Tetris Final Code

```
#include <Arduino.h>
#include <FastLED.h>

//Definitions for all the pins we need and all the LED board
parameters
#define LED_PIN     2
#define NUM_LEDS    256
#define LED_TYPE    WS2812
#define COLOR_ORDER GRB
#define BRIGHTNESS  20
#define MATRIX_WIDTH  16
#define MATRIX_HEIGHT 16

//assignments for all the global variables used throughout the
entire program (keeping time/pace, noises, and score)
unsigned long fallTimer = 0;
unsigned long fallInterval = 1000;
unsigned long score = 0;
const int speakerPin = 9;
const int blipFrequency = 30;
const int lockFrequency = 200;
const int littleFrequency = 900;
const int thungFrequency = 1500;
const int GOF1 = 800;
const int GOF2 = 500;
const int GOF3 = 200;


//all of the Tetris structures, Tetrimino shapes, colors, and
position logic initiaition
CRGB leds[NUM_LEDS];

struct Tetrimino {
  bool shape[4][4];
```

```
  CRGB color;
};
Tetrimino currentTetrimino, nextTetrimino;
int posX = 0, posY = 0; //initialize position of piece
bool board[MATRIX_HEIGHT][MATRIX_WIDTH - 4] = {0};
const bool TETRIMINOS[7][4][4] = {
  // I-Block
  {
    {0, 0, 0, 0},
    {1, 1, 1, 1},
    {0, 0, 0, 0},
    {0, 0, 0, 0}
  },
  // J-Block
  {
    {1, 0, 0, 0},
    {1, 1, 1, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}
  },
  // L-Block
  {
    {0, 0, 1, 0},
    {1, 1, 1, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}
  },
  // O-Block
  {
    {1, 1, 0, 0},
    {1, 1, 0, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}
  },
  // S-Block
  {
```

```
    {0, 1, 1, 0},
    {1, 1, 0, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}
  },
  // Z-Block
  {
    {1, 1, 0, 0},
    {0, 1, 1, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}
  },
  // T-Block
  {
    {0, 1, 0, 0},
    {1, 1, 1, 0},
    {0, 0, 0, 0},
    {0, 0, 0, 0}
  }
};
const CRGB TETRIMINO_COLORS[7] = {
  CRGB::Blue,    // I-Block
  CRGB::Green,   // J-Block
  CRGB::Orange,  // L-Block
  CRGB::Yellow,  // O-Block
  CRGB::Magenta, // S-Block
  CRGB::Red,     // Z-Block
  CRGB::Cyan     // T-Block
};

//digits to display score at game over
const bool digits[10][5][4] = {
  {//0
    {1, 1, 1, 0},
    {1, 0, 1, 0},
    {1, 0, 1, 0},
```

```
    {1, 0, 1, 0},
    {1, 1, 1, 0}
  },
  {//1
    {0, 1, 0, 0},
    {0, 1, 0, 0},
    {0, 1, 0, 0},
    {0, 1, 0, 0},
    {0, 1, 0, 0}
  },
  {//2
    {1, 1, 1, 0},
    {0, 0, 1, 0},
    {1, 1, 1, 0},
    {1, 0, 0, 0},
    {1, 1, 1, 0}
  },
  {//3
    {1, 1, 1, 0},
    {1, 0, 0, 0},
    {1, 1, 1, 0},
    {1, 0, 0, 0},
    {1, 1, 1, 0}
  },
  {//4
    {1, 0, 0, 0},
    {1, 0, 0, 0},
    {1, 1, 1, 0},
    {1, 0, 1, 0},
    {1, 0, 1, 0}
  },
  {//5
    {1, 1, 1, 0},
    {1, 0, 0, 0},
    {1, 1, 1, 0},
    {0, 0, 1, 0},
```

```
    {1, 1, 1, 0}
  },
  {//6
    {1, 1, 1, 0},
    {1, 0, 1, 0},
    {1, 1, 1, 0},
    {0, 0, 1, 0},
    {1, 1, 1, 0}
  },
  {//7
    {0, 0, 1, },
    {0, 0, 1, 0},
    {0, 1, 0, 0},
    {1, 0, 0, 0},
    {1, 1, 1, 0}
  },
  {//8
    {1, 1, 1, 0},
    {1, 0, 1, 0},
    {1, 1, 1, 0},
    {1, 0, 1, 0},
    {1, 1, 1, 0}
  },
  {//9
    {1, 1, 1, 0},
    {1, 0, 0, 0},
    {1, 1, 1, 0},
    {1, 0, 1, 0},
    {1, 1, 1, 0}
  }
};

//rainbow animation to transition between game states
void rainbowAnimation(uint8_t wait) {
  // Turn on LEDs sequentially in a rainbow of hues
  for (uint16_t i = 0; i < NUM_LEDS; i++) {
