```
/*
 * File:        Maze Generator
 * Author:      Jack Brzozowski, Dilan Lakhani, Kyle Infantino
 * Adapted from code by Christian Hill, https://scipython.com/blog/making-a-maze/
 * Target PIC:  PIC32MX250F128B
 */

////////////////////////////////////
// clock AND protoThreads configure!
// You MUST check this file!
#include "config_1_3_2.h"
// threading library
#include "pt_cornell_1_3_2_python.h"

#include "tft_gfx.h"
#include "tft_master.h"
////////////////////////////////////
// need for rand function
#include <stdlib.h>
#include <stdfix.h>
#include <math.h>
#include <stdint.h>
////////////////////////////////////


// === thread structures ============================================
// thread control structs
// note that UART input and output are threads

// system 1 second interval tick
int sys_time_seconds ;

int dimensionx; // dimensions of maze in # of nodes
int dimensiony; //
int start;
int end;

short xleft;    //coordinates of tft border
short xright;
short ybottom;
short ytop;
int xlen;
int ylen;
short radius;

#define MAX_DIM_X 24 //maximum number of nodes in x and y
#define MAX_DIM_Y 18
#define timer_match 25000

char receive_string[64];

unsigned int slider_value;
unsigned int button_value;
unsigned int toggle_value;
char keypush_value;
char keypush_down;
char move1;
char move2;
char slider_id;
char button_id;
char toggle_id;
char keypush_id;
char new_slider;
char new_string;
char new_button;
char new_toggle;
char new_keypush;
```

```
short difficulty;
short mode;
short time_trial_en;
short game_overFlag;
short game_over;

short user1_x; // x position of user 1
short user1_y;
short user2_x;
short user2_y;

short winner; // 1 if winner is player 1, 2 if it is player 2

volatile int counter; // checks how many ADC samples have occured

void __ISR(_TIMER_2_VECTOR, ipl2) Timer2Handler(void)
{
   // clear the interrupt flag
   mT2ClearIntFlag();
   counter++;
}

//==================== Generate Maze and Write to TFT
 ========================================================================
typedef struct MazeNode{ // each cell in the maze
   //coordinates
   int8_t x;
   int8_t y;
   // walls
   int8_t north;
   int8_t south;
   int8_t east;
   int8_t west;
   // backpointer directions: North (0,1); South (0,-1); East(1,0); West(-1,0)
   // To get to the backptr node, add the backptr to the current x and y
   int8_t backptr_x;
   int8_t backptr_y;

   int8_t inFrontier; // 1 if cell has been added to frontier set
} node_t;

node_t nodes [MAX_DIM_X][MAX_DIM_Y];

int hasAllWalls(node_t node){ //1 if node has all walls, 0 else
   return node.north & node.south & node.east & node.west;
}

void knockDownWall(node_t* node1, node_t* node2){ // node 1 is selected node, node 2 is backptr node
   // declare local vars for TFT wall knockdown
   static short x;
   static short y;
   static short x0;
   static short x1;
   static short y0;
   static short y1;
   if (node1->x > node2->x){ //vertical wall
      node1->west = 0;
      node2->east = 0;
      x = (node1->x)*xlen; //get coordinates
      y0 = (node1->y)*ylen;
      y1 = ((node1->y)+1)*ylen;
      tft_drawLine(x,y0,x,y1,ILI9340_BLACK); //erase wall
   } else if (node1->x < node2->x){
      node1->east = 0;
      node2->west = 0;
      x = (node2->x)*xlen;
```

```
        y0 = (node2->y)*ylen;
        y1 = ((node2->y)+1)*ylen;
        tft_drawLine(x,y0,x,y1,ILI9340_BLACK); //erase wall
    } else if (node1->y > node2->y){ //horizontal wall
        node1->south = 0;
        node2->north = 0;
        x0 = (node1->x)*xlen;
        x1 = ((node1->x)+1)*xlen;
        y = (node1->y)*ylen;
        tft_drawLine(x0,y,x1,y,ILI9340_BLACK); //erase wall
    } else{
        node1->north = 0;
        node2->south = 0;
        x0 = (node2->x)*xlen;
        x1 = ((node2->x)+1)*xlen;
        y = (node2->y)*ylen;
        tft_drawLine(x0,y,x1,y,ILI9340_BLACK); //erase wall
    }
}

node_t* neighbors[4]; // Position relative to current node: 0 is North, 1 is S, 2 is E, 3 is W

void findValidNeighbors(node_t node){
    //sets global neighbors array to neighbor nodes if node has all walls and not already in frontier
