University of California Riverside

Intro to Embedded Systems

EE/CS 120B

Custom Project Final Report:

Tetris Game

Instructor: Philip Brisk

Arturo Perez

March 14, 2019

Table of Contents

Introduction..... 2

Hardware..... 3

- Components..... 3
- Hardware connections.....3

Software..... 4

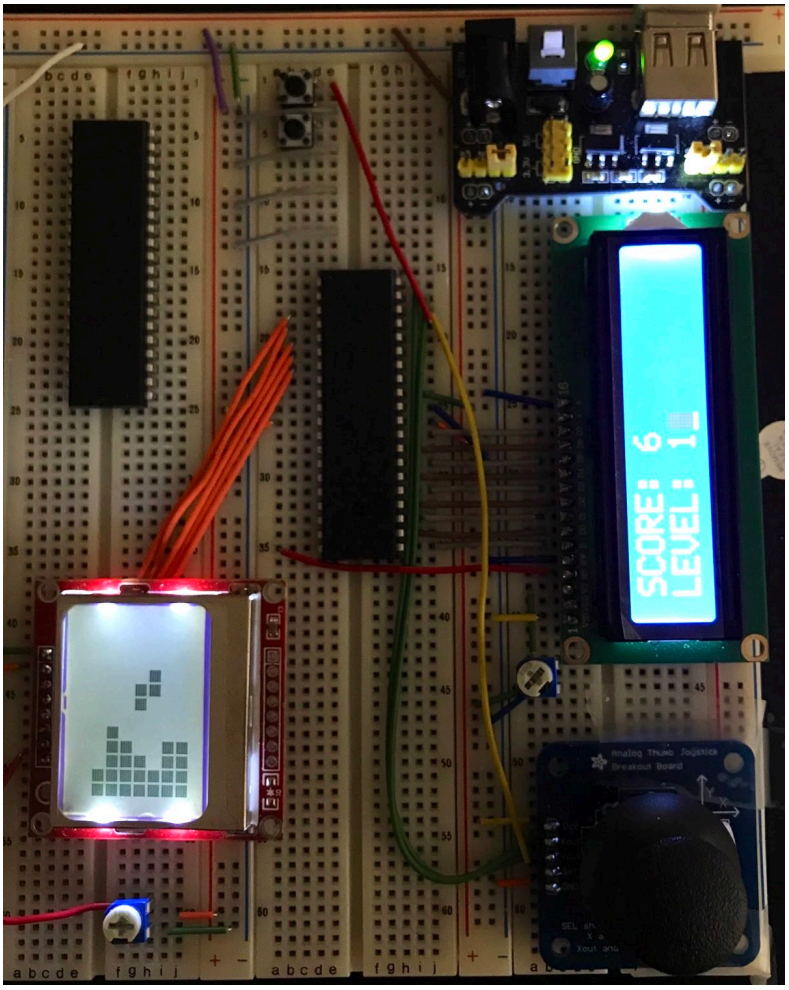
Complexities 10

Video Demo Link.....10

Bugs and shortcomings 10

Introduction:

Tetris is a two-dimensional game where players match pieces in an organized way. For this project, pieces appear at the top of the screen and move downwards towards the bottom. During this time, players can change the position of the pieces on the screen by moving the joystick on the left or right directions. Pressing the joystick in the on its center pushes an integrated button rotating the shape 90 degrees clockwise. Moving to the down direction of the joystick speeds up the drop-down of the piece. Points are earned each time the player places a shape. Levels are increased when one or more lines are complete across the screen. Players loose the game if they reach the top of the screen. A second screen is used to display scores and game information. The game can be reseted at anytime by pressing a button.