```

```
    leds[i] = CHSV((i * 256 / NUM_LEDS) % 256, 255, 255);
    FastLED.show();
    FastLED.delay(wait);
  }

  // Wait a bit before turning off
  FastLED.delay(wait * 5);

  // Turn off LEDs sequentially
  for (uint16_t i = 0; i < NUM_LEDS; i++) {
    leds[i] = CHSV(0, 0, 0); // CHSV(0, 0, 0) is off/black
    FastLED.show();
    FastLED.delay(wait);
  }
}

//setup initiates game, starts borders, generates pieces, and
allows for input
void setup() {
  FastLED.addLeds<LED_TYPE, LED_PIN, COLOR_ORDER>(leds,
NUM_LEDS).setCorrection(TypicalLEDStrip);
  FastLED.setBrightness(BRIGHTNESS);
  FastLED.clear();
  rainbowAnimation(1);
  clearBoard();
  Serial.begin(115200);
  setupBorders();
  randomSeed(analogRead(0));
  spawnNewTetrimino();
}

//main game loop that moves and locks pieces, checks for game
over, and gets user input
void loop() {
  unsigned long currentTime = millis();
  // Handle falling pieces
```

```
if (currentTime - fallTimer > fallInterval) {
  clearTetrimino();
  posY++;
  if (checkCollision(posX, posY, currentTetrimino.shape)) {
    posY--;
    lockTetrimino();
    clearAndDropRows();
    spawnNewTetrimino();
    // Check for game over and reset the game
    if (isGameOver()) {
      tone(speakerPin, GOF1, 500);
      delay(500);
      tone(speakerPin, GOF2, 1000);
      delay(500);
      tone(speakerPin, GOF3, 1000);
      delay(500);
      clearBoard();
      displayScore(score); // Display the score
      delay(5000);
      // Reset game state
      score = 0; // Reset the score
      rainbowAnimation(1);
      setupBorders();
      clearBoardArray();  // Clear the board array
      randomSeed(analogRead(0));
      spawnNewTetrimino();
    }
    Serial.print("Score: ");
    Serial.println(score);
    displayScoreInBinary();
  }
  drawTetrimino();
  FastLED.show();
  fallTimer = currentTime; //update fall timer
}
// Process input without waiting for the falling interval to
```

```
complete
  processInput();
}

//method for getting all user input from serial monitor
void processInput() {
  if (Serial.available() > 0) {
    char input = Serial.read();
    clearTetrimino();
    bool soundPlayed = false;

    if (input == 'd') { // Move right
      posX--;
      if (checkCollision(posX, posY, currentTetrimino.shape)) {
        posX++;
      } else {
        soundPlayed = true;
      }
    } else if (input == 'a') { // Move left
      posX++;
      if (checkCollision(posX, posY, currentTetrimino.shape)) {
        posX--;
      } else {
        soundPlayed = true;
      }
    } else if (input == 'w') { // Slow drop
      posY++;
      posY++;
      if (checkCollision(posX, posY, currentTetrimino.shape)) {
        posY--;
      }
    } else if (input == 's') { // Fast drop
      score = score + 10;
      while (!checkCollision(posX, posY + 1,
currentTetrimino.shape)) {
        posY++;
```

```
      }
    } else if (input == 'q') { // Rotate counter-clockwise
      rotateTetrimino(true);
      if (checkCollision(posX, posY, currentTetrimino.shape)) {
        rotateTetrimino(false); // Revert the rotation if it
causes a collision
      } else {
        soundPlayed = true;
      }
    } else if (input == 'e') { // Rotate clockwise
      rotateTetrimino(false);
      if (checkCollision(posX, posY, currentTetrimino.shape)) {
        rotateTetrimino(true); // Revert the rotation if it