    if (node.x != 0){ //if not leftmost node
        //West Neighbor
        if (hasAllWalls(nodes[node.x-1][node.y]) && (!(nodes[node.x-1][node.y].inFrontier))){ // if neightbor still has all its walls
            neighbors[3] = &nodes[node.x-1][node.y];
            // set backptr to East
            neighbors[3]->backptr_x = 1;
            neighbors[3]->backptr_y = 0;
            neighbors[3]->inFrontier = 1; // mark that it is in frontier set
        }
        else {
            neighbors[3] = NULL; //not a valid neighbor
        }
    } else {
        neighbors[3] = NULL;
    }
    if (node.x != (dimensionx-1)){
        // East Neighbor
        if (hasAllWalls(nodes[node.x+1][node.y]) && (!nodes[node.x+1][node.y].inFrontier)){
            neighbors[2] = &nodes[node.x+1][node.y];
            // set backptr to West
            neighbors[2]->backptr_x = -1;
            neighbors[2]->backptr_y = 0;
            neighbors[2]->inFrontier = 1; // mark that it is in frontier set
        }
        else {
            neighbors[2] = NULL;
        }
    } else {
        neighbors[2] = NULL;
    }
    if (node.y != 0){
        // South Neighbor
        if (hasAllWalls(nodes[node.x][node.y-1]) && (!nodes[node.x][node.y-1].inFrontier)){
            neighbors[1] = &nodes[node.x][node.y-1];
            // set backptr to North
            neighbors[1]->backptr_x = 0;
            neighbors[1]->backptr_y = 1;
            neighbors[1]->inFrontier = 1; // mark that it is in frontier set
        }
        else {
            neighbors[1] = NULL;
        }
```

```
    } else {
        neighbors[1] = NULL;
    }
    if (node.y != (dimensiony-1)){
        // North Neighbor
        if (hasAllWalls(nodes[node.x][node.y+1]) && (!nodes[node.x][node.y+1].inFrontier)){
            neighbors[0] = &nodes[node.x][node.y+1];
            // set backptr to South
            neighbors[0]->backptr_x = 0;
            neighbors[0]->backptr_y = -1;
            neighbors[0]->inFrontier = 1; // mark that it is in frontier set
        }
        else {
            neighbors[0] = NULL;
        }
    } else {
        neighbors[0] = NULL;
    }
}

void generateMaze(){
    //draw maze
    //Initialize 2D-array
    static int i;
    static int j;
    for (i = 0; i < dimensionx; i++){
        for (j = 0; j < dimensiony; j++){
            //Dont re-declare nodes, just reference them
            nodes[i][j].x = i;
            nodes[i][j].y = j;
            // initially set all walls to True
            nodes[i][j].north = 1;
            nodes[i][j].south = 1;
            nodes[i][j].east = 1;
            nodes[i][j].west = 1;
            // set backpointers to NULL initially
            nodes[i][j].backptr_x = NULL;
            nodes[i][j].backptr_y = NULL;
            // initially not in frontier set
            nodes[i][j].inFrontier = 0;
        }
    }
    node_t* frontier[(dimensionx)*(dimensiony)];
    //Obtain a random starting cell
    static node_t* currentCell;
    currentCell = &nodes[(rand() % dimensionx)][(rand() % dimensiony)];
    currentCell->inFrontier = 1;
    static node_t* nextCell;
    static int index;
    start = 0;
    end = 0;
    do {
        findValidNeighbors(*currentCell); //Update Neighbors array
        for(i=0; i<4; i++){
            // Add nodes to frontier
            if (neighbors[i] != NULL){
                //Enqueue
                frontier[end] = (neighbors[i]);
                end++;
            }
        }
        index = (rand() % (end - start)) + start; //Random num from start to end-1 (end is first free position)
        nextCell = frontier[index]; //Get a random cell from the frontier
        //Knock Down Wall between next cell and its backptr
        knockDownWall(nextCell,&nodes[nextCell->x+nextCell->backptr_x][nextCell->y+nextCell->backptr_y]);
         //Dequeue
```

```
        currentCell = nextCell;
        frontier[index] = frontier[start];
        start++;

    } while(start != end); // while frontier is not empty

}

void printMaze(){ // test our maze algorithm by printing to Python GUI
