 1] == 0) && (lightArray[centerx +
SPEED][17 - centery2 - 1] == 0) && (lightArray[centerx +
SPEED][17 - centery3 - 1] == 0) && (lightArray[centerx +
SPEED][17 - centery4 - 1] == 0))
        {
```

```
            if (centery4 < PIN_LEFT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 += 1;
                centery2 += 1;
                centery3 += 1;
                centery4 += 1;
            }
        }
        if ((currentStateR == LOW) && (lightArray[centerx +
SPEED + 2][17 - centery3 + 1] == 0) && (lightArray[centerx +
SPEED + 4][17 - centery3 + 1] == 0) && (lightArray[centerx +
SPEED + 6][17 - centery3 + 1] == 0) && (lightArray[centerx +
SPEED + 6][17 - centery3 + 1] == 0))
        {
            if (centery1 > PIN_RIGHT3)
            {
                centery1b = centery1;
                centery2b = centery2;
                centery3b = centery3;
                centery4b = centery4;
                centery1 -= 1;
                centery2 -= 1;
                centery3 -= 1;
                centery4 -= 1;
            }
        }
        // rotate
        if ((currentRotateState == LOW) && (lightArray[centerx
+ 2][17 - centery3] == 0) && (lightArray[centerx + 4][17 -
centery3] == 0) && (lightArray[centerx + 6][17 - centery3] ==
0))
        {
            // turnoff current state
            turnoffI(centerx);
            // light next state
            currentI = (currentI + 1) % 2;
            lightI(centerx);
        }
    }

    // Change Speed
```

```
    updateSpeed();

    if (currentI == 0)
    {
        if (currentStateDrop == LOW)
        {
            turnoffI(centerx);
            while ((centerx < NUMPIXELS - 8) &&
(lightArray[centerx + 6 + SPEED][17 - centery3] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery3] = 2;
            lightArray[centerx + 2][17 - centery3] = 2;
            lightArray[centerx + 4][17 - centery3] = 2;
            lightArray[centerx + 6][17 - centery3] = 2;
            lightI(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentI = 0;
        }
        else if ((centerx < NUMPIXELS - 8) &&
(lightArray[centerx + 6 + SPEED][17 - centery3] == 0))
        {

            turnoffI(centerx);

            centerx = centerx + SPEED;
            showCol();
        }
        else
        {

            lightArray[centerx][17 - centery3] = 2;
            lightArray[centerx + 2][17 - centery3] = 2;
            lightArray[centerx + 4][17 - centery3] = 2;
            lightArray[centerx + 6][17 - centery3] = 2;
```

```
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();
            currentI = 0;
        }
    }
    if (currentI == 1)
    {
        if (currentStateDrop == LOW)
        {
            turnoffI(centerx);
            while ((centerx < NUMPIXELS) && (lightArray[centerx
+ SPEED][17 - centery1] == 0) && (lightArray[centerx +
SPEED][17 - centery2] == 0) && (lightArray[centerx + SPEED][17
- centery3] == 0) && (lightArray[centerx + SPEED][17 -
centery4] == 0))
            {
                centerx = centerx + SPEED;
            }
            lightArray[centerx][17 - centery1] = 2;
            lightArray[centerx][17 - centery2] = 2;
            lightArray[centerx][17 - centery3] = 2;
            lightArray[centerx][17 - centery4] = 2;
            lightI(centerx);
            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();

            currentI = 0;
        }
        else if ((centerx < NUMPIXELS) && (lightArray[centerx +
SPEED][17 - centery1] == 0) && (lightArray[centerx + SPEED][17
- centery2] == 0) && (lightArray[centerx + SPEED][17 -
```

```
centery3] == 0) && (lightArray[centerx + SPEED][17 - centery4]
== 0))
        {

            turnoffI(centerx);

            centerx = centerx + SPEED;
            showCol();
        }
        else
        {

            lightArray[centerx][17 - centery1] = 2;
            lightArray[centerx][17 - centery2] = 2;
            lightArray[centerx][17 - centery3] = 2;
            lightArray[centerx][17 - centery4] = 2;

            centerx = 0;
            centery1 = PIN;
            centery2 = PIN_2;
            centery3 = PIN_3;
            centery4 = PIN_4;
            checkLine();

            showCol();
            generateNumber();

            currentI = 0;
        }
    }
}

// check whether to clear the line
void checkLine()
{
    audio_state = mute_bg;
    audioControl();
    bool playAudioOnce = false;
    int product = 1;

    int productArray[38];
    for (int i = 0; i < 38; i = i + 1)
    {
        productArray[i] = 0;
    }
```

```
    for (int i = 36; i > 0; i = i - 2)
    {
        product = 1;
        for (int j = 0; j < 10; j = j + 1)
        {
            product = product * lightArray[i][j];
        }
        if (product != 0)
        {
            productArray[i] = product;
        }
    }
    blink(productArray);

    for (int i = 36; i > 0; i = i - 2)
    {
        product = 1;
        for (int j = 0; j < 10; j = j + 1)
        {
            product = product * lightArray[i][j];
        }

        if (product != 0)
        {
            updateScore();
            if (!playAudioOnce)
            {
                audio_state = add_point;
                audioControl();
                playAudioOnce = true;
            }

            arrayUpdate(i);

            i = i + 2;
        }
    }
    audio_state = unmute_bg;
    audioControl();
    arrayLight();
}
void blink(int productArray[])
{
    for (int blinktime = 0; blinktime < 4; blinktime++)
```

```
    {
        for (int i = 0; i < 38; i = i + 2)
        {
            if (productArray[i] != 0)
            {
                if (blinktime % 2 == 0)
                {
                    col1.setPixelColor(i, Colors::BLACK);
                    col2.setPixelColor(i, Colors::BLACK);
                    col3.setPixelColor(i, Colors::BLACK);
                    col4.setPixelColor(i, Colors::BLACK);
                    colL1.setPixelColor(i, Colors::BLACK);
                    colL2.setPixelColor(i, Colors::BLACK);
                    colL3.setPixelColor(i, Colors::BLACK);
                    colR1.setPixelColor(i, Colors::BLACK);
                    colR2.setPixelColor(i, Colors::BLACK);
                    colR3.setPixelColor(i, Colors::BLACK);
                }
                else
                {
                    col1.setPixelColor(i, Colors::WHITE);
                    col2.setPixelColor(i, Colors::WHITE);
                    col3.setPixelColor(i, Colors::WHITE);
                    col4.setPixelColor(i, Colors::WHITE);
                    colL1.setPixelColor(i, Colors::WHITE);
                    colL2.setPixelColor(i, Colors::WHITE);
                    colL3.setPixelColor(i, Colors::WHITE);
                    colR1.setPixelColor(i, Colors::WHITE);
                    colR2.setPixelColor(i, Colors::WHITE);
                    colR3.setPixelColor(i, Colors::WHITE);
                }
            }
        }
        showCol();
    }
}

void arrayUpdate(int line)
{

    for (int i = line; i > 0; i = i - 2)
    {
        for (int j = 0; j < 10; j = j + 1)
        {
            lightArray[i][j] = lightArray[i - 2][j];
```