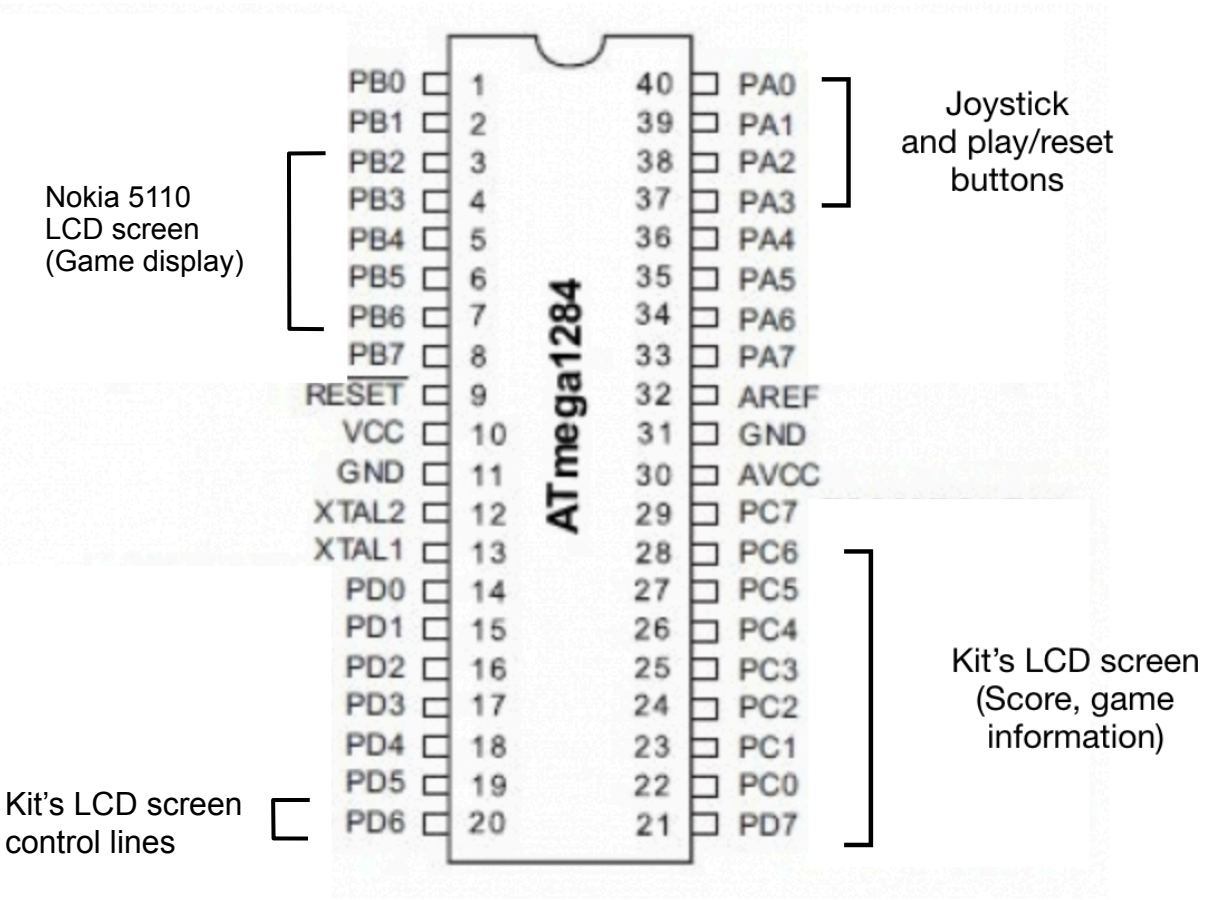


Hardware

This project includes the following hardware pieces.

- ATMega1284 microcontroller
- Nokia 5110 LCD Screen
- Analog Joystick
- Buttons
- Kit's LCD screen

