/\*

\*/

const int WIDTH = 8;  //10

const int HEIGHT = 8;  //16

boolean Play = true;

int board[HEIGHT][WIDTH];

int score = 0;

int BLOCKS[][4][4] = {

  {  //Square  0

    { 0,0,0,0 },

    { 0,1,1,0 },

    { 0,1,1,0 },

    { 0,0,0,0 }

  },

  { //Tall  1

    { 0,0,0,0 },

    { 0,0,0,0 },

    { 1,1,1,1 },

    { 0,0,0,0 }

  },

  {  //WASD  2

    { 0,0,0,0 },

    { 0,1,0,0 },

    { 1,1,1,0 },

    { 0,0,0,0 }

  },

  {  //Z  3

    { 0,0,0,0 },

    { 1,1,0,0 },

    { 0,1,1,0 },

    { 0,0,0,0 }

  },

  { //S  4

    { 0,0,0,0 },

    { 0,1,1,0 },

    { 1,1,0,0 },

    { 0,0,0,0 }

  },

  { //J  5

    { 0,0,0,0 },

    { 1,0,0,0 },

    { 1,1,1,0 },

    { 0,0,0,0 }

  },

  { //L  6

    { 0,0,0,0 },  //0000000000

    { 0,0,1,0 },

    { 1,1,1,0 },

    { 0,0,0,0 }

  }

};

//Upper left corner of the current block, represented by a 4x4 space like above

int currentX;

int currentY;

boolean hit;

int gameSpeed = 650; //1 second delay between drops

//----LED MATRIX SETUP----

const int row[8] = {

  9,4,13,6,17,12,16,10

};

const int col[8] = {

  5,15,14,8,11,7,3,2

};

void displayMatrix() {

  int prevX = -1;

  int prevY = -1; /\*

  for (int i=0; i < HEIGHT; i++) {

    for (int j=0; j < WIDTH; j++) {

      digitalWrite(col[j], HIGH);

      digitalWrite(row[i], LOW);

    }

  }  \*/

  digitalWrite(row[HEIGHT-1], LOW);

  for (int i=0; i < HEIGHT; i++) {

    for (int j=0; j < WIDTH; j++) {

      if (board[i][j] != 0) {

        digitalWrite(col[j], LOW);

        digitalWrite(row[i], HIGH);

        //Turn Previous off

        digitalWrite(col[prevX], HIGH);

        digitalWrite(row[prevY], LOW);

        //Store Current

        prevX = j;

        prevY = i;

      }

      digitalWrite(col[prevX], HIGH);

      digitalWrite(row[prevY], LOW);

      //Fixes odd anomaly bugs

      digitalWrite(row[HEIGHT-1], LOW);

    }

  }

}

//--------TETRIS--------

void resetBoard(int num) {

  //board[HEIGHT][WIDTH];

  for (int i=0; i < HEIGHT; i++) {

    for (int j=0; j < WIDTH; j++) {

      board[i][j] = num;

      score = 0;

    }

  }

}

void gameOver() {

  Serial.println("GAME OVER"); //debug purposes

  Play = false;

  //Some cool animation shizzle right hizzle

  resetBoard(7);

  for (int j=0; j < WIDTH; j++) {

    for (int i=0; i <= 50; i++) {

      delay(1);

      displayMatrix();

      if (i == 25) { // make this dependant on score for difficulty increase

        for (int i=0; i < HEIGHT; i=i+2) {

          board[i][j] = 0;

          board[i+1][WIDTH-j-1] = 0;

        }

      }

    }

  }

  resetBoard(0);

  delay(2000);

  Play = true;

  newBlock();

}

void newBlock() {

  int block[4][4];                     //Make new block based on one of templates

  int type = random(0,7);

  Serial.print("Type: ");

  Serial.println(type);

  for (int i=0; i < 4; i++) {

    for (int j=0; j < 4; j++) {

      block[i][j] = BLOCKS[type][i][j];

    }

  }

  int shift = 1;

  if (type == 1) { shift = 0; }

  boolean open = true;                 //Place block in map if possible, otherwise end game

  int CENTER = 3; //ensures block starts in middle

  for (int i=0; i < 4; i++) {

    for (int j=0; j < 4; j++) {

      if ((board[i][CENTER+j] == 3) && (block[i][j] == 1)) {

        open = false;

      }

      else {

        if (block[i][j] == 1) {

          board[i-shift][CENTER+j] = 1;

        }

        currentX = CENTER;

        currentY = 0; //y-origin is at top

      }

    }

    if (open == false) {

      gameOver();

    }

  }

}

void printBoard() {

  for (int i=0; i < HEIGHT; i++) {

    for (int j=0; j < WIDTH; j++) {

      if (board[i][j] == 0) { Serial.print("."); }

      else { Serial.print(board[i][j]); }

    }

    Serial.println();