```
        }
    }
    for (int j = 0; j < 10; j = j + 1)
    {
        lightArray[0][j] = 0;
    }
}
// according to the lightArray to set pixel color
void arrayLight()
{
    for (int i = 0; i < 38; i++)
    {
        for (int j = 0; j < 10; j++)
        {
            switch (17 - j)
            {
            case PIN:
                col1.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_2:
                col2.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_3:
                col3.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_4:
                col4.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_LEFT1:
                colL1.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_LEFT2:
                colL2.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_LEFT3:
                colL3.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_RIGHT1:
                colR1.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_RIGHT2:
                colR2.setPixelColor(i, Colors::BLACK);
                break;
            case PIN_RIGHT3:
                colR3.setPixelColor(i, Colors::BLACK);
```

```
                    break;
                }
                if (lightArray[i][j] == 1)
                {
                    switch (17 - j)
                    {
                    case PIN:
                        col1.setPixelColor(i, colorL);
                        break;
                    case PIN_2:
                        col2.setPixelColor(i, colorL);
                        break;
                    case PIN_3:
                        col3.setPixelColor(i, colorL);
                        break;
                    case PIN_4:
                        col4.setPixelColor(i, colorL);
                        break;
                    case PIN_LEFT1:
                        colL1.setPixelColor(i, colorL);
                        break;
                    case PIN_LEFT2:
                        colL2.setPixelColor(i, colorL);
                        break;
                    case PIN_LEFT3:
                        colL3.setPixelColor(i, colorL);
                        break;
                    case PIN_RIGHT1:
                        colR1.setPixelColor(i, colorL);
                        break;
                    case PIN_RIGHT2:
                        colR2.setPixelColor(i, colorL);
                        break;
                    case PIN_RIGHT3:
                        colR3.setPixelColor(i, colorL);
                        break;
                    }
                }
                if (lightArray[i][j] == 2)
                {
                    switch (17 - j)
                    {
                    case PIN:
                        col1.setPixelColor(i, colorI);
                        break;
```

```
                case PIN_2:
                    col2.setPixelColor(i, colorI);
                    break;
                case PIN_3:
                    col3.setPixelColor(i, colorI);
                    break;
                case PIN_4:
                    col4.setPixelColor(i, colorI);
                    break;
                case PIN_LEFT1:
                    colL1.setPixelColor(i, colorI);
                    break;
                case PIN_LEFT2:
                    colL2.setPixelColor(i, colorI);
                    break;
                case PIN_LEFT3:
                    colL3.setPixelColor(i, colorI);
                    break;
                case PIN_RIGHT1:
                    colR1.setPixelColor(i, colorI);
                    break;
                case PIN_RIGHT2:
                    colR2.setPixelColor(i, colorI);
                    break;
                case PIN_RIGHT3:
                    colR3.setPixelColor(i, colorI);
                    break;
                }
            }
            if (lightArray[i][j] == 3)
            {
                switch (17 - j)
                {
                case PIN:
                    col1.setPixelColor(i, colorO);
                    break;
                case PIN_2:
                    col2.setPixelColor(i, colorO);
                    break;
                case PIN_3:
                    col3.setPixelColor(i, colorO);
                    break;
                case PIN_4:
                    col4.setPixelColor(i, colorO);
                    break;
```

```
                case PIN_LEFT1:
                    colL1.setPixelColor(i, color0);
                    break;
                case PIN_LEFT2:
                    colL2.setPixelColor(i, color0);
                    break;
                case PIN_LEFT3:
                    colL3.setPixelColor(i, color0);
                    break;
                case PIN_RIGHT1:
                    colR1.setPixelColor(i, color0);
                    break;
                case PIN_RIGHT2:
                    colR2.setPixelColor(i, color0);
                    break;
                case PIN_RIGHT3:
                    colR3.setPixelColor(i, color0);
                    break;
                }
            }
            if (lightArray[i][j] == 4)
            {
                switch (17 - j)
                {
                case PIN:
                    col1.setPixelColor(i, colorZ);
                    break;
                case PIN_2:
                    col2.setPixelColor(i, colorZ);
                    break;
                case PIN_3:
                    col3.setPixelColor(i, colorZ);
                    break;
                case PIN_4:
                    col4.setPixelColor(i, colorZ);
                    break;
                case PIN_LEFT1:
                    colL1.setPixelColor(i, colorZ);
                    break;
                case PIN_LEFT2:
                    colL2.setPixelColor(i, colorZ);
                    break;
                case PIN_LEFT3:
                    colL3.setPixelColor(i, colorZ);
                    break;
```

```
                case PIN_RIGHT1:
                    colR1.setPixelColor(i, colorZ);
                    break;
                case PIN_RIGHT2:
                    colR2.setPixelColor(i, colorZ);
                    break;
                case PIN_RIGHT3:
                    colR3.setPixelColor(i, colorZ);
                    break;
                }
            }
            if (lightArray[i][j] == 5)
            {
                switch (17 - j)
                {
                case PIN:
                    col1.setPixelColor(i, colorT);
                    break;
                case PIN_2:
                    col2.setPixelColor(i, colorT);
                    break;
                case PIN_3:
                    col3.setPixelColor(i, colorT);
                    break;
                case PIN_4:
                    col4.setPixelColor(i, colorT);
                    break;
                case PIN_LEFT1:
                    colL1.setPixelColor(i, colorT);
                    break;
                case PIN_LEFT2:
                    colL2.setPixelColor(i, colorT);
                    break;
                case PIN_LEFT3:
                    colL3.setPixelColor(i, colorT);
                    break;
                case PIN_RIGHT1:
                    colR1.setPixelColor(i, colorT);
                    break;
                case PIN_RIGHT2:
                    colR2.setPixelColor(i, colorT);
                    break;
                case PIN_RIGHT3:
                    colR3.setPixelColor(i, colorT);
                    break;
```

```
                }
            }
        }
    }
}

// turnoff
void turnoff0(int centerx)
{
    switch (centery1b)
    {
    case PIN:

        col2.setPixelColor(centerx, Colors::BLACK);
        col2.setPixelColor(centerx + 2, Colors::BLACK);
        col3.setPixelColor(centerx, Colors::BLACK);
        col3.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_2:

        col3.setPixelColor(centerx, Colors::BLACK);
        col3.setPixelColor(centerx + 2, Colors::BLACK);
        col4.setPixelColor(centerx, Colors::BLACK);
        col4.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_3:

        col4.setPixelColor(centerx, Colors::BLACK);
        col4.setPixelColor(centerx + 2, Colors::BLACK);
        colL1.setPixelColor(centerx, Colors::BLACK);
        colL1.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_4:

        colL1.setPixelColor(centerx, Colors::BLACK);
        colL1.setPixelColor(centerx + 2, Colors::BLACK);
        colL2.setPixelColor(centerx, Colors::BLACK);
        colL2.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_LEFT1:
```