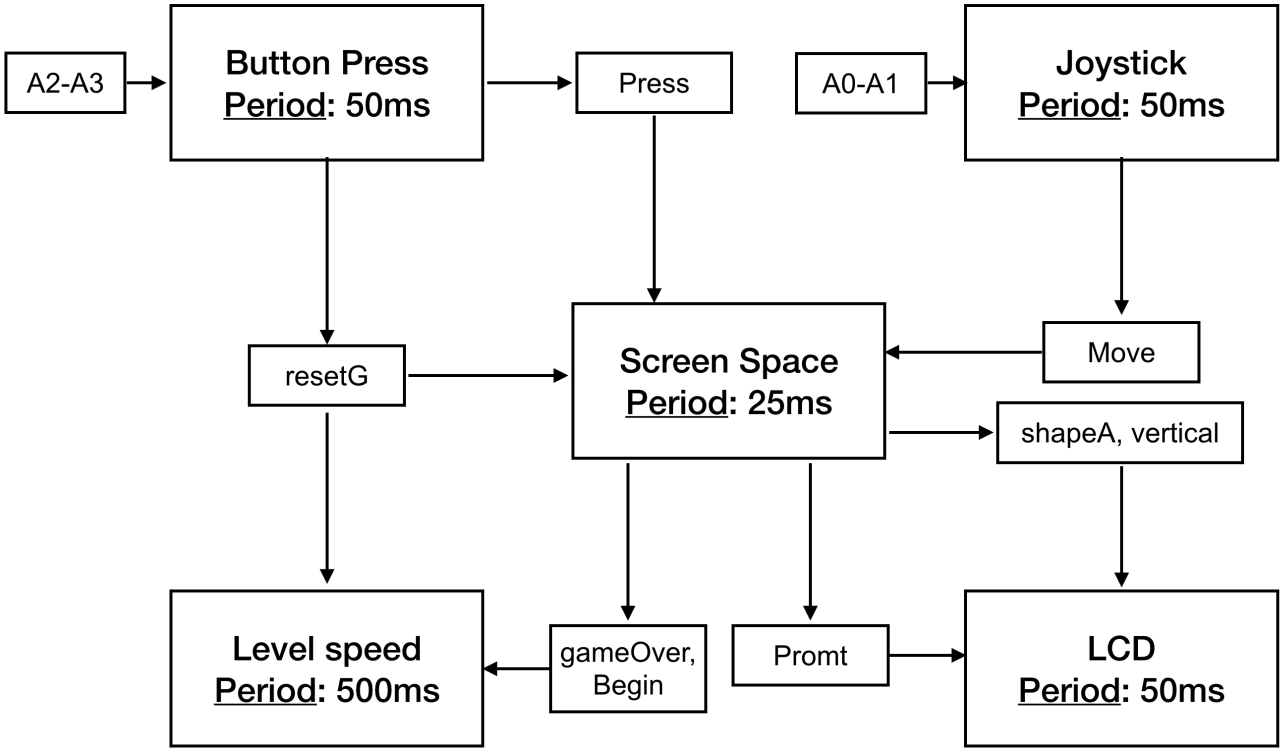
Hardware Connections:



Software

The project consists of a collection of Concurrent State Machines and game functions. State machines structure the game by sampling inputs, writing outputs, and providing sequential order to the game. Game functions calculate the orientation in space of shapes, their physical characteristics and behaviors. Below is a detailed description of SMs and functions used in this project.

Task Communication Diagram

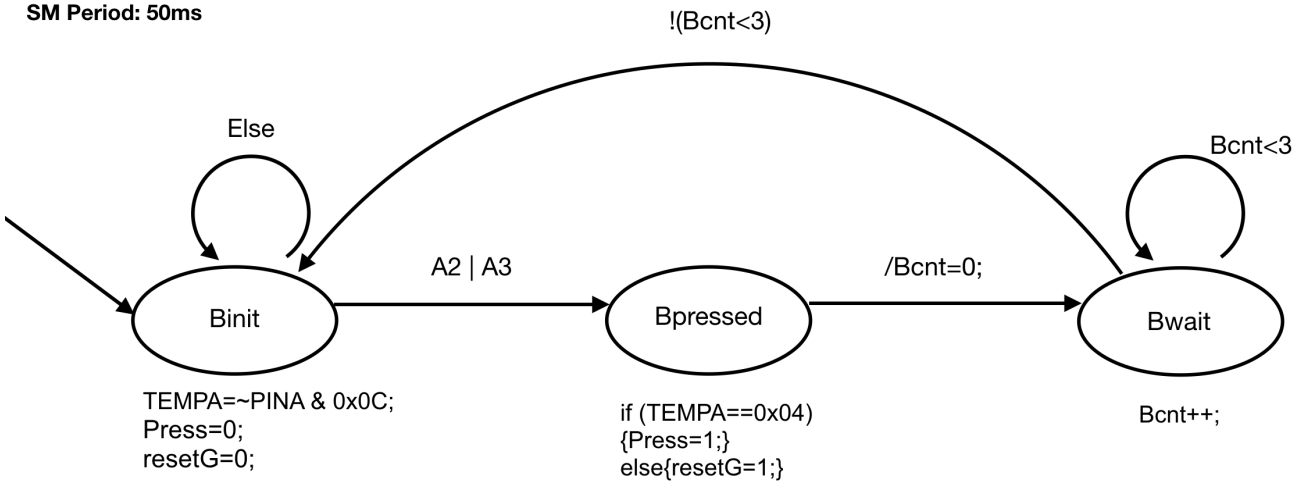


Final Project State Machine: Button Press

This State Machine reads button presses for the game and reset functionality.
It sets global flags for communication with other SM's on the program.