    //Print a line at the top
    int i;
    for(i=0; i<(dimensionx*3+1); i++){
        printf("-");
    }
    printf("\r");
    // Generate the maze line by line
    int j;
    for(j=0; j<dimensiony; j++){
        printf("|");
        for(i=0; i<dimensionx; i++){
            if (nodes[i][j].east){
                printf("  |");
            }
            else{
                printf("   ");
            }
        }
        printf("\r");
        printf("+");
        for(i=0; i<dimensionx; i++){
            if(nodes[i][j].north){
                printf("--+");
            }
            else{
                printf("  +");
            }
        }
        printf("\r");
    }
}

void drawGrid(){
    // draw initial grid of walls

    //TFT 320x240

    //Clear TFT
    tft_fillScreen(ILI9340_BLACK);

    static int dimx;
    static int dimy;
    for (dimx = 0; dimx <= dimensionx; dimx++){ //draw vertical lines along x axis
        tft_drawLine((short)(xlen*dimx),ybottom,(short)(xlen*dimx),ytop,ILI9340_WHITE);
    }
    for (dimy = 0; dimy <= dimensiony; dimy++){ //draw horizontal lines along y axis
        tft_drawLine(xleft,(short)(dimy*ylen),xright,(short)(dimy*ylen),ILI9340_WHITE);
    }
    //Delete entrance and exit to maze
    tft_drawLine(xleft, ybottom, xleft, ylen, ILI9340_BLACK);
    tft_drawLine(xleft+(dimensionx*xlen), ybottom+((dimensiony-1)*ylen), xleft+(dimensionx*xlen), ybottom+dimensiony*ylen,
ILI9340_BLACK);

    //Fill cell for entrance red and exit green
    tft_fillRect(xleft+2, ybottom+2, xlen-4, ylen-4,ILI9340_RED);
    tft_fillRect((xleft+((dimensionx-1)*xlen)+2), (ybottom+((dimensiony-1)*ylen)+2), xlen-4, ylen-4, ILI9340_GREEN);
    //Fill in edges if dimensions don't divide evenly into TFT pixel dimensions
```

```
    tft_fillRect(xleft, ytop-((ytop-ybottom) % dimensiony)+1 , (xright-xleft), (ytop-ybottom) % dimensiony, ILI9340_BLACK);
    tft_fillRect(xright-((xright-xleft)%dimensionx)+1, ybottom, (xright-xleft) % dimensionx, (ytop-ybottom), ILI9340_BLACK);
}

//returns the x coordinate of the center of this cell on the tft
#define getCenterX(x) ((x)*xlen+xlen/2)
//returns the y coordinate of the center of this cell on the tft
#define getCenterY(y) ((y)*ylen+ylen/2)



int8_t canMove (short userX, short userY, char direction){
    node_t node = nodes[userX/xlen][userY/ylen];

    // check if the node the player is in has walls in the direction trying to move
    if (direction == 'd'){ //flipped for TFT vs perspective
        return !node.north || userY < getCenterY(userY/ylen);
    } else if (direction == 'u'){ //South wall is on the top side of the node
        return !node.south || userY > getCenterY(userY/ylen);
    } else if (direction == 'r'){
        return !node.east || userX < getCenterX(userX/xlen);
    } else if (direction == 'l'){
        return !node.west || userX > getCenterX(userX/xlen);
    } else {
        return 0;
    }
}

void drawPlayer(){
    if (difficulty == 0){
        radius = 5; // Biggest user icon for easy
    }
    if (difficulty == 1){
        radius = 4; // Smaller user icon for medium
    }
    if (difficulty == 2){
        radius = 3;
    }
    user1_x = xlen/2;
    user1_y = ylen/2;
    user2_x = xlen/2;
    user2_y = ylen/2;

    if (mode == 1){
        tft_fillCircle(user1_x, user1_y, radius, ILI9340_WHITE);
    } else {
        tft_fillCircle(user1_x, user1_y, radius, ILI9340_WHITE);
        tft_fillCircle(user2_x, user2_y, radius, ILI9340_BLUE);
    }

}


// === new game thread =====================================================
static PT_THREAD (protothread_newgame(struct pt *pt))
{
    PT_BEGIN(pt);
    while(1){
        // wait for a new string from Python
        PT_YIELD_UNTIL(pt, new_button==1);
        new_button = 0;
        game_over = 0;
        // hard: 24x18
        // med:  16x12
        // easy: 12x9
        if (difficulty == 0){ //easy
```

```
        dimensionx = 12;
        dimensiony = 9;
    }
    else if (difficulty == 1 ){
        dimensionx = 16;
        dimensiony = 12;
    }
    else { //difficulty == 2
        dimensionx = 24;