```
        colL2.setPixelColor(centerx, Colors::BLACK);
        colL2.setPixelColor(centerx + 2, Colors::BLACK);
        colL3.setPixelColor(centerx, Colors::BLACK);
        colL3.setPixelColor(centerx + 2, Colors::BLACK);
        break;
    case PIN_RIGHT1:

        col1.setPixelColor(centerx, Colors::BLACK);
        col1.setPixelColor(centerx + 2, Colors::BLACK);
        col2.setPixelColor(centerx, Colors::BLACK);
        col2.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_RIGHT2:

        colR1.setPixelColor(centerx, Colors::BLACK);
        colR1.setPixelColor(centerx + 2, Colors::BLACK);
        col1.setPixelColor(centerx, Colors::BLACK);
        col1.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_RIGHT3:

        colR2.setPixelColor(centerx, Colors::BLACK);
        colR2.setPixelColor(centerx + 2, Colors::BLACK);
        colR1.setPixelColor(centerx, Colors::BLACK);
        colR1.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_RIGHT3 - 1:
        colR3.setPixelColor(centerx, Colors::BLACK);
        colR3.setPixelColor(centerx + 2, Colors::BLACK);
        colR2.setPixelColor(centerx, Colors::BLACK);
        colR2.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    }
}

void turnoffL(int centerx)
{
    if (currentL == 0)
    {
        switch (centery1b)
        {
```

```
        case PIN:

            col2.setPixelColor(centerx + 4, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_2:

            col3.setPixelColor(centerx + 4, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_3:

            col4.setPixelColor(centerx + 4, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_4:

            colL1.setPixelColor(centerx + 4, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_LEFT1:

            colL2.setPixelColor(centerx + 4, Colors::BLACK);
            colL3.setPixelColor(centerx, Colors::BLACK);
            colL3.setPixelColor(centerx + 2, Colors::BLACK);
            colL3.setPixelColor(centerx + 4, Colors::BLACK);
            break;
        case PIN_RIGHT1:

            col1.setPixelColor(centerx + 4, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 4, Colors::BLACK);
```

```
            break;
        case PIN_RIGHT2:

            colR1.setPixelColor(centerx + 4, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_RIGHT3:

            colR2.setPixelColor(centerx + 4, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx + 4, Colors::BLACK);
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        }
    }
    if (currentL == 1)
    {
        switch (centery1b)
        {
        case PIN:

            col1.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_2:

            col2.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
```

```
        break;
    case PIN_3:

        col3.setPixelColor(centerx, Colors::BLACK);
        col4.setPixelColor(centerx, Colors::BLACK);
        colL1.setPixelColor(centerx, Colors::BLACK);
        colL1.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_4:

        col4.setPixelColor(centerx, Colors::BLACK);
        colL1.setPixelColor(centerx, Colors::BLACK);
        colL2.setPixelColor(centerx, Colors::BLACK);
        colL2.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_LEFT1:

        colL1.setPixelColor(centerx, Colors::BLACK);
        colL2.setPixelColor(centerx, Colors::BLACK);
        colL3.setPixelColor(centerx, Colors::BLACK);
        colL3.setPixelColor(centerx + 2, Colors::BLACK);
        break;
    case PIN_RIGHT1:

        colR1.setPixelColor(centerx, Colors::BLACK);
        col1.setPixelColor(centerx, Colors::BLACK);
        col2.setPixelColor(centerx, Colors::BLACK);
        col2.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_RIGHT2:

        colR2.setPixelColor(centerx, Colors::BLACK);
        colR1.setPixelColor(centerx, Colors::BLACK);
        col1.setPixelColor(centerx, Colors::BLACK);
        col1.setPixelColor(centerx + 2, Colors::BLACK);

        break;
    case PIN_RIGHT3:

        colR3.setPixelColor(centerx, Colors::BLACK);
        colR2.setPixelColor(centerx, Colors::BLACK);
```

```
            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        }
    }
    if (currentL == 2)
    {
        switch (centery1b)
        {
        case PIN:

            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 4, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_2:

            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 4, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_3:

            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 4, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_4:

            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 4, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_LEFT1:

            colL2.setPixelColor(centerx, Colors::BLACK);
```

```
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 4, Colors::BLACK);
            colL3.setPixelColor(centerx, Colors::BLACK);
            break;
        case PIN_RIGHT1:

            col1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 4, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_RIGHT2:

            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 4, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_RIGHT3:

            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx + 4, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_RIGHT3 - 1:

            colR3.setPixelColor(centerx, Colors::BLACK);
            colR3.setPixelColor(centerx + 2, Colors::BLACK);
            colR3.setPixelColor(centerx + 4, Colors::BLACK);
            colR2.setPixelColor(centerx, Colors::BLACK);

            break;
        }
    }
    if (currentL == 3)
    {
        switch (centery1b)
        {
        case PIN:

            col1.setPixelColor(centerx, Colors::BLACK);
```

```
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_2:

            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_3:

            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_4:

            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_LEFT1:

            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL3.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_RIGHT1:

            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_RIGHT2:
```

```
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_RIGHT3:

            colR3.setPixelColor(centerx, Colors::BLACK);
            colR3.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        }
    }
}

void turnoffZ(int centerx)
{
    if (currentZ == 0)
    {
        switch (centery1b)
        {
        case PIN:

            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 4, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_2:

            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 4, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_3:

            col4.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 4, Colors::BLACK);
```

```
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_4:

            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 4, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_LEFT1:

            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 4, Colors::BLACK);
            colL3.setPixelColor(centerx, Colors::BLACK);
            colL3.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_RIGHT1:

            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 4, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_RIGHT2:

            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 4, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_RIGHT3:

            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx + 4, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx + 2, Colors::BLACK);
```

```
            colR3.setPixelColor(centerx + 4, Colors::BLACK);
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        }
    }
    if (currentZ == 1)
    {
        switch (centery1b)
        {
        case PIN:

            col1.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_2:

            col2.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_3:

            col3.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_4:

            col4.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_LEFT1:
```

```
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL3.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_RIGHT1:

            colR1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_RIGHT2:

            colR2.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_RIGHT3:

            colR3.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        }
    }
}
void turnoffT(int centerx)
{
    if (currentT == 0)
    {
        switch (centery1b)
        {
        case PIN:

            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 4, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
```

```
            break;
        case PIN_2:
            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 4, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_3:
            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 4, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_4:
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 4, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_LEFT1:

            colL2.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 4, Colors::BLACK);
            colL3.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_RIGHT1:
            col1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 4, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_RIGHT2:
            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 4, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_RIGHT3:
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx + 4, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            break;
```

```
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx, Colors::BLACK);
            colR3.setPixelColor(centerx + 2, Colors::BLACK);
            colR3.setPixelColor(centerx + 4, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        }
    }
    if (currentT == 1)
    {
        switch (centery1b)
        {
        case PIN:

            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_2:
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_3:
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_4:
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL3.setPixelColor(centerx + 2, Colors::BLACK);

            break;
        case PIN_RIGHT1:
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            break;
```

```
        case PIN_RIGHT2:
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_RIGHT3:
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            break;
        }
    }
    if (currentT == 2)
    {
        switch (centery1b)
        {
        case PIN:

            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_2:
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 4, Colors::BLACK);
            break;
        case PIN_3:
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 4, Colors::BLACK);
            break;
        case PIN_4:
```