Used Variables:
unsigned char Bcnt; // Counter for sampling rate
unsigned char Press; // Button press flag
unsigned char resetG; // Reset button flag
unsigned char TEMPA; // Stores the readings from port A

SM Period: 50ms

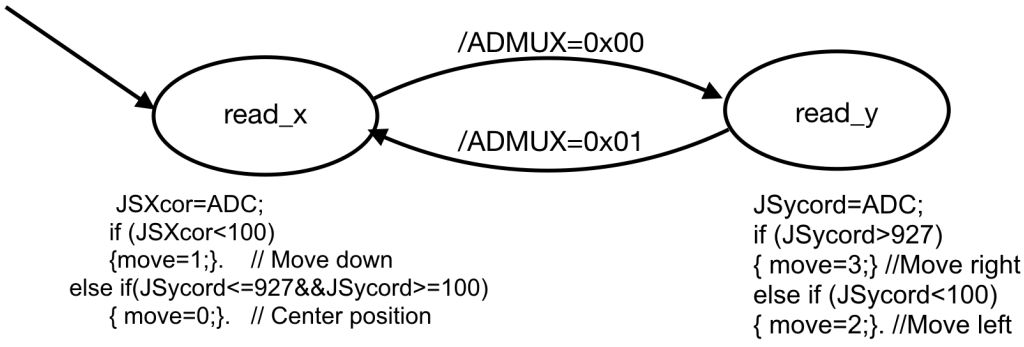


Final Project State Machine: Joystick

This State Machine reads the ADC ports A0 and A1 and compares analog readings to different thresholds in order to define directions of the Joystick.

Used Variables:
unsigned char move; // Stores the movement direction.
unsigned long JSXcord; // Stores the X value read from ADC
unsigned long JSYcord; // Stores the Y value from ADC