        dimensiony = 18;
    }

    xlen = (xright - xleft) / dimensionx;
    ylen = (ytop - ybottom) / dimensiony;


    // === Draw a dimension x dimension grid on the tft
    drawGrid();

    srand(PT_GET_TIME());
    // // === Create a maze that is dimension x dimension in size ==================
    generateMaze();

    drawPlayer();

    if (time_trial_en){
        //time trial enabled
        counter = 0;
        OpenTimer2(T2_ON | T2_SOURCE_INT | T2_PS_1_16, timer_match);
        ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_2);
        mT2ClearIntFlag(); // and clear the interrupt flag
    }

  } // END WHILE(1)
  PT_END(pt);
} // thread newgame

// === player thread =========================================================
static PT_THREAD (protothread_player(struct pt *pt))
{
  PT_BEGIN(pt);

  //starting positions
  user1_x = getCenterX(0);
  user1_y = getCenterY(0);
  user2_x = getCenterX(0);
  user2_y = getCenterY(0);

  while(1){
    // wait for a new keypush from Python
    PT_YIELD_TIME_msec(10);

    if (canMove(user1_x, user1_y, move1)) {
      tft_fillCircle(user1_x, user1_y, radius, ILI9340_BLACK);
      if (move1 == 'u'){ //Add conditions here to determine the legality of this motion
        user1_x = getCenterX(user1_x/xlen);
        user1_y--; //looks up on TFT
      } else if (move1 == 'd'){
        user1_x = getCenterX(user1_x/xlen);
        user1_y++; //looks down on TFT
      } else if (move1 == 'l'){
        user1_y = getCenterY(user1_y/ylen);
        user1_x--;
      } else if (move1 == 'r'){
        user1_y = getCenterY(user1_y/ylen);
        user1_x++;
```

```
            }
        }
        tft_fillCircle(user1_x, user1_y, radius, ILI9340_WHITE);
        if ( mode == 2 ){
            if (canMove(user2_x, user2_y, move2)) {
                tft_fillCircle(user2_x, user2_y, radius, ILI9340_BLACK);
                if (move2 == 'u'){
                    user2_x = getCenterX(user2_x/xlen);
                    user2_y--; //looks up on TFT
                } else if (move2 == 'd'){
                    user2_x = getCenterX(user2_x/xlen);
                    user2_y++; //looks down on TFT
                } else if (move2 == 'l'){
                    user2_y = getCenterY(user2_y/ylen);
                    user2_x--;
                } else if (move2 == 'r'){
                    user2_y = getCenterY(user2_y/ylen);
                    user2_x++;
                }
            }
            tft_fillCircle(user2_x, user2_y, radius, ILI9340_BLUE);
            if (((user2_x/xlen) == (dimensionx-1)) && ((user2_y/ylen) == (dimensiony-1))){
                winner = 2;
                game_overFlag = 1;
            }
        }
        if ( ((user1_x/xlen) == (dimensionx-1)) && ((user1_y/ylen) == (dimensiony-1) )){
            winner = 1;
            game_overFlag = 1;
        }

    } // END WHILE(1)
    PT_END(pt);
} // thread player

// === end game thread =======================================================
static PT_THREAD (protothread_endgame(struct pt *pt))
{
    PT_BEGIN(pt);
    static char buffer[60];
    static int sec;
    static int hundredth;
    while(1){
        // wait for a new string from Python
        PT_YIELD_UNTIL(pt, game_overFlag==1);
        game_overFlag = 0; //Clear Flag
        CloseTimer2();
        if (!game_over){
            game_over = 1;
            tft_setCursor(60, 30);
            tft_setTextColor(ILI9340_WHITE);  tft_setTextSize(4);
            if (time_trial_en){
                sec = counter/100;
                hundredth = counter % 100;
                sprintf(buffer, "Player %d\n\n    WINS!\n\n Time: %d.%ds", winner, sec, hundredth);
                printf("%d %d.%d", difficulty, sec, hundredth);
            } else {
                sprintf(buffer, "Player %d\n\n    WINS!\n", winner);
            }
            tft_writeString(buffer);
        }
    } // END WHILE(1)
    PT_END(pt);
} // thread endgame

// === keypush thread ========================================================
```

```
static PT_THREAD (protothread_keypush(struct pt *pt))