```
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_LEFT1:
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL3.setPixelColor(centerx, Colors::BLACK);
            colL3.setPixelColor(centerx + 2, Colors::BLACK);
            colL3.setPixelColor(centerx + 4, Colors::BLACK);

            break;
        case PIN_RIGHT1:
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 4, Colors::BLACK);
            break;
        case PIN_RIGHT2:
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 4, Colors::BLACK);
            break;
        case PIN_RIGHT3:
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 4, Colors::BLACK);
            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx + 4, Colors::BLACK);
            break;
        }
    }
    if (currentT == 3)
    {
        switch (centery1b)
        {
        case PIN:
```

```
            col2.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_2:
            col3.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);
            break;
        case PIN_3:
            col4.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);
            break;
        case PIN_4:
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);
            colL3.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_RIGHT1:
            col1.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            break;
        case PIN_RIGHT2:
            colR1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            break;
        case PIN_RIGHT3:
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx, Colors::BLACK);
```

```
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            break;
        }
    }
}

void turnoffI(int centerx)
{
    if (currentI == 0)
    {
        switch (centery1b)
        {
        case PIN:

            col3.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx + 2, Colors::BLACK);
            col3.setPixelColor(centerx + 4, Colors::BLACK);
            col3.setPixelColor(centerx + 6, Colors::BLACK);

            break;
        case PIN_2:

            col4.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx + 2, Colors::BLACK);
            col4.setPixelColor(centerx + 4, Colors::BLACK);
            col4.setPixelColor(centerx + 6, Colors::BLACK);

            break;
        case PIN_3:

            colL1.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx + 2, Colors::BLACK);
            colL1.setPixelColor(centerx + 4, Colors::BLACK);
            colL1.setPixelColor(centerx + 6, Colors::BLACK);

            break;
        case PIN_4:

            colL2.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx + 2, Colors::BLACK);
            colL2.setPixelColor(centerx + 4, Colors::BLACK);
            colL2.setPixelColor(centerx + 6, Colors::BLACK);
```

```
            break;
        case PIN_LEFT1:

            colL3.setPixelColor(centerx, Colors::BLACK);
            colL3.setPixelColor(centerx + 2, Colors::BLACK);
            colL3.setPixelColor(centerx + 4, Colors::BLACK);
            colL3.setPixelColor(centerx + 6, Colors::BLACK);
            break;
        case PIN_RIGHT1:

            col2.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx + 2, Colors::BLACK);
            col2.setPixelColor(centerx + 4, Colors::BLACK);
            col2.setPixelColor(centerx + 6, Colors::BLACK);

            break;
        case PIN_RIGHT2:

            col1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx + 2, Colors::BLACK);
            col1.setPixelColor(centerx + 4, Colors::BLACK);
            col1.setPixelColor(centerx + 6, Colors::BLACK);

            break;
        case PIN_RIGHT3:

            colR1.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx + 2, Colors::BLACK);
            colR1.setPixelColor(centerx + 4, Colors::BLACK);
            colR1.setPixelColor(centerx + 6, Colors::BLACK);

            break;
        case PIN_RIGHT3 - 1:
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx + 2, Colors::BLACK);
            colR2.setPixelColor(centerx + 4, Colors::BLACK);
            colR2.setPixelColor(centerx + 6, Colors::BLACK);

            break;
        case PIN_RIGHT3 - 2:
            colR3.setPixelColor(centerx, Colors::BLACK);
            colR3.setPixelColor(centerx + 2, Colors::BLACK);
            colR3.setPixelColor(centerx + 4, Colors::BLACK);
            colR3.setPixelColor(centerx + 6, Colors::BLACK);
```

```
            break;
        }
    }
    if (currentI == 1)
    {
        switch (centery1b)
        {
        case PIN:

            col1.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_2:

            col2.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_3:

            col3.setPixelColor(centerx, Colors::BLACK);
            col4.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_4:

            col4.setPixelColor(centerx, Colors::BLACK);
            colL1.setPixelColor(centerx, Colors::BLACK);
            colL2.setPixelColor(centerx, Colors::BLACK);
            colL3.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_RIGHT1:

            colR1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);
            col3.setPixelColor(centerx, Colors::BLACK);
```

```
            break;
        case PIN_RIGHT2:

            colR2.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);
            col2.setPixelColor(centerx, Colors::BLACK);

            break;
        case PIN_RIGHT3:

            colR3.setPixelColor(centerx, Colors::BLACK);
            colR2.setPixelColor(centerx, Colors::BLACK);
            colR1.setPixelColor(centerx, Colors::BLACK);
            col1.setPixelColor(centerx, Colors::BLACK);

            break;
        }
    }
}
// light
void light0(int centerx)
{

    switch (centery1)
    {
    case PIN:
        col2.setPixelColor(centerx, color0);
        col2.setPixelColor(centerx + 2, color0);
        col3.setPixelColor(centerx, color0);
        col3.setPixelColor(centerx + 2, color0);
        break;
    case PIN_2:
        col3.setPixelColor(centerx, color0);
        col3.setPixelColor(centerx + 2, color0);
        col4.setPixelColor(centerx, color0);
        col4.setPixelColor(centerx + 2, color0);
        break;
    case PIN_3:
        col4.setPixelColor(centerx, color0);
        col4.setPixelColor(centerx + 2, color0);
        colL1.setPixelColor(centerx, color0);
        colL1.setPixelColor(centerx + 2, color0);
        break;
```

```
    case PIN_4:
        colL1.setPixelColor(centerx, color0);
        colL1.setPixelColor(centerx + 2, color0);
        colL2.setPixelColor(centerx, color0);
        colL2.setPixelColor(centerx + 2, color0);
        break;
    case PIN_LEFT1:
        colL2.setPixelColor(centerx, color0);
        colL2.setPixelColor(centerx + 2, color0);
        colL3.setPixelColor(centerx, color0);
        colL3.setPixelColor(centerx + 2, color0);
        break;
    case PIN_RIGHT1:
        col1.setPixelColor(centerx, color0);
        col1.setPixelColor(centerx + 2, color0);
        col2.setPixelColor(centerx, color0);
        col2.setPixelColor(centerx + 2, color0);
        break;
    case PIN_RIGHT2:
        colR1.setPixelColor(centerx, color0);
        colR1.setPixelColor(centerx + 2, color0);
        col1.setPixelColor(centerx, color0);
        col1.setPixelColor(centerx + 2, color0);
        break;
    case PIN_RIGHT3:
        colR2.setPixelColor(centerx, color0);
        colR2.setPixelColor(centerx + 2, color0);
        colR1.setPixelColor(centerx, color0);
        colR1.setPixelColor(centerx + 2, color0);
        break;
    case PIN_RIGHT3 - 1:
        colR3.setPixelColor(centerx, color0);
        colR3.setPixelColor(centerx + 2, color0);
        colR2.setPixelColor(centerx, color0);
        colR2.setPixelColor(centerx + 2, color0);
        break;
    }
}

void lightL(int centerx)
{
    if (currentL == 0)
    {
        switch (centery1)
        {
```

