1- Melhorias

Para essa etapa do projeto foi transferida parte do código que anteriormente estava codificado para arduino, no Energia, para a MSP430, no code composer studio. Foi alterado o tempo de timer para o início do jogo. Para o projeto, a interrupção será usada a cada um segundo para atualizar o display. A atualização do display consiste em contabilizar pontos, caso chegue ao final do jogo (game over) ou fazer a peça descer.
A parte em assembly do projeto será implementada para fazer o shift das peças do jogo, tanto para a direita quanto para a esquerda. Para a próxima etapa do jogo será finalizado o código para a MSP, também contará o placar ao final de cada ponto obtido, que será atualizado a cada novo ponto feito e será mostrado na matriz de leds e em seguida dará continuidade ao jogo e também contará com um modo de pause.

2- Código

```
#include <msp430f5529.h>
#include "TRandom.h"
#include "MatrixDisplay.h"
#include "Tetris.h"
#include "Effects.h"
#include "MIDIPlayer.h"
#include "LEDControl.h"
#include <math.h>

/** Port definitions **/
// MAX7219
int ledMatrixCLK = P1_4;        // MAX7219 clock port
int ledMatrixCS = P1_5;         // MAX7219 load port
int ledMatrixDIN = P1_3;        // MAX7219 Data In port

// press buttons
int goRigthSwitchPin = 4;       // go rigth switch pin
int goDownSwitchPin = 6;        // go down switch pin
int rotateSwitchPin = 7;        // rotate switch pin
int goLeftSwitchPin = 3;        // go left switch pin
int pausePin = 5;               // pause switch pin

//level counter led pins
int counterLed_0 = 14;          // counter LED #0
int counterLed_1 = 15;          // counter LED #1
int counterLed_2 = 16;          // counter LED #2
int counterLed_3 = 17;          // counter LED #3

// next piece led pins
int nextPieceLed_0 = 13;        // next piece LED #0
int nextPieceLed_1 = 18;        // next piece LED #1
int nextPieceLed_2 = 19;        // next piece LED #2

//level counter led status
unsigned char counterLedStatus[5] = {0, 0, 0, 0, 0};
// next piece led status
unsigned char nextPieceLedStatus[3] = {0, 0, 0};

// button flags
unsigned char goRigthFlag = 0x00;       // go rigth interrupt request
unsigned char goLeftFlag = 0x00;        // go left interrupt request
unsigned char goDownFlag = 0x00;        // go down interrupt request
unsigned char rotateFlag = 0x00;        // rotate piece interrupt request
```

```
// switches control
unsigned char lastGoRigthSwitchStatus = 0x00;       // last go rigth switch pin status
unsigned char lastGoDownSwitchStatus = 0x00;        // last go down switch pin status
unsigned char lastRotateSwitchStatus = 0x00;        // last rotate switch pin status
unsigned char lastGoLeftSwitchStatus = 0x00;        // last go left switch pin status

// speaker
int speakerPin = 12;                  // speaker pin

// timer variables
int timer1_counter;                   // global timer
unsigned char gameTimeStep = 0x00;  // game timer step

/*
 * interrupt service routine
 * lauch freq equals to 64Hz
 */

void ISR_update() {

    //TCNT1 = timer1_counter;     // preload timer
     noInterrupts();                // disable all interrupts

    // update the next piece and level LED status
                setCounter(counterLedStatus, getLevel());
                setPiece(nextPieceLedStatus, getNextPiece());
                    digitalWrite(counterLed_0, counterLedStatus[0]);
                    digitalWrite(counterLed_1, counterLedStatus[1]);
                    digitalWrite(counterLed_2, counterLedStatus[2]);
                    digitalWrite(counterLed_3, counterLedStatus[3]);
                    digitalWrite(nextPieceLed_0, nextPieceLedStatus[0]);
                    digitalWrite(nextPieceLed_1, nextPieceLedStatus[1]);
                    digitalWrite(nextPieceLed_2, nextPieceLedStatus[2]);

   // sets the counter limit (game frequency control))