{
   PT_BEGIN(pt);

   while(1){
      // wait for a new keypush from Python
      PT_YIELD_UNTIL(pt, new_keypush==1);
      new_keypush = 0; //clear flag
      if (keypush_down == 'p'){
         if (keypush_id == 1){
            move1 = keypush_value;
         }
         if (keypush_id == 2){
           move2 = keypush_value;
         }
      }else{
         if (keypush_id == 1){
            move1 = 'n'; //Dont move
         }
         if (keypush_id == 2){
           move2 = 'n';
         }
      }

   } // END WHILE(1)
   PT_END(pt);
} // thread keypush

static PT_THREAD (protothread_serial(struct pt *pt))
{
   PT_BEGIN(pt);
   static char junk;
   //
   //
   while(1){
      //printf("Serial Thread running\n");
      // There is no YIELD in this loop because there are
      // YIELDS in the spawned threads that determine the
      // execution rate while WAITING for machine input
      // =============================================
      // NOTE!! -- to use serial spawned functions
      // you MUST edit config_1_3_2 to
      // (1) uncomment the line -- #define use_uart_serial
      // (2) SET the baud rate to match the PC terminal
      // =============================================

      // now wait for machine input from python
      // Terminate on the usual <enter key>
      PT_terminate_char = '\r' ;
      PT_terminate_count = 0 ;
      PT_terminate_time = 0 ;
      // note that there will NO visual feedback using the following function
      PT_SPAWN(pt, &pt_input, PT_GetMachineBuffer(&pt_input) );

      // Parse the string from Python
      // There can be toggle switch, button, slider, and string events
      // slider
      if (PT_term_buffer[0]=='s'){
         sscanf(PT_term_buffer, "%c %d %d", &junk, &slider_id, &slider_value);
         if (toggle_value == 1){
            // only schedule sldier events if in programming mode
            new_slider = 1;
         }
      }

      // pushbutton
```

```
      if (PT_term_buffer[0]=='b'){
         // signal the new game thread
         new_button = 1;
         // subtracting '0' converts ascii to binary for 1 character
         button_id = (PT_term_buffer[1] - '0')*10 + (PT_term_buffer[2] - '0');
         button_value = PT_term_buffer[3] - '0';
         difficulty = (PT_term_buffer[5] - '0'); //0, 1, or 2
         mode = (PT_term_buffer[7] - '0'); // 1 or 2
         time_trial_en = (PT_term_buffer[9] - '0');
      }

      // keypush
      if (PT_term_buffer[0]=='k'){
         // signal the player thread
         new_keypush = 1;
         keypush_id = PT_term_buffer[1] - '0';
         keypush_value = PT_term_buffer[2];
         keypush_down = PT_term_buffer[3];
      }

      // toggle switch
      if (PT_term_buffer[0]=='t'){
         // subtracting '0' converts ascii to binary for 1 character
         new_toggle = 1;
         toggle_id = (PT_term_buffer[1] - '0')*10 + (PT_term_buffer[2] - '0');
         toggle_value = PT_term_buffer[3] - '0';
      }

      // string from python input line
      if (PT_term_buffer[0]=='$'){
         // signal parsing thread
         new_string = 1;
         // output to thread which parses the string
         // while striping off the '$'
         strcpy(receive_string, PT_term_buffer+1);
      }
   } // END WHILE(1)
   PT_END(pt);
} // thread serial

// === string input thread =======================================================
// process text from python
static PT_THREAD (protothread_python_string(struct pt *pt))
{
   PT_BEGIN(pt);
   while(1){
      // wait for a new string from Python
      PT_YIELD_UNTIL(pt, new_string==1);
      new_string = 0;
      // parse frequency command
      if (receive_string[0] == 'f'){
      }
      //
      else if (receive_string[0] == 'v'){
      }
      //
      else if (receive_string[0] == 'h'){
         // help
         printf("help ...list the avaliable commands\r");
         // default string
         printf("Any other string is just echoed back\r");
      }
      else if (receive_string[0] == 't'){
         char buffer[20];
         printf(buffer);
      }
```

```