```
        case PIN:

            col2.setPixelColor(centerx + 4, colorL);
            col3.setPixelColor(centerx, colorL);
            col3.setPixelColor(centerx + 2, colorL);
            col3.setPixelColor(centerx + 4, colorL);

            break;
        case PIN_2:

            col3.setPixelColor(centerx + 4, colorL);
            col4.setPixelColor(centerx, colorL);
            col4.setPixelColor(centerx + 2, colorL);
            col4.setPixelColor(centerx + 4, colorL);

            break;
        case PIN_3:

            col4.setPixelColor(centerx + 4, colorL);
            colL1.setPixelColor(centerx, colorL);
            colL1.setPixelColor(centerx + 2, colorL);
            colL1.setPixelColor(centerx + 4, colorL);

            break;
        case PIN_4:

            colL1.setPixelColor(centerx + 4, colorL);
            colL2.setPixelColor(centerx, colorL);
            colL2.setPixelColor(centerx + 2, colorL);
            colL2.setPixelColor(centerx + 4, colorL);

            break;
        case PIN_LEFT1:

            colL2.setPixelColor(centerx + 4, colorL);
            colL3.setPixelColor(centerx, colorL);
            colL3.setPixelColor(centerx + 2, colorL);
            colL3.setPixelColor(centerx + 4, colorL);
            break;
        case PIN_RIGHT1:

            col1.setPixelColor(centerx + 4, colorL);
            col2.setPixelColor(centerx, colorL);
            col2.setPixelColor(centerx + 2, colorL);
            col2.setPixelColor(centerx + 4, colorL);
```

```
            break;
        case PIN_RIGHT2:

            colR1.setPixelColor(centerx + 4, colorL);
            col1.setPixelColor(centerx, colorL);
            col1.setPixelColor(centerx + 2, colorL);
            col1.setPixelColor(centerx + 4, colorL);

            break;
        case PIN_RIGHT3:

            colR2.setPixelColor(centerx + 4, colorL);
            colR1.setPixelColor(centerx, colorL);
            colR1.setPixelColor(centerx + 2, colorL);
            colR1.setPixelColor(centerx + 4, colorL);

            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx + 4, colorL);
            colR2.setPixelColor(centerx, colorL);
            colR2.setPixelColor(centerx + 2, colorL);
            colR2.setPixelColor(centerx + 4, colorL);

            break;
        }
    }
    if (currentL == 1)
    {
        switch (centery1)
        {
        case PIN:

            col1.setPixelColor(centerx, colorL);
            col2.setPixelColor(centerx, colorL);
            col3.setPixelColor(centerx, colorL);
            col3.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_2:

            col2.setPixelColor(centerx, colorL);
            col3.setPixelColor(centerx, colorL);
            col4.setPixelColor(centerx, colorL);
            col4.setPixelColor(centerx + 2, colorL);
```

```
            break;
        case PIN_3:

            col3.setPixelColor(centerx, colorL);
            col4.setPixelColor(centerx, colorL);
            colL1.setPixelColor(centerx, colorL);
            colL1.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_4:

            col4.setPixelColor(centerx, colorL);
            colL1.setPixelColor(centerx, colorL);
            colL2.setPixelColor(centerx, colorL);
            colL2.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_LEFT1:

            colL1.setPixelColor(centerx, colorL);
            colL2.setPixelColor(centerx, colorL);
            colL3.setPixelColor(centerx, colorL);
            colL3.setPixelColor(centerx + 2, colorL);
            break;
        case PIN_RIGHT1:

            colR1.setPixelColor(centerx, colorL);
            col1.setPixelColor(centerx, colorL);
            col2.setPixelColor(centerx, colorL);
            col2.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_RIGHT2:

            colR2.setPixelColor(centerx, colorL);
            colR1.setPixelColor(centerx, colorL);
            col1.setPixelColor(centerx, colorL);
            col1.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_RIGHT3:

            colR3.setPixelColor(centerx, colorL);
            colR2.setPixelColor(centerx, colorL);
```

```
            colR1.setPixelColor(centerx, colorL);
            colR1.setPixelColor(centerx + 2, colorL);

            break;
        }
    }
    if (currentL == 2)
    {
        switch (centery1)
        {
        case PIN:

            col2.setPixelColor(centerx, colorL);
            col2.setPixelColor(centerx + 2, colorL);
            col2.setPixelColor(centerx + 4, colorL);
            col3.setPixelColor(centerx, colorL);

            break;
        case PIN_2:

            col3.setPixelColor(centerx, colorL);
            col3.setPixelColor(centerx + 2, colorL);
            col3.setPixelColor(centerx + 4, colorL);
            col4.setPixelColor(centerx, colorL);

            break;
        case PIN_3:

            col4.setPixelColor(centerx, colorL);
            col4.setPixelColor(centerx + 2, colorL);
            col4.setPixelColor(centerx + 4, colorL);
            colL1.setPixelColor(centerx, colorL);

            break;
        case PIN_4:

            colL1.setPixelColor(centerx, colorL);
            colL1.setPixelColor(centerx + 2, colorL);
            colL1.setPixelColor(centerx + 4, colorL);
            colL2.setPixelColor(centerx, colorL);

            break;
        case PIN_LEFT1:

            colL2.setPixelColor(centerx, colorL);
```

```
            colL2.setPixelColor(centerx + 2, colorL);
            colL2.setPixelColor(centerx + 4, colorL);
            colL3.setPixelColor(centerx, colorL);
            break;
        case PIN_RIGHT1:

            col1.setPixelColor(centerx, colorL);
            col1.setPixelColor(centerx + 2, colorL);
            col1.setPixelColor(centerx + 4, colorL);
            col2.setPixelColor(centerx, colorL);

            break;
        case PIN_RIGHT2:

            colR1.setPixelColor(centerx, colorL);
            colR1.setPixelColor(centerx + 2, colorL);
            colR1.setPixelColor(centerx + 4, colorL);
            col1.setPixelColor(centerx, colorL);

            break;
        case PIN_RIGHT3:

            colR2.setPixelColor(centerx, colorL);
            colR2.setPixelColor(centerx + 2, colorL);
            colR2.setPixelColor(centerx + 4, colorL);
            colR1.setPixelColor(centerx, colorL);

            break;
        case PIN_RIGHT3 - 1:

            colR3.setPixelColor(centerx, colorL);
            colR3.setPixelColor(centerx + 2, colorL);
            colR3.setPixelColor(centerx + 4, colorL);
            colR2.setPixelColor(centerx, colorL);

            break;
        }
    }
    if (currentL == 3)
    {
        switch (centery1)
        {
        case PIN:

            col1.setPixelColor(centerx, colorL);
```

```
            col1.setPixelColor(centerx + 2, colorL);
            col2.setPixelColor(centerx + 2, colorL);
            col3.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_2:

            col2.setPixelColor(centerx, colorL);
            col2.setPixelColor(centerx + 2, colorL);
            col3.setPixelColor(centerx + 2, colorL);
            col4.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_3:

            col3.setPixelColor(centerx, colorL);
            col3.setPixelColor(centerx + 2, colorL);
            col4.setPixelColor(centerx + 2, colorL);
            colL1.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_4:

            col4.setPixelColor(centerx, colorL);
            col4.setPixelColor(centerx + 2, colorL);
            colL1.setPixelColor(centerx + 2, colorL);
            colL2.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_LEFT1:

            colL1.setPixelColor(centerx, colorL);
            colL1.setPixelColor(centerx + 2, colorL);
            colL2.setPixelColor(centerx + 2, colorL);
            colL3.setPixelColor(centerx + 2, colorL);
            break;
        case PIN_RIGHT1:

            colR1.setPixelColor(centerx, colorL);
            colR1.setPixelColor(centerx + 2, colorL);
            col1.setPixelColor(centerx + 2, colorL);
            col2.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_RIGHT2:
```