SM Period: 50ms

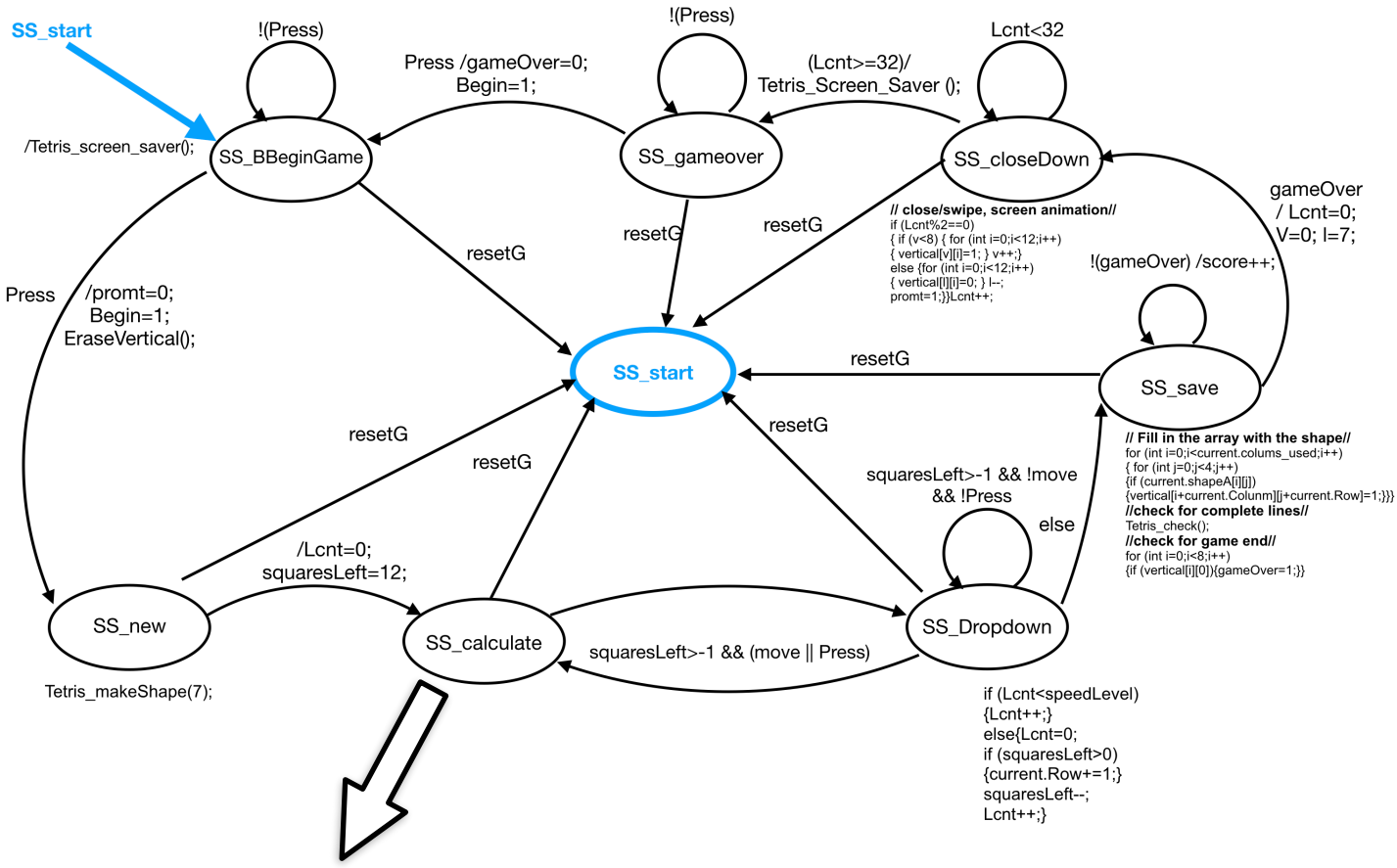


Final Project State Machine: Screen Space

This State Machine is the main driver of the game. It reads inputs from other SM's to calculates the position of the shapes on the screen at any given time. The SM also calculates positions after changes in location and face of the shape. It keeps track of the squares left before coming in contact with another shape or boundary. Once the shape reaches its destination, the shape is aded to a two dimensional array (vertical) that represents the occupied squares on the game screen. Then, a new shape is created with a randomizer function. The State Machine also handles the gameOver state, closing animation screen, and a screen saver for waiting times.

Used Variables:
signed char squaresLeft; // Number of positions left before shape touches the ground/shapes
unsigned char Lcnt; // counter for drop-down speed, as levels increase, shape drop-down speeds up
unsigned char v=0; // Index variable for close down animation
unsigned char l=7; // Index variable for close down animation
unsigned char gameOver=0; //Game over flag, notifies Speed level SM when the game finalizes
unsigned char promt=0; // Flag that informs the LCD SM to print screen saver or close down animation.
unsigned char Begin=0; // Flag that informs Speed level SM when the game begins.