        //
        else {
        }
    } // END WHILE(1)
    PT_END(pt);
} // thread python_string


// === Main  ======================================================

void main(void) {
    //SYSTEMConfigPerformance(PBCLK);
    // ANSELA = 0;
    // ANSELB = 0;

    // === config threads ==========
    // turns OFF UART support and debugger pin, unless defines are set
    PT_setup();

    INTEnableSystemMultiVectoredInt();

    OpenTimer2(T2_ON | T2_SOURCE_INT | T2_PS_1_16, timer_match);
    ConfigIntTimer2(T2_INT_ON | T2_INT_PRIOR_2);
    mT2ClearIntFlag(); // and clear the interrupt flag


    // init the display
    tft_init_hw();
    tft_begin();
    // tft_fillScreen(ILI9340_BLACK);
    //240x320 vertical display
    tft_setRotation(1); // Use tft_setRotation(1) for 320x240

    pt_add(protothread_newgame, 0);
    pt_add(protothread_player, 0);
    pt_add(protothread_keypush, 0);
    pt_add(protothread_python_string, 0);
    pt_add(protothread_serial, 0);
    pt_add(protothread_endgame, 0);


    // init the threads
    PT_INIT(&pt_sched);

    pt_sched_method = SCHED_ROUND_ROBIN ;

    dimensionx = 12; // dimensionx x dimensiony maze
    dimensiony = 9;
    start = 0;
    end = 0;

    // hard: 24x18
    // med:  16x12
    // easy: 12x9

    xleft = 0; // set maze bounds
    xright = 320;
    ybottom = 0;
    ytop = 240;
    xlen = (xright-xleft) / dimensionx;  // set wall spacings
    ylen = (ytop - ybottom) / dimensiony;

    // === Draw a dimension x dimension grid on the tft
    drawGrid();
    // === Create a maze that is dimension x dimension in size ================
    static int i;
    static int j;
```

```
    for (i = 0; i < MAX_DIM_X; i++){
        for (j = 0; j < MAX_DIM_Y; j++){
            static node_t node;
            node.x = i;
            node.y = j;
            // initially set all walls to True
            node.north = 1;
            node.south = 1;
            node.east = 1;
            node.west = 1;
            // set backpointers to NULL initially
            node.backptr_x = NULL;
            node.backptr_y = NULL;
            // initially not in frontier set
            node.inFrontier = 0;
            //Add to array
            nodes[i][j] = node;
        }
    }

    generateMaze();
    drawPlayer();

    // === For debugging, print out the maze to the python interface
    // printMaze();
    // tft_drawPixel(100, 100, ILI9340_WHITE); //draw user1 dot
    // tft_fillCircle(getCenterX(0), getCenterY(0), radius, ILI9340_WHITE); //draw user2 dot

    // round-robin scheduler for threads
    PT_SCHEDULE(protothread_sched(&pt_sched));
} // main

// === end  ======================================================
```

```
import PySimpleGUI as sg
# on linux:
# sudo pip3 install pyserial
# /home/bruce/.local/lib/python3.5
# sudo python3.5 pic_target_3.py
#
# you can find out more about PySimleGUI at
# https://pysimplegui.readthedocs.io/en/latest/
#
# --event format to PIC--
# Four sharacters for each non-string event:
# pushbutton event 'b' + 2 digit button number + value (1,0)
# toggle sw event 't' + 2 digit button number + value (1,0)
# slider event 's' + 1-digit slider number + n digit value
# listbox event 'l' + + 1-digit listbox number + 1 digit selection number
# radio button 'r' + 1 digit group number + 1 digit selection numbr
# -- string --
# strings typed in the input line are sent in their entirety.
# -- reset --
# RESET has NO code on PIC!
# serial reset event sends a rs-232 BREAK which is connected
# through a filter to MCLR pin
#
# Python_TX_pin--(100ohm)--(+Schottky Diode-)------>(target MCLR pin)
#                                    |
#                                (10uf)------(1kohm)
#                                  |          |
#                                  ------------
#                                      |
#                                   (PIC gnd)
#
import time
import serial
import keyboard
import os
# open microcontroller serial port
# For windows the device will be 'COMx'
ser = serial.Serial('COM4', 115200, timeout=0.001)  # open serial port 38400