```
            colR2.setPixelColor(centerx, colorL);
            colR2.setPixelColor(centerx + 2, colorL);
            colR1.setPixelColor(centerx + 2, colorL);
            col1.setPixelColor(centerx + 2, colorL);

            break;
        case PIN_RIGHT3:

            colR3.setPixelColor(centerx, colorL);
            colR3.setPixelColor(centerx + 2, colorL);
            colR2.setPixelColor(centerx + 2, colorL);
            colR1.setPixelColor(centerx + 2, colorL);

            break;
        }
    }
}

void lightZ(int centerx)
{
    if (currentZ == 0)
    {
        switch (centery1)
        {
        case PIN:

            col2.setPixelColor(centerx + 2, colorZ);
            col2.setPixelColor(centerx + 4, colorZ);
            col3.setPixelColor(centerx, colorZ);
            col3.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_2:

            col3.setPixelColor(centerx + 2, colorZ);
            col3.setPixelColor(centerx + 4, colorZ);
            col4.setPixelColor(centerx, colorZ);
            col4.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_3:

            col4.setPixelColor(centerx + 2, colorZ);
            col4.setPixelColor(centerx + 4, colorZ);
```

```
            colL1.setPixelColor(centerx, colorZ);
            colL1.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_4:

            colL1.setPixelColor(centerx + 2, colorZ);
            colL1.setPixelColor(centerx + 4, colorZ);
            colL2.setPixelColor(centerx, colorZ);
            colL2.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_LEFT1:

            colL2.setPixelColor(centerx + 2, colorZ);
            colL2.setPixelColor(centerx + 4, colorZ);
            colL3.setPixelColor(centerx, colorZ);
            colL3.setPixelColor(centerx + 2, colorZ);
            break;
        case PIN_RIGHT1:

            col1.setPixelColor(centerx + 2, colorZ);
            col1.setPixelColor(centerx + 4, colorZ);
            col2.setPixelColor(centerx, colorZ);
            col2.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_RIGHT2:

            colR1.setPixelColor(centerx + 2, colorZ);
            colR1.setPixelColor(centerx + 4, colorZ);
            col1.setPixelColor(centerx, colorZ);
            col1.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_RIGHT3:

            colR2.setPixelColor(centerx + 2, colorZ);
            colR2.setPixelColor(centerx + 4, colorZ);
            colR1.setPixelColor(centerx, colorZ);
            colR1.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx + 2, colorZ);
```

```
            colR3.setPixelColor(centerx + 4, colorZ);
            colR2.setPixelColor(centerx, colorZ);
            colR2.setPixelColor(centerx + 2, colorZ);

            break;
        }
    }
    if (currentZ == 1)
    {
        switch (centery1)
        {
        case PIN:

            col1.setPixelColor(centerx, colorZ);
            col2.setPixelColor(centerx, colorZ);
            col2.setPixelColor(centerx + 2, colorZ);
            col3.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_2:

            col2.setPixelColor(centerx, colorZ);
            col3.setPixelColor(centerx, colorZ);
            col3.setPixelColor(centerx + 2, colorZ);
            col4.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_3:

            col3.setPixelColor(centerx, colorZ);
            col4.setPixelColor(centerx, colorZ);
            col4.setPixelColor(centerx + 2, colorZ);
            colL1.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_4:

            col4.setPixelColor(centerx, colorZ);
            colL1.setPixelColor(centerx, colorZ);
            colL1.setPixelColor(centerx + 2, colorZ);
            colL2.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_LEFT1:
```

```
            colL1.setPixelColor(centerx, colorZ);
            colL2.setPixelColor(centerx, colorZ);
            colL2.setPixelColor(centerx + 2, colorZ);
            colL3.setPixelColor(centerx + 2, colorZ);
            break;
        case PIN_RIGHT1:

            colR1.setPixelColor(centerx, colorZ);
            col1.setPixelColor(centerx, colorZ);
            col1.setPixelColor(centerx + 2, colorZ);
            col2.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_RIGHT2:

            colR2.setPixelColor(centerx, colorZ);
            colR1.setPixelColor(centerx, colorZ);
            colR1.setPixelColor(centerx + 2, colorZ);
            col1.setPixelColor(centerx + 2, colorZ);

            break;
        case PIN_RIGHT3:

            colR3.setPixelColor(centerx, colorZ);
            colR2.setPixelColor(centerx, colorZ);
            colR2.setPixelColor(centerx + 2, colorZ);
            colR1.setPixelColor(centerx + 2, colorZ);

            break;
        }
    }
}
void lightT(int centerx)
{
    if (currentT == 0)
    {
        switch (centery1)
        {
        case PIN:

            col2.setPixelColor(centerx, colorT);
            col2.setPixelColor(centerx + 2, colorT);
            col2.setPixelColor(centerx + 4, colorT);
            col3.setPixelColor(centerx + 2, colorT);
```

```
            break;
        case PIN_2:
            col3.setPixelColor(centerx, colorT);
            col3.setPixelColor(centerx + 2, colorT);
            col3.setPixelColor(centerx + 4, colorT);
            col4.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_3:
            col4.setPixelColor(centerx, colorT);
            col4.setPixelColor(centerx + 2, colorT);
            col4.setPixelColor(centerx + 4, colorT);
            colL1.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_4:
            colL1.setPixelColor(centerx, colorT);
            colL1.setPixelColor(centerx + 2, colorT);
            colL1.setPixelColor(centerx + 4, colorT);
            colL2.setPixelColor(centerx + 2, colorT);

            break;
        case PIN_LEFT1:

            colL2.setPixelColor(centerx, colorT);
            colL2.setPixelColor(centerx + 2, colorT);
            colL2.setPixelColor(centerx + 4, colorT);
            colL3.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_RIGHT1:
            col1.setPixelColor(centerx, colorT);
            col1.setPixelColor(centerx + 2, colorT);
            col1.setPixelColor(centerx + 4, colorT);
            col2.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_RIGHT2:
            colR1.setPixelColor(centerx, colorT);
            colR1.setPixelColor(centerx + 2, colorT);
            colR1.setPixelColor(centerx + 4, colorT);
            col1.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_RIGHT3:
            colR2.setPixelColor(centerx, colorT);
            colR2.setPixelColor(centerx + 2, colorT);
            colR2.setPixelColor(centerx + 4, colorT);
            colR1.setPixelColor(centerx + 2, colorT);
            break;
```