causes a collision
      } else {
        soundPlayed = true;
      }
    }

    if (soundPlayed) {
      tone(speakerPin, blipFrequency, 100); // Play the blip
sound for 50 milliseconds
    }
    drawTetrimino();
    FastLED.show();
  }
}

//displays the next piece in the top left corner before the
current piece is placed
void displayNextTetrimino() {
  for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 4; y++) {
      int displayX = x + 11;
      int displayY = y + 2;
      if (nextTetrimino.shape[y][x]) {
```

```
        setMatrixLEDColor(displayX, displayY,
nextTetrimino.color);
      } else {
        setMatrixLEDColor(displayX, displayY, CRGB::Black);
      }
    }
  }
}

//the method is used to change the colors of individual LEDs, it
is called whenever color changes happen
void setMatrixLEDColor(int x, int y, CRGB color) {
  int ledIndex;
  if (y % 2 == 0) {
    ledIndex = y * MATRIX_WIDTH + x;
  } else {
    ledIndex = (y + 1) * MATRIX_WIDTH - x - 1;
  }
  leds[ledIndex] = color;
}

//initiallizes borders around the score and next piece area,
shows the playing space
void setupBorders() {
  for (int i = 0; i < MATRIX_HEIGHT; i++) {
    setMatrixLEDColor(10, i, CRGB::White);
    setMatrixLEDColor(15, i, CRGB::White);
  }
    // Fill in the top 4 LEDs
  for (int i = 11; i <= 14; i++) {
    setMatrixLEDColor(i, MATRIX_HEIGHT - 1, CRGB::White);
  }
  // Fill in the bottom 4 LEDs
  for (int i = 11; i <= 14; i++) {
    setMatrixLEDColor(i, 0, CRGB::White);
  }
```

```
  for (int i = 11; i <= 14; i++) {
    setMatrixLEDColor(i, 7, CRGB::White);
  }
  for (int i = 11; i <= 14; i++) {
    setMatrixLEDColor(i, 8, CRGB::White);
  }
}

//places the current tetrimino in the starting place to fall
void spawnNewTetrimino() {
  posY = 0; // Start with the Tetrimino partially off the top of
the screen
  posX = 3;
  if (nextTetrimino.color.r == 0 && nextTetrimino.color.g == 0
&& nextTetrimino.color.b == 0) { // If it's the first Tetrimino
    int randomTetrimino = random(0, 7); //randomizes the next
incoming tetrimino
    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        currentTetrimino.shape[i][j] =
TETRIMINOS[randomTetrimino][i][j];
      }
    }
    currentTetrimino.color = TETRIMINO_COLORS[randomTetrimino];
  } else {
    currentTetrimino = nextTetrimino;
  }
  // Generate next Tetrimino
  int randomTetrimino = random(0, 7);
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < 4; j++) {
      nextTetrimino.shape[i][j] = TETRIMINOS[randomTetrimino][i]
[j];
    }
  }
  nextTetrimino.color = TETRIMINO_COLORS[randomTetrimino];
```

```
  displayNextTetrimino(); // Display the next Tetrimino
}

//checks for collisions between pieces and borders of the
playing space
bool checkCollision(int newX, int newY, bool shape[4][4]) {
  for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 4; y++) {
      if (shape[y][x]) {
        int boardX = newX + x;
        int boardY = newY + y;

        if (boardX < 0 || boardX >= MATRIX_WIDTH - 6 || boardY
>= MATRIX_HEIGHT || board[boardY][boardX]) {
          return true;
        }
      }
    }
  }
  return false;
}

//locks the piece in place whenever it runs into another piece
or the bottom row of the playing space
void lockTetrimino() {
  for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 4; y++) {
      if (currentTetrimino.shape[y][x]) {