  }

  Serial.println("\n");

}

void rotate(int cw) { //1 for clockwise, -1 for counter-clockwise

  int block[4][4];

  int newX;

  int newY;

  for (int i=0; i < 4; i++) {   //defaults temp block to 0s

    for (int j=0; j < 4; j++) {

      block[i][j] = 0;

    }

  }

  //int iSkip;

  //int jSkip;

 for (int i=0; i < 4; i++) {

   for (int j=0; j < 4; j++) {

     if (board[currentY+i][currentX+j] == 1) {

        Serial.print("x: ");

        Serial.print(j);

        Serial.print(", y: ");

        Serial.println(i);

        if (cw == -1) {

          block[3-j][i] = 1;

        }

        else {

          block[j][3-i] = 1;

        }

     }

   }

 }

  boolean topSpace = true;                        //Detect offcentered-ness in 4x4

  boolean leftSpace = true;

  boolean midRightSpace = true;

  boolean rightSpace = true;

  boolean bottomSpace = true;

  for (int i=0; i < 4; i++) {

    if (block[i][0] == 1) { leftSpace = false; }

    if (block[i][3] == 1) { rightSpace = false; }

    if (block[i][2] == 1) { midRightSpace = false; }

    if (block[0][i] == 1) { topSpace = false; }

    if (block[3][i] == 1) { bottomSpace = false; }

  }

  //Debug

  for (int i=0; i < 4; i++) {

    Serial.println();

    for (int j=0; j < 4; j++) {

      Serial.print(block[i][j]);

    }

  }

  Serial.println("\n");

  //Debug

  if (leftSpace && !rightSpace) {     //if 3 spaces wide prioritize left, otherwise leave alone

    Serial.println("left prioritization");

    for (int i=0; i < 4; i++) {

      for (int j=1; j <= 3; j++) {

        block[i][j-1] = block[i][j];

        block[i][j] = 0;

      }

    }

  }

    if (!leftSpace && midRightSpace) {

    Serial.println("right prioritization");

    for (int i=0; i < 4; i++) {

      for (int j=2; j >= 0; j--) {

        block[i][j+1] = block[i][j];

        block[i][j] = 0;

      }

    }

  }

  if (bottomSpace && !topSpace) {                    //bring to top

    for (int i=2; i >= 0; i--) {

      for (int j=0; j < 4; j++) {

        block[i+1][j] = block[i][j];

        block[i][j] = 0;

      }

    }

  }

  Serial.print("topSpace = ");

  Serial.println(topSpace);

  Serial.print("rightSpace = ");

  Serial.println(rightSpace);

  Serial.print("leftSpace = ");

  Serial.println(leftSpace);

  //Debug

  for (int i=0; i < 4; i++) {

    Serial.println();

    for (int j=0; j < 4; j++) {

      Serial.print(block[i][j]);

    }

  }

  Serial.println("\n");

  //Debug

  boolean canRotate = true;

  for (int i=0; i < 4; i++) {      //Place rotated block

    for (int j=0; j < 4; j++) {

      if ((block[i][j] == 1) && (board[currentY+i][currentX+j] == 3)) {

        canRotate = false;

      }

    }

  }

  if (canRotate) {

    for (int i=0; i < 4; i++) {                      //Erase un-rotated block off of current 4x4

      for (int j=0; j < 4; j++) {

        if (board[currentY+i][currentX+j] == 1) {

          board[currentY+i][currentX+j] = 0;

        }

      }

    }

    for (int i=0; i < 4; i++) {      //Place rotated block

      for (int j=0; j < 4; j++) {

        if (block[i][j] == 1) {

          board[currentY+i][currentX+j] = 1;

        }

      }

    }

  }

}

void shift(String dir) {  //LEFT AND RIGHT MOVEMENTS

  boolean canMove = true;

  if (dir == "left") {

    for (int i=0; i < 4; i++) {

      for (int j=0; j < 4; j++) {

        if (board[currentY+i][currentX+j] == 1) {

          if (!(currentX+j-1 >= 0) || (board[currentY+i][currentX+j-1] == 3)) {

            canMove = false;

          }

        }

      }

    }

    if (canMove) {

      for (int i=0; i < 4; i++) {

        for (int j=0; j < 4; j++) {

          if (board[currentY+i][currentX+j] == 1) {

            board[currentY+i][currentX+j-2] = 1; //2

            board[currentY+i][currentX+j] = 0;

          }

        }

      }

      currentX = currentX-2;

    }

  }

  if (dir == "right") {

    for (int i=0; i < 4; i++) {

      for (int j=0; j < 4; j++) {

        if (board[currentY+i][currentX+j] == 1) {

          if (!(currentX+j+1 < WIDTH) || (board[currentY+i][currentX+j+1] == 3)) {

            canMove = false;