```
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx, colorT);
            colR3.setPixelColor(centerx + 2, colorT);
            colR3.setPixelColor(centerx + 4, colorT);
            colR2.setPixelColor(centerx + 2, colorT);
            break;
        }
    }
    if (currentT == 1)
    {
        switch (centery1)
        {
        case PIN:

            col2.setPixelColor(centerx + 2, colorT);
            col3.setPixelColor(centerx, colorT);
            col3.setPixelColor(centerx + 2, colorT);
            col4.setPixelColor(centerx + 2, colorT);

            break;
        case PIN_2:
            col3.setPixelColor(centerx + 2, colorT);
            col4.setPixelColor(centerx, colorT);
            col4.setPixelColor(centerx + 2, colorT);
            colL1.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_3:
            col4.setPixelColor(centerx + 2, colorT);
            colL1.setPixelColor(centerx, colorT);
            colL1.setPixelColor(centerx + 2, colorT);
            colL2.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_4:
            colL1.setPixelColor(centerx + 2, colorT);
            colL2.setPixelColor(centerx, colorT);
            colL2.setPixelColor(centerx + 2, colorT);
            colL3.setPixelColor(centerx + 2, colorT);

            break;
        case PIN_RIGHT1:
            col1.setPixelColor(centerx + 2, colorT);
            col2.setPixelColor(centerx, colorT);
            col2.setPixelColor(centerx + 2, colorT);
            col3.setPixelColor(centerx + 2, colorT);
            break;
```

```
        case PIN_RIGHT2:
            colR1.setPixelColor(centerx + 2, colorT);
            col1.setPixelColor(centerx, colorT);
            col1.setPixelColor(centerx + 2, colorT);
            col2.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_RIGHT3:
            colR2.setPixelColor(centerx + 2, colorT);
            colR1.setPixelColor(centerx, colorT);
            colR1.setPixelColor(centerx + 2, colorT);
            col1.setPixelColor(centerx + 2, colorT);
            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx + 2, colorT);
            colR2.setPixelColor(centerx, colorT);
            colR2.setPixelColor(centerx + 2, colorT);
            colR1.setPixelColor(centerx + 2, colorT);
            break;
        }
    }
    if (currentT == 2)
    {
        switch (centery1)
        {
        case PIN:

            col2.setPixelColor(centerx + 2, colorT);
            col3.setPixelColor(centerx, colorT);
            col3.setPixelColor(centerx + 2, colorT);
            col3.setPixelColor(centerx + 4, colorT);

            break;
        case PIN_2:
            col3.setPixelColor(centerx + 2, colorT);
            col4.setPixelColor(centerx, colorT);
            col4.setPixelColor(centerx + 2, colorT);
            col4.setPixelColor(centerx + 4, colorT);
            break;
        case PIN_3:
            col4.setPixelColor(centerx + 2, colorT);
            colL1.setPixelColor(centerx, colorT);
            colL1.setPixelColor(centerx + 2, colorT);
            colL1.setPixelColor(centerx + 4, colorT);
            break;
        case PIN_4:
```

```
            colL1.setPixelColor(centerx + 2, colorT);
            colL2.setPixelColor(centerx, colorT);
            colL2.setPixelColor(centerx + 2, colorT);
            colL2.setPixelColor(centerx + 4, colorT);

            break;
        case PIN_LEFT1:
            colL2.setPixelColor(centerx + 2, colorT);
            colL3.setPixelColor(centerx, colorT);
            colL3.setPixelColor(centerx + 2, colorT);
            colL3.setPixelColor(centerx + 4, colorT);

            break;
        case PIN_RIGHT1:
            col1.setPixelColor(centerx + 2, colorT);
            col2.setPixelColor(centerx, colorT);
            col2.setPixelColor(centerx + 2, colorT);
            col2.setPixelColor(centerx + 4, colorT);
            break;
        case PIN_RIGHT2:
            colR1.setPixelColor(centerx + 2, colorT);
            col1.setPixelColor(centerx, colorT);
            col1.setPixelColor(centerx + 2, colorT);
            col1.setPixelColor(centerx + 4, colorT);
            break;
        case PIN_RIGHT3:
            colR2.setPixelColor(centerx + 2, colorT);
            colR1.setPixelColor(centerx, colorT);
            colR1.setPixelColor(centerx + 2, colorT);
            colR1.setPixelColor(centerx + 4, colorT);
            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx + 2, colorT);
            colR2.setPixelColor(centerx, colorT);
            colR2.setPixelColor(centerx + 2, colorT);
            colR2.setPixelColor(centerx + 4, colorT);
            break;
        }
    }
    if (currentT == 3)
    {
        switch (centery1)
        {
        case PIN:
```

```
            col2.setPixelColor(centerx, colorT);
            col3.setPixelColor(centerx + 2, colorT);
            col3.setPixelColor(centerx, colorT);
            col4.setPixelColor(centerx, colorT);

            break;
        case PIN_2:
            col3.setPixelColor(centerx, colorT);
            col4.setPixelColor(centerx + 2, colorT);
            col4.setPixelColor(centerx, colorT);
            colL1.setPixelColor(centerx, colorT);
            break;
        case PIN_3:
            col4.setPixelColor(centerx, colorT);
            colL1.setPixelColor(centerx + 2, colorT);
            colL1.setPixelColor(centerx, colorT);
            colL2.setPixelColor(centerx, colorT);
            break;
        case PIN_4:
            colL1.setPixelColor(centerx, colorT);
            colL2.setPixelColor(centerx + 2, colorT);
            colL2.setPixelColor(centerx, colorT);
            colL3.setPixelColor(centerx, colorT);

            break;
        case PIN_RIGHT1:
            col1.setPixelColor(centerx, colorT);
            col2.setPixelColor(centerx + 2, colorT);
            col2.setPixelColor(centerx, colorT);
            col3.setPixelColor(centerx, colorT);
            break;
        case PIN_RIGHT2:
            colR1.setPixelColor(centerx, colorT);
            col1.setPixelColor(centerx + 2, colorT);
            col1.setPixelColor(centerx, colorT);
            col2.setPixelColor(centerx, colorT);
            break;
        case PIN_RIGHT3:
            colR2.setPixelColor(centerx, colorT);
            colR1.setPixelColor(centerx + 2, colorT);
            colR1.setPixelColor(centerx, colorT);
            col1.setPixelColor(centerx, colorT);
            break;
        case PIN_RIGHT3 - 1:
            colR3.setPixelColor(centerx, colorT);
```

```
            colR2.setPixelColor(centerx + 2, colorT);
            colR2.setPixelColor(centerx, colorT);
            colR1.setPixelColor(centerx, colorT);
            break;
        }
    }
}

void lightI(int centerx)
{
    if (currentI == 0)
    {
        switch (centery1)
        {
        case PIN:

            col3.setPixelColor(centerx, colorI);
            col3.setPixelColor(centerx + 2, colorI);
            col3.setPixelColor(centerx + 4, colorI);
            col3.setPixelColor(centerx + 6, colorI);

            break;
        case PIN_2:

            col4.setPixelColor(centerx, colorI);
            col4.setPixelColor(centerx + 2, colorI);
            col4.setPixelColor(centerx + 4, colorI);
            col4.setPixelColor(centerx + 6, colorI);

            break;
        case PIN_3:

            colL1.setPixelColor(centerx, colorI);
            colL1.setPixelColor(centerx + 2, colorI);
            colL1.setPixelColor(centerx + 4, colorI);
            colL1.setPixelColor(centerx + 6, colorI);

            break;
        case PIN_4:

            colL2.setPixelColor(centerx, colorI);
            colL2.setPixelColor(centerx + 2, colorI);
            colL2.setPixelColor(centerx + 4, colorI);
            colL2.setPixelColor(centerx + 6, colorI);
```