        int boardX = posX + x;
        int boardY = posY + y;
        if (boardY >= 0) { // Make sure we don't access negative
indices
          board[boardY][boardX] = true;
          setMatrixLEDColor(boardX, boardY,
currentTetrimino.color);
          tone(speakerPin, lockFrequency, 100);
```

```
        }
      }
    }
  }
}

//used to draw the current pieces on the LED board using their
globally intialized structures
void drawTetrimino() {
  for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 4; y++) {
      if (currentTetrimino.shape[y][x]) {
        setMatrixLEDColor(posX + x, posY + y,
currentTetrimino.color);
      }
    }
  }
}

//clears the old state of the piece so that the new state can be
shown as it falls
void clearTetrimino() {
  for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 4; y++) {
      if (currentTetrimino.shape[y][x]) {
        setMatrixLEDColor(posX + x, posY + y, CRGB::Black);
      }
    }
  }
}

//rotates the tetrimino whenver the user inputs 'e' or 'q'
void rotateTetrimino(bool counterClockwise) {
  bool rotated[4][4];

  for (int y = 0; y < 4; y++) {
```

```
    for (int x = 0; x < 4; x++) {
      if (counterClockwise) {
        rotated[y][x] = currentTetrimino.shape[x][3 - y];
      } else {
        rotated[y][x] = currentTetrimino.shape[3 - x][y];
      }
    }
  }

  memcpy(currentTetrimino.shape, rotated, sizeof(rotated));//
copies contents of rotated array to determine the correct
orientation
}

//clears the rows that are full and calls the dropRow function
to drops the remaining
void clearAndDropRows() {
  bool rowFull;
  int rowsCleared = 0;

  for (int y = 0; y < MATRIX_HEIGHT; y++) {
    rowFull = true;
    for (int x = 0; x < MATRIX_WIDTH - 6; x++) {
      if (!board[y][x]) {
        rowFull = false;
        break;
      }
    }

    if (rowFull) {
      rowsCleared++;
      dropRow(y);
      y--; // Check the same row again after dropping
    }
  }
```

```
  if (rowsCleared > 0) {
    updateScore(rowsCleared);
  }
}

//drops the rows necessary after clearing a row
void dropRow(int row) {
  for (int y = row; y > 0; y--) {
    for (int x = 0; x < MATRIX_WIDTH - 6; x++) {
      board[y][x] = board[y - 1][x];
    }
  }

  // Clear the top row
  for (int x = 0; x < MATRIX_WIDTH - 6; x++) {
    board[0][x] = false;
  }

  // Redraw the board to reflect the dropped rows
  for (int y = 0; y <= row; y++) {
    for (int x = 0; x < MATRIX_WIDTH - 6; x++) {
      if (board[y][x]) {
        setMatrixLEDColor(x, y, currentTetrimino.color);
      } else {
        setMatrixLEDColor(x, y, CRGB::Black);
      }
    }
  }

  // Restore the borders
  setupBorders();
}

//checks to see if a row is full and returns a boolean value
based on results
bool isRowFull(int row) {
```

```
  for (int col = 0; col < MATRIX_WIDTH - 6; col++) {
    if (!board[row][col]) {
      return false;
    }
  }
  return true;
}

//sets all the LEDs of a specified row to black
void clearRow(int row) {
  for (int y = row; y > 0; y--) {
    for (int x = 1; x < MATRIX_WIDTH - 1; x++) {
      setMatrixLEDColor(x, y, CRGB::Black);    }
  }
  for (int x = 1; x < MATRIX_WIDTH - 1; x++) {
    leds[XY(x, 0)] = CRGB::Black;
  }
}

//whenever rows are cleared, this function adds the respective
score to the total
void updateScore(int linesCleared) {
  switch (linesCleared) {
    case 1:
      score += 100;
      tone(speakerPin, littleFrequency, 200); // Play the
'little' sound for 100 milliseconds
      break;
    case 2:
      score += 300;
      tone(speakerPin, thungFrequency, 400); // Play the 'thung'
sound for 100 milliseconds
      break;
    case 3:
      score += 500;
      tone(speakerPin, thungFrequency, 600); // Play the 'thung'
```

```
sound for 200 milliseconds
      break;
    case 4:
      score += 800;
      tone(speakerPin, thungFrequency, 500);
      delay(150);
      tone(speakerPin, thungFrequency, 500);
      break;
  }
  displayScoreInBinary();
}

//this function displays the updated value of the globally
assigned "score" variable in the bottom right corner behind the
border
void displayScoreInBinary() {
  int scoreCopy = score;
  for (int row = 14; row >= 11; row--) {
    for (int col = 14; col >= 11; col--) {
      if (scoreCopy % 2 == 1) {
        setMatrixLEDColor(col, row, CRGB::Green); // Set LED
color to Green if the corresponding bit is 1
      } else {
        setMatrixLEDColor(col, row, CRGB::Black); // Set LED
color to Black if the corresponding bit is 0
      }
      scoreCopy >>= 1;
    }
  }
}

//drops all rows above the cleared row, then updates LED matrix
to reflect new positions
void dropRowsAbove(int rowCleared) {
  for (int row = rowCleared; row > 0; row--) {
    for (int col = 0; col < MATRIX_WIDTH - 4; col++) {
```

```
      board[row][col] = board[row - 1][col];
      setMatrixLEDColor(col, row, leds[XY(col, row - 1) %
NUM_LEDS]);
    }
  }
}

//initiates all of the LEDs on the board as an X,Y value in a
16x16 matrix
int XY(int x, int y) {
  int ledIndex;
  if (y % 2 == 0) {
    ledIndex = y * MATRIX_WIDTH + x;
  } else {
    ledIndex = (y + 1) * MATRIX_WIDTH - x - 1;
  }
  return ledIndex;
}

//checks for game over condition, which is any piece that gets
locked in the top row of the board
bool isGameOver() {
  for (int x = 0; x < MATRIX_WIDTH - 4; x++) {
    if (board[0][x]) {
      return true;
    }
  }
  return false;
}

//clears the board array that represents the playable space
void clearBoardArray() {
  for (int y = 0; y < MATRIX_HEIGHT; y++) {
    for (int x = 0; x < MATRIX_WIDTH; x++) {
      board[y][x] = false;
    }
```

```
  }
}

//clears the entire board and changes all LED colors to black
void clearBoard() {
  for (uint16_t i = 0; i < NUM_LEDS; i++) {
    leds[i] = CRGB::Black;
  }
  FastLED.show();
}

//displays a digit in the structure declared at the top of the
program
void drawDigit(int digit, int topLeftX, int topLeftY) {
  for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 5; y++) {
      if (digits[digit][y][x]) {
        setMatrixLEDColor(topLeftX + x, topLeftY - y,
CRGB::Green); // change y coordinate to flip vertically
      }
    }
  }
}

//calls the method above to draw each digit of the total score
void displayScore(unsigned long score) {
  int digitsCount = 4;
  int positions[4][2] = {{0, 10}, {4, 10}, {8, 10}, {12,
10}}; // update y coordinates to match the new drawDigit
function
  int scoreArray[4] = {0};

  for (int i = 0; i < digitsCount; i++) {
    scoreArray[i] = score % 10;
    score /= 10;
  }
```

```
  for (int i = digitsCount - 1; i >= 0; i--) {
    drawDigit(scoreArray[i], positions[i][0], positions[i][1]);
  }

  FastLED.show();
  delay(2000); // Display the score for 2 seconds
}
```