          }

        }

      }

    }

  }

  if (canMove) {

    for (int i=0; i < 4; i++) {

      for (int j=3; j >= 0; j--) {

        if (board[currentY+i][currentX+j] == 1) {

          board[currentY+i][currentX+j+1] = 1;

          board[currentY+i][currentX+j] = 0;

        }

      }

    }

    currentX++;

  }

}

void controls() { //RECEIVES INPUT

  int incomingByte;

  if (Serial.available() > 0) {

    // read the incoming byte:

    incomingByte = Serial.read();

    Serial.print("received: ");  //For finding ascii character values

    Serial.println(incomingByte, DEC);

    if (Play) {

      if (incomingByte == 122) { rotate(-1); } //Z key  Counter-clockwise

      if (incomingByte == 120) { rotate(1); }  //X key  Clockwise

      if (incomingByte == 97) { shift("left"); }

      if (incomingByte == 100) { shift("right"); }

      if (incomingByte == 115) { //S key

        while (!hit) { drop(); } //HARD DROP

      }

    }

    if (incomingByte == 113) {  //Q key  RESET

      resetBoard(0);

      Play = true;

      newBlock();

    }

  }

}

void drop() {

  hit = false;

  for (int i=0; i < 4; i++) {                //                      |

    for (int j=0; j < 4; j++) {              //space directly below \|/

      if ( ((board[currentY+i][currentX+j] == 1) && (board[currentY+i+1][currentX+j] == 3)) || ((board[currentY+i][currentX+j] == 1) && (currentY+i == HEIGHT-1)) ) {   //   Checks to see if every chunk from 4x4 is active, if so,

        hit = true;                                                                           // looks to see if chunk below is already occupied. If every

      }                                                                                       // chunk is unoccupied, the block is dropped. Otherwise,

    }                                                                                         // a new block is spawned.

  }

  if (hit == false) { //if not hit anything below, drop

    for (int i=3; i >= 0; i--) {

      for (int j=3; j >= 0; j--) {

       if (board[currentY+i][currentX+j] == 1) {            //Probably bad form but whatever...

          board[currentY+i][currentX+j] = 0;     //Shifts matrix down

          board[currentY+i+1][currentX+j] = 2;// Changes block to temp state (2) for shift

        }

      }

    }

    for (int i=0; i <= 4; i++) {

      for (int j=0; j < 4; j++) {

       if (board[currentY+i][currentX+j] == 2) {

          board[currentY+i][currentX+j] = 1;     //Reverts back to allow for movement again

        }

      }

    }

    currentY++; //Updates reference

  }

  else {

    for (int i=0; i < 4; i++) {

      for (int j=0; j < 4; j++) {

       if (board[currentY+i][currentX+j] == 1) {

          board[currentY+i][currentX+j] = 3;     //Makes blocks permanantly stuck

        }

      }

    }

    newBlock();

  }

  //Serial.print("Hit: ");

  //Serial.println(hit);

}

void lineCheck() {

  boolean full;

  for (int I=0; I < HEIGHT; I++) {

    full = true;

    for (int j=0; j < WIDTH; j++) {

      if (!(board[I][j] == 3)) {

        full = false;

      }

    }

    if (full) {

      for (int j=0; j < WIDTH; j++) {

        board[I][j] = 0;

      }

      for (int i=I-1; i >= 0; i--) {

        for (int j=0; j < WIDTH; j++) {

          if (board[i][j] == 3) {

            board[i+1][j] = 3;

            board[i][j] = 0;

          }

        }

      }

      score++;

    }

  }

}

// ARDUINO INITIALIZATION //

void setup() {

  for (int thisPin = 0; thisPin < 8; thisPin++) {

    // initialize the output pins:

    pinMode(col[thisPin], OUTPUT);

    pinMode(row[thisPin], OUTPUT);

    // take the col pins (i.e. the cathodes) high to ensure that

    // the LEDS are off:

    digitalWrite(col[thisPin], HIGH);

  }

  resetBoard(0);

  Serial.begin(9600);

  Serial.println("\n\nTetris Debug\n");

  newBlock();

  printBoard();

}

// ARDUINO UPDATE //

void loop() {

  if (Play) {

    for (int i=0; i <= gameSpeed; i++) {

      lineCheck();

      controls();

      delay(1); // allows for controls to be used every millisecond, in theory

      displayMatrix();

      if (i == gameSpeed) { // make this dependant on score for difficulty increase

        drop();

        delay(1);

        printBoard();

        Serial.print("\nScore: ");

        Serial.println(score);

      }

    }

  }

  else { controls(); }

}