```
            break;
        case PIN_LEFT1:

            colL3.setPixelColor(centerx, colorI);
            colL3.setPixelColor(centerx + 2, colorI);
            colL3.setPixelColor(centerx + 4, colorI);
            colL3.setPixelColor(centerx + 6, colorI);
            break;
        case PIN_RIGHT1:

            col2.setPixelColor(centerx, colorI);
            col2.setPixelColor(centerx + 2, colorI);
            col2.setPixelColor(centerx + 4, colorI);
            col2.setPixelColor(centerx + 6, colorI);

            break;
        case PIN_RIGHT2:

            col1.setPixelColor(centerx, colorI);
            col1.setPixelColor(centerx + 2, colorI);
            col1.setPixelColor(centerx + 4, colorI);
            col1.setPixelColor(centerx + 6, colorI);

            break;
        case PIN_RIGHT3:

            colR1.setPixelColor(centerx, colorI);
            colR1.setPixelColor(centerx + 2, colorI);
            colR1.setPixelColor(centerx + 4, colorI);
            colR1.setPixelColor(centerx + 6, colorI);

            break;
        case PIN_RIGHT3 - 1:
            colR2.setPixelColor(centerx, colorI);
            colR2.setPixelColor(centerx + 2, colorI);
            colR2.setPixelColor(centerx + 4, colorI);
            colR2.setPixelColor(centerx + 6, colorI);

            break;
        case PIN_RIGHT3 - 2:
            colR3.setPixelColor(centerx, colorI);
            colR3.setPixelColor(centerx + 2, colorI);
            colR3.setPixelColor(centerx + 4, colorI);
            colR3.setPixelColor(centerx + 6, colorI);
```

```
            break;
        }
    }
    if (currentI == 1)
    {
        switch (centery1)
        {
        case PIN:

            col1.setPixelColor(centerx, colorI);
            col2.setPixelColor(centerx, colorI);
            col3.setPixelColor(centerx, colorI);
            col4.setPixelColor(centerx, colorI);

            break;
        case PIN_2:

            col2.setPixelColor(centerx, colorI);
            col3.setPixelColor(centerx, colorI);
            col4.setPixelColor(centerx, colorI);
            colL1.setPixelColor(centerx, colorI);

            break;
        case PIN_3:

            col3.setPixelColor(centerx, colorI);
            col4.setPixelColor(centerx, colorI);
            colL1.setPixelColor(centerx, colorI);
            colL2.setPixelColor(centerx, colorI);

            break;
        case PIN_4:

            col4.setPixelColor(centerx, colorI);
            colL1.setPixelColor(centerx, colorI);
            colL2.setPixelColor(centerx, colorI);
            colL3.setPixelColor(centerx, colorI);

            break;
        case PIN_RIGHT1:

            colR1.setPixelColor(centerx, colorI);
            col1.setPixelColor(centerx, colorI);
            col2.setPixelColor(centerx, colorI);
            col3.setPixelColor(centerx, colorI);
```

```
            break;
        case PIN_RIGHT2:

            colR2.setPixelColor(centerx, colorI);
            colR1.setPixelColor(centerx, colorI);
            col1.setPixelColor(centerx, colorI);
            col2.setPixelColor(centerx, colorI);

            break;
        case PIN_RIGHT3:

            colR3.setPixelColor(centerx, colorI);
            colR2.setPixelColor(centerx, colorI);
            colR1.setPixelColor(centerx, colorI);
            col1.setPixelColor(centerx, colorI);

            break;
        }
    }
}
// light for shape prediction
void lightOPre()
{
    colPL.setPixelColor(0, colorO);
    colPL.setPixelColor(2, colorO);
    colPR.setPixelColor(0, colorO);
    colPR.setPixelColor(2, colorO);
}
void lightTPre()
{
    colPR.setPixelColor(0, colorT);
    colPR.setPixelColor(2, colorT);
    colPR.setPixelColor(4, colorT);
    colPL.setPixelColor(2, colorT);
}
void lightZPre()
{
    colPL.setPixelColor(0, colorZ);
    colPL.setPixelColor(2, colorZ);
    colPR.setPixelColor(2, colorZ);
    colPR.setPixelColor(4, colorZ);
}
void lightLPre()
{
```

```
    colPR.setPixelColor(4, colorL);
    colPL.setPixelColor(0, colorL);
    colPL.setPixelColor(2, colorL);
    colPL.setPixelColor(4, colorL);
}
void lightIPre()
{
    colPL.setPixelColor(0, colorI);
    colPL.setPixelColor(2, colorI);
    colPL.setPixelColor(4, colorI);
    colPL.setPixelColor(6, colorI);
}

void displayScore(int score)
{
    int ones = 0;
    int tens = 0;
    if (score <= 9)
    {
        ones = score;
    }
    else if (score > 9)
    {
        ones = score % 10;
        tens = score / 10;
    }

    for (int j = 0; j < 7; j++)
    {
        digitalWrite(ones_score_pin[j], numbers[ones][j]);
        digitalWrite(tens_score_pin[j], numbers[tens][j]);
    }
}

void displayHighscore(int score)
{
    int ones = 0;
    int tens = 0;
    if (score <= 9)
    {
        ones = score;
    }
    else if (score > 9)
    {
        ones = score % 10;
```

```
        tens = score / 10;
    }

    for (int j = 0; j < 7; j++)
    {
        digitalWrite(ones_highscore_pin[j], numbers[ones][j]);
        digitalWrite(tens_highscore_pin[j], numbers[tens][j]);
    }
}

void audioControl()
{
    char wavFile[33];
    switch (audio_state)
    {
    case add_point:
        tone(buzzer, 1000);
        delay(20);
        noTone(buzzer);
        delay(10);
        tone(buzzer, 4500);
        delay(100);
        noTone(buzzer);
        break; // Play point sound on pins 5,2
    case game_over:
        strcpy_P(wavFile, wav_table[0]);
        audio.play(wavFile);
        break; // Play gameover sound on pins 5,2
    case background:
        strcpy_P(wavFile, wav_table[1]);
        audio.play(wavFile);
        break; // Play background music on pins 6,7
    case increase:
        audio.volume(1);
        break; // Increase volume by 1
    case decrease:
        audio.volume(0);
        break; // Decrease volume by 1
    case mute_bg:
        audio.setVolume(0);
        break; // mute background on output 0
    case unmute_bg:
        audio.setVolume(3);
        break; // Unmute background on output 0
    case stop_all:
```

```
        audio.stopPlayback();
        break; // Stop all playback
    default:
        break;
    }
    audio_state = default_null;
}
```