SM Period: 25ms



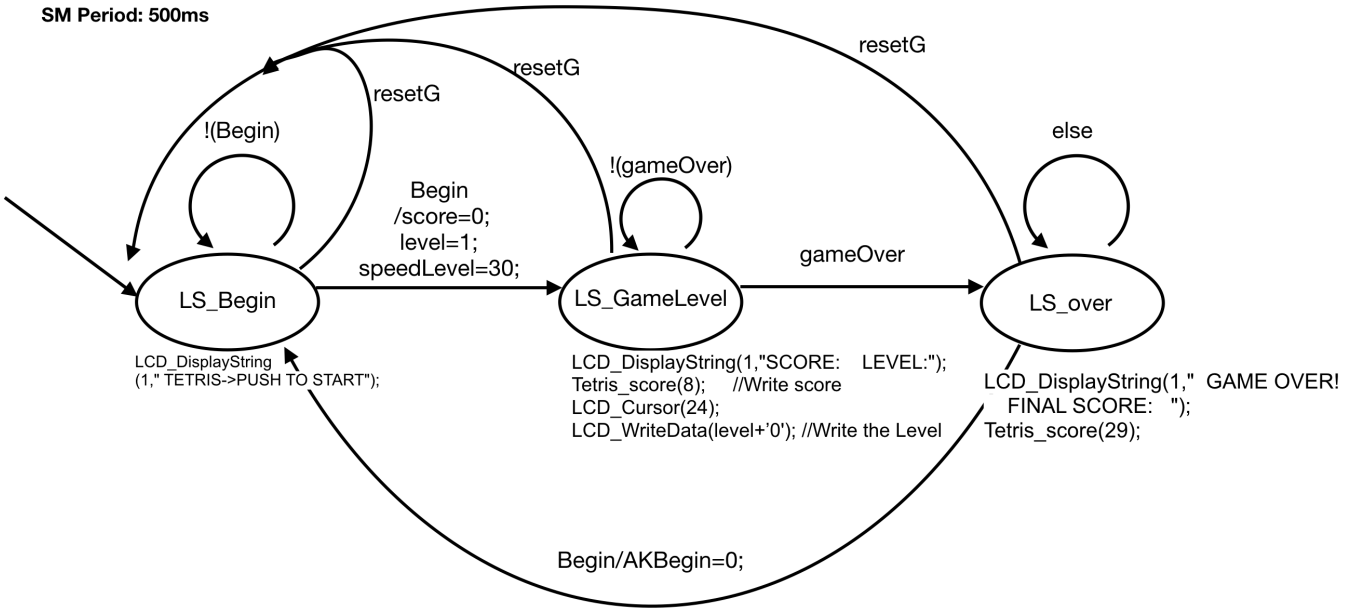

```
if (move==1)   //Move shape down
{ if (squaresLeft>0)
{current.Row+=1;
for (int i=0;i<current.colums_used;i++)     // Recalculate the the postion of the shape after moving down
{squaresLeft=Tetris_min(squaresLeft,12-Tetris_Bottom_up(i+current.Colunm)-current.Row-used(i));}}
}
else if (move==2){//Move Shape Left
if(current.Colunm+(current.colums_used-1)<7 && Tetris_safe_L_Move(current.Colunm,current.Row)){current.Colunm+=1;}//Check if it is safe to move left
for (int i=0;i<current.colums_used;i++)   // Recalculate the the postion of the shape after moving left
{squaresLeft=Tetris_min(squaresLeft,12-Tetris_Bottom_up(i+current.Colunm)-current.Row-used(i));}
}
else if (move==3)//Move Shape Right
{squaresLeft=12;
if (current.Colunm>0 && Tetris_safe_R_Move(current.Colunm,current.Row)){current.Colunm-=1;}//Check if it is safe to move right
for (int i=0;i<current.colums_used;i++)// Recalculate the the position of the shape after moving right
{squaresLeft=Tetris_min(squaresLeft,12-Tetris_Bottom_up(i+current.Colunm)-current.Row-used(i));}
}
else if (Press)  //Rotate/change the shape's face
{ squaresLeft=12;
if (current.face==4){current.face=1;}
else{current.face+=1;}
changeFace(current.type,current.face);
for (int i=0;i<current.colums_used;i++)// Recalculate the the position of the shape after shifting face
{squaresLeft=Tetris_min(squaresLeft,12-Tetris_Bottom_up(i+current.Colunm)-current.Row-used(i));}
Press=0;}
else{squaresLeft=13;
for (int i=0;i<current.colums_used;i++)// Calculate the the position after creating a new shape
{squaresLeft=Tetris_min(squaresLeft,12-Tetris_Bottom_up(i+current.Colunm)-current.Row-used(i));}}
```

Final Project State Machine: Level Speed

This State Machine controls the prompts on the second screen as well of displaying the score and level.

Used Variables:
unsigned char score; // Variable to store the score
unsigned char level; // Variable to store the level
unsigned char speedLevel; //Variable that keeps the speed of the shape drop-down

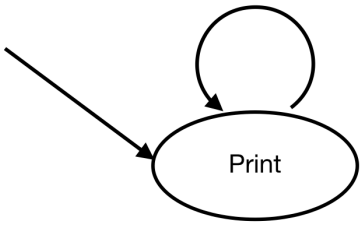
SM Period: 500ms



Final Project State Machine: LCD

This State Machine updates the Nokia 5110 LCD screen with the shapes and their current locations on the screen

SM Period: 50ms




```
      if (!promt)
{     for (int i=0;i<4;i++)      // Print the current shape
{ for (int j=0;j<4;j++)
{ if (current.shapeA[i][j])
{ int8_t c=Xlength*(j+current.Row);
int8_t r=Ylength*(i+current.Column);
if (c>=0)
{Tetris_Draw_A_Sqaure(c,r);}}}}

for (int i=0;i<8;i++)           //Print the rest of the squares in the game
{ for (int j=0;j<12;j++)
{  if (vertical[i][j])
{uint8_t c=Xlength*j;
uint8_t r=Ylength*i;
Tetris_Draw_A_Sqaure(c,r);}}}

else{for (int i=0;i<8;i++)
{ for (int j=0;j<12;j++)        // If prompt, Print the contents of vertical,
                                   usually the screen saver.
{  if (vertical[i][j])
{uint8_t c=Xlength*j;
uint8_t r=Ylength*i;
Tetris_Draw_A_Sqaure(c,r);}}}}
```