# sg.theme('DarkBlack1')   # Add a touch of color
# All the stuff inside your window.
# This a heirachical list of items to be displayd in the window
# First list is first row controls, etc
# Buttons:
#   Realtime buttons respond to push-events
#   After the window is defined below, release events may be bound to each realtime button
#   The 'key' for each button must be of the form 'pushbutNN',
#   where 'NN' are digits 0-9 defining the button number
# Toggles:
#   Toggle switches are actually checkboxes
#   The 'key' for each checkbox must be of the form 'toggleNN',
#   where 'NN' are digits 0-9 defining the checkbox number
# Text
#   The text input field acts like the one-line Arduino serial send box.
#   The multiline output box receives serial from the PIC. text typed here is ignored.

font_spec = 'Courier 24 bold'
heading_color = '#1e2d7c'
toggle_color = '#e5dcb3'

#Read and display current highscores
highscores = {'0':"None", '1':"None", '2':"None"} # Easy, Med, Hard
highscoreFile = open("maze_highscores.txt", "a")
highscoreFile.close()
highscoreFile = open("maze_highscores.txt", "r+")
scores = highscoreFile.readlines()
highscoreFile.close()
```

```
for s in scores:
  highscores[s[0]] = s[2:].replace('\n', '')

layout = [
        [sg.Text('Maze Interface',background_color=heading_color, text_color='#ffffff')],
        [sg.Text('Difficulty',background_color=heading_color, text_color='#ffffff')],
        [sg.Radio('Easy', "RADIO0", default = True, key='radio0', font='Helvetica 12', enable_events=True),
          sg.Radio('Medium ', "RADIO0", default = False, key='radio1', font='Helvetica 12', enable_events=True),
          sg.Radio('Hard', "RADIO0", default = False, key='radio2',font='Helvetica 12', enable_events=True)],
        [sg.Text('Highscores',background_color=heading_color, text_color='#ffffff')],
        [sg.Text('Easy',background_color=heading_color, text_color='#ffffff'),
          sg.Text('Medium',background_color=heading_color, text_color='#ffffff'),
          sg.Text('Hard',background_color=heading_color, text_color='#ffffff')],
        [sg.Text(highscores['0'],   key= "0", background_color=heading_color, text_color='#ffffff'),
          sg.Text(highscores['1'], key= "1", background_color=heading_color, text_color='#ffffff'),
          sg.Text(highscores['2'], key= "2", background_color=heading_color, text_color='#ffffff')],
        [sg.Text('Mode',background_color=heading_color, text_color='#ffffff')],
        [sg.Radio('1 Player', "RADIO1", default = True, key='radio3', font='Helvetica 12', enable_events=True),
          sg.Radio('2 Player', "RADIO1", default = False, key='radio4', font='Helvetica 12', enable_events=True),
          sg.Checkbox('Time Trial Enable', key='t_en', font='Helvetica 8', enable_events=True,
           background_color=heading_color,text_color='#ffffff')],
        [sg.RealtimeButton('New Game', key='pushbut01', font='Helvetica 12')],
        [sg.Text('System Controls', background_color=heading_color, text_color='#ffffff')],
        [sg.Button('Exit', font='Helvetica 12')],
        [sg.Checkbox('reset_enable', key='r_en', font='Helvetica 8', enable_events=True,
           background_color=heading_color,text_color='#ffffff'),
           sg.Button('RESET PIC', key='rtg', font='Helvetica 8')],
      ]

# Create the Window
window = sg.Window('ECE4760 Maze Game Interface', layout, location=(0,0),
                   return_keyboard_events=True, use_default_focus=True,
                   element_justification='c', finalize=True)


# Event Loop to process "events"
# event is set by window.read
event = 0
#
# button state machine variables
button_on = 0
button_which = '0'

selected_difficulty = 0 # initialize diffulty and mode
selected_mode = 1
time_trial_enable = 0

#initialize movement key states
Wstate = 0
Astate = 0
Sstate = 0
Dstate = 0

UPstate = 0
DOWNstate = 0
LEFTstate = 0
RIGHTstate = 0

while True:
  # p1 keyboard interrupts
  if keyboard.is_pressed('w') != Wstate:  # if key 'w' is pressed
    Wstate = not Wstate
    if Wstate:
      ser.write(('k1up\r').encode()) #keyboard, p1, up, pressed
    else:
      ser.write(('k1ur\r').encode()) #keyboard, p1, up, released
```

```
if keyboard.is_pressed('s') != Sstate:
    Sstate = not Sstate
    if Sstate:
        ser.write(('k1dp\r').encode()) #keyboard, p1, down, pressed
    else:
        ser.write(('k1dr\r').encode()) #keyboard, p1, down, released
elif keyboard.is_pressed('a') != Astate:
    Astate = not Astate
    if Astate:
        ser.write(('k1lp\r').encode()) #keyboard, p1, leftt, pressed
    else:
        ser.write(('k1lr\r').encode()) #keyboard, p1, left, released
elif keyboard.is_pressed('d') != Dstate:
    Dstate = not Dstate
    if Dstate:
        ser.write(('k1rp\r').encode()) #keyboard, p1, right, pressed
    else:
        ser.write(('k1rr\r').encode()) #keyboard, p1, right, released


# p2 keyboard interrupts
if keyboard.is_pressed('up arrow') != UPstate:
    UPstate = not UPstate
    if UPstate:
        ser.write(('k2up\r').encode()) #keyboard, p2, up, pressed
    else:
        ser.write(('k2ur\r').encode()) #keyboard, p2, up, released
elif keyboard.is_pressed('down arrow') != DOWNstate:
    DOWNstate = not DOWNstate
    if DOWNstate:
        ser.write(('k2dp\r').encode()) #keyboard, p2, down, pressed
    else:
        ser.write(('k2dr\r').encode()) #keyboard, p2, down, released
elif keyboard.is_pressed('left arrow') != LEFTstate:
    LEFTstate = not LEFTstate
    if LEFTstate:
        ser.write(('k2lp\r').encode()) #keyboard, p2, left, pressed
    else:
        ser.write(('k2lr\r').encode()) #keyboard, p2, left, released
elif keyboard.is_pressed('right arrow') != RIGHTstate:
    RIGHTstate = not RIGHTstate
    if RIGHTstate:
        ser.write(('k2rp\r').encode()) #keyboard, p2, right, pressed
    else:
        ser.write(('k2rr\r').encode()) #keyboard, p2, right, released



# time out paramenter makes the system non-blocking
# If there is no event the call returns event  '__TIMEOUT__'
event, values = window.read(timeout=20) # timeout=10

# if user closes window using windows 'x' or clicks 'Exit' button
if event == sg.WIN_CLOSED or event == 'Exit': #
    break

# assign variables to set correct New Game
if event[0:3] == 'rad':
    if event[5] == '0':
        selected_difficulty = 0 #Easy Difficulty
    elif event[5] == '1':
        selected_difficulty = 1 #Medium Difficulty
    elif event[5] == '2':
        selected_difficulty = 2 #Hard Difficulty
```

```
    elif event[5] == '3':
      selected_mode = 1        #Single Player Mode
    else:
      selected_mode = 2        #Two Player Mode

  # toggle events
  if event == 't_en':
    time_trial_enable = window.Element('t_en').get()


  # button events
  if event[0:3]  == 'pus' and button_on == 0 :
    # 'b' for button, two numeral characters, a '1' for pushed, and a terminator
    ser.write(('b' + event[7:9] + '1' + 'd'+ str(selected_difficulty) + 'm' + str(selected_mode) + 't' + str(time_trial_enable) +
'\r').encode())


   # reset events
  switch_state = window.Element('r_en').get()
  if event[0:3] == 'rtg' and switch_state == 1 :
    # drops the data line for 100 mSec
    ser.send_break() #optional duration; duration=0.01
    window['r_en'].Update(value = False)
    window['radio0'].Update(value = True) #Reset button to easy
    window['radio3'].Update(value = True)
    window['t_en'].Update(value = False) #not in time trial

  # character loopback from PIC
  pic_str = ""
  while ser.in_waiting > 0:
    pic_char = chr(ser.read(size=1)[0])
    pic_str += pic_char
  if (pic_str != ""):
    #Get difficulty
    diff= pic_str[0]
    #Get time
    time = pic_str[2:]

    highscoreFile = open("maze_highscores.txt", "a") #make file if not created
    highscoreFile.close()
    highscoreFile = open("maze_highscores.txt", "r+")
    lines = highscoreFile.readlines()
    highscoreFile.close()
    prev = 0 #track if previous score has been set at difficulty
    for i in range(len(lines)):
      if lines[i][0] == diff:
        prev = 1
        if float(time) < float(lines[i][2:]):
          lines[i] = pic_str + '\n'
          #Update highscores live on the GUI
          highscores[diff] = pic_str[2:].replace('\n', '')
          print(highscores)
          window[diff].Update(value = highscores[diff])
    if not prev:
      try:
        if (int(pic_str[0]) <= 3):
          lines.append(pic_str + '\n')
          highscores[diff] = pic_str[2:].replace('\n', '')
          window[diff].Update(value = highscores[diff])
      except:
        pass
    highscoreFile = open("maze_highscores.txt", "w") #write back scores
    highscoreFile.writelines(lines)
    highscoreFile.close()

# close port and Bail out
```

```
ser.close()
window.close()
```