Game Functions

This is overview of the main game functions used in the project. See code folder for complete functions.

```
//==================TETRIS GAME FUNCTIONS==============///
const uint8_t Xlength=7;                        // Lenght of a single square
const uint8_t Ylength=6;                        // Width of the single square
enum ShapeType{T,RevL,Z,RevZ,L,I,Square}Shpe;   // Type definition for different shapes

unsigned char Tetris_min(unsigned char a,unsigned char b)  // Calculates the minimun of two numbbers
{return (a<b)? a:b;}

struct Shape        // Definition of a shape
{
    uint8_t type;   // Stores the type
    uint8_t face;   // Stores the current face
    int8_t Row;     // Stores the current row
    uint8_t Columm; // Stores current columm
    uint8_t colums_used; // Columms used in the 4x4 array by the current shape
    unsigned char shapeA[4][4]; // Array that stores the squares in the shape
};
struct Shape current;
unsigned char vertical[8][12];
unsigned char RCount;
unsigned char speedLevel=30;

/*  This function checks whether is safe to move a shape to the
 right: if there is enough space available along the shape's squares,
 returns 1, else return 0.
 */
uint8_t Tetris_safe_R_Move(uint8_t col, uint8_t row)
{ uint8_t safe=1;
    for (int j=0; j<4 ;j++)
    { if (current.shapeA[0][j]==0 && current.shapeA[1][j]==1)
        { if (vertical[col][j+row]==1)
             {safe=0;}
        }
      if(current.shapeA[0][j]==1)
      {  if (vertical[col-1][j+row]==1)
             {safe=0;}
      }
    }
    return safe;
}
/*
 This function checks whether is safe to move a shape to the left: if
 there is enough space available along the shape's squares, returns 1,
 else return 0.
 */
uint8_t Tetris_safe_L_Move(uint8_t col, uint8_t row)
{ uint8_t safe=1;
    for (int j=0; j<4 ;j++)
    { if (current.shapeA[current.colums_used-1][j]==0 && current.shapeA[current.colums_used-2][j]==1)
        { if (vertical[col+current.colums_used-1][j+row]==1)
            {safe=0;}
        }
        if(current.shapeA[current.colums_used-1][j]==1)
        {  if (vertical[col+current.colums_used][j+row]==1)
            {safe=0;}
        }
    }
```

```
/*  This function is called whenever a shape has been placed to check if
 any row (line from left to right in the game screen) has been completed.
 If so, the function calls EraseRow to delete the row.
 */
void Tetris_check(){
    RCount=0;
        for (int j=11;j>=2;j--)
        { for (int i=0;i<8;i++)
            { if (vertical[i][j])
                {RCount++;}
            }
            if (RCount==8)
            { EraseRow(j);
                j=j+1;
            }
            RCount=0;
        }
}
unsigned char score;            // Keeps the score of the game
unsigned char level;            // Keeps the level of the game

/*  EraseRow function deletes a row that is completed during game play.
 It also increases the level and updates the drop-down speed of the game.
 */
void EraseRow(unsigned char R)
{
    for (int i=0;i<8;i++)
    { for (int k=R;k>=2;k--)
        {vertical[i][k]=vertical[i][k-1];}
    }
    level++;
    score=score+5;

    if (level>6)
    {speedLevel=speedLevel-1;}
    else{speedLevel=speedLevel-4;}

}
uint8_t prev=0;
/*  This function creates a new shape and stores its values in current.
 It uses a randomizer function to select a new shape. It then calls changeFace
 function to fill in the shape array with it correponding values.
 */
void Tetris_makeShape(uint8_t param)
{
    uint8_t RandomNum = rand() % param;
     while (prev==RandomNum)
     { RandomNum=rand()%param; }
    prev=RandomNum;
    current.type=RandomNum;
    current.face=1;
    current.Row=-3;
    {current.Column=3;}
    changeFace(current.type,current.face);

}
/*  This function counts the number of occupied squares for a specific column c.
 */
unsigned char Tetris_Bottom_up(unsigned char c){

    for(int i=0;i<12;i++)
    {if (vertical[c][i])
        { if (i<=current.Row)
            {  if ((current.type==L&&current.face==1)||(current.type==RevL&&curren
                    {i=current.Row+1;}
                else{i=current.Row;}
                while (vertical[c][i]==1)
                {i++;}
               while(vertical[c][i]==0 && i<12)
               {i++;}
                return (12-i);
                break;
            }
        else{return (12-i);}
        break;}
    }
    return 0;
}
/*  This function counts the number of occupied squares in the shape array.
 */
unsigned char used(unsigned char c){
    for(int i=3;i>=0;i--)
    {if (current.shapeA[c][i])
        {return (i+1);
        break;}
    }
    return 0;
}
/*  This function draws a customized square for the game shapes
 */
void Tetris_Draw_A_Sqaure(uint8_t x, uint8_t y)
{
    for (int i=x+1;i<x+6+1;i++)
    for (int j=y;j<y+5;j++)
    {{nokia_lcd_set_pixel(i,j,1);}}
}
/* This function draws a customized screen saver for the idle times
 on the nokia screen. For instance when the player losses the game
 or is about to begin a new game.
 */
void Tetris_screen_saver()
{
        Tetris_erase_shape();
   //Draw chess-board like pattern at the top of the screen//
    for (int i=0;i<8;i++)
    {for(int j=0;j<4;j++)
        { vertical[i][j]=0;
            if ((i%2)==0)
            {if ((j%2)==0)
                {vertical[i][j]=1;}
            }
            else{if (((j+1)%2)==0)
                {vertical[i][j]=1;}
```

```
/* This function deletes the current shape on the array
 */
void Tetris_erase_shape()
{
        for (int i=0;i<4;i++)
        { for (int j=0;j<4;j++)
            { current.shapeA[i][j]=0;}}
}
/* This function replaces the current shape in the array for
 a rotated version.
 */
void changeFace(uint8_t type,uint8_t face)
{
    if (type==Square)
    { if(face==1||face==2||face==3||face==4)
        {   current.colums_used=2;
            Tetris_erase_shape();
            current.shapeA[0][0]=1;
            current.shapeA[0][1]=1;
            current.shapeA[1][1]=1;
            current.shapeA[1][0]=1;
        }
}
    if (type==I)
    { if(face==1||face==3)
        {   current.colums_used=1;
            Tetris_erase_shape();
            current.shapeA[0][0]=1;
            current.shapeA[0][1]=1;
            current.shapeA[0][2]=1;
            current.shapeA[0][3]=1;
    }
    if (face==2||face==4)
    {
            if (!Tetris_safe_L_Move(current.Column+3,current.Row)||!Tetris_safe_L_Move(current.Column+2,current.Row)||!
                Tetris_safe_L_Move(current.Column,current.Row)||!Tetris_safe_L_Move(current.Column+1,current.Row) ||
                current.Column+(current.colums_used-1)==7)
            {current.face=1;}
```

Complexities:

- Communication and adjustment of analog joystick.
- Nokia 5110 serial bus communication (included library from https://github.com/LittleBuster/avr-nokia5110) and customized character types.
- Game logic and custom rendering functions.

Youtube Link

https://youtu.be/km8RTgAOW_I

Known Bugs and Shortcomings

- The reset button triggers the screen-saver. This in turn creates three random shapes to display on the screen. After pressing the reset button, the array of the shape is filled with squares making the shape indistinguishable. This behavior occurs only when the reset buttons is pressed. This bug could be caused as the array may no be properly erased or the random function gets called multiple times after pressing the reset button.
- The longest shape, the ‘l’ like stick, can not be rotated when there is less than 4 unfulfilled spaces to its left. In oder to rotate the shape, it must first moved to the right to allocate the space. The bug was also present for the other shapes and was solved in the rotation function by checking the surrounding area and making the proper shift on the shape. Due to the shape’s long form and the structure of the function, the rotate function does not apply for this shape. To solve this bug, a dedicated rotation function must be implemented.

Cited Sources

-Used libraries and functions from EE/CS 120B course.
-Used library from Github user https://github.com/LittleBuster/avr-nokia5110