   if (gameTimeStep == getTimeEnd()) {
      gameTimeStep = 0x00;
   } else {
      gameTimeStep++;
   }

   // sets the go down flag to true
         if (gameTimeStep == 0x00 && isGamePaused() == 0x00) {
      goDownFlag = 0x01;
   }

    MidiCLKTrigger();              // triggs the MIDIPlayer

     interrupts();                  // enable all interrupts

   if (isGamePaused() == 0x00) {
       updateDisplay(getField());  // updates field if game is not paused
   }
}

/*
  * Sets the microcontroller pin mode, matrix LCD,
 * timers and starts the game
 */
void setup() {

   //TACCR0 = 49999;
   //TACCTL0 = CCIE;
                TA0CTL = MC_1 | ID_3|TASSEL_1|TACLR;
```

```
  //TA0CTL=0b0000001011010010;
   TA0CCR0=2065951; // Set TACCR0 = 2000 to generate a 1s timebase @ 16MHz with a divisor of 8
    TA0CCTL0=BIT4;// Enable interrupts when TAR = TACCR0

  // LED matrix SETUP
      pinMode(ledMatrixCLK, OUTPUT); // clock pin
       pinMode(ledMatrixCS, OUTPUT); // load pin
       pinMode(ledMatrixDIN, OUTPUT); // data in pin

  // switches setup
     pinMode(goRigthSwitchPin, INPUT); // rigth switch
     pinMode(goDownSwitchPin, INPUT); // down sitch
       pinMode(rotateSwitchPin, INPUT); // rotate switch
      pinMode(goLeftSwitchPin, INPUT); // left switch
   pinMode(pausePin, INPUT);          // pause switch

  // counterLed stup
      pinMode(counterLed_0, OUTPUT); // counter LED #0
      pinMode(counterLed_1, OUTPUT); // counter LED #1
      pinMode(counterLed_2, OUTPUT); // counter LED #2
      pinMode(counterLed_3, OUTPUT); // counter LED #3

  // nextPieceLed setup
    pinMode(nextPieceLed_0, OUTPUT); // next piece LED #0
    pinMode(nextPieceLed_1, OUTPUT); // next piece LED #1
    pinMode(nextPieceLed_2, OUTPUT); // next piece LED #2

          setSpeakerPinNum(speakerPin); // speaker pin

       // wait 0.05 seconds to set and initializes LED matrix
  delay(50);
                setDisplayPins(ledMatrixCS, ledMatrixCLK, ledMatrixDIN);
  initDisplay(0x01);

  intro();            // start intro
  setDisplayBrite(0);
  startGame();        // start game

  // initialize timer1
    noInterrupts();           // disable all interrupts
    //TCCR1A = 0;                 // sets Timer/Counter1 Control Register A to zero
    //TCCR1B = 0;                 // sets Timer/Counter1 Control Register B to zero


    // Set timer1_counter to the correct value for our interrupt interval
     timer1_counter = 64560; // preload timer 65536-16MHz/256/64Hz
    //TCNT1 = timer1_counter; // preload timer
    //TCCR1B |= (1 << CS12);  // 256 prescaler
    //TIMSK1 |= (1 << TOIE1); // enable timer overflow interrupt


      __enable_interrupt(); // enable all interrupts

   /* hardware hazard - not implemted yet */
    //attachInterrupt(0, getInput, RISING); // hardware interrups

}

/*
```

```
 * reads the buttons status and sets the flags accordingly
 * it only changes the flag status if there is a change of button status to HIGH
 * to avoid repetitions
 * this function is only used by polling
 */
void getInput() {
   // go rigth witch
       if  (digitalRead(goRigthSwitchPin)  != lastGoRigthSwitchStatus) {
       if (lastGoRigthSwitchStatus == LOW) {
          lastGoRigthSwitchStatus = HIGH;
          goRigthFlag = 0x01;
          return;
       } else {
          lastGoRigthSwitchStatus = LOW;
       }
   }
   // go down switch
       if  (digitalRead(goDownSwitchPin)  != lastGoDownSwitchStatus) {
       if (lastGoDownSwitchStatus == LOW) {
          lastGoDownSwitchStatus = HIGH;
          goDownFlag = 0x01;
          return;
       } else {
          lastGoDownSwitchStatus = LOW;
       }
   }
   // rotate switch
         if  (digitalRead(rotateSwitchPin)  != lastRotateSwitchStatus) {
       if (lastRotateSwitchStatus == LOW) {
          lastRotateSwitchStatus = HIGH;
          rotateFlag = 0x01;
          return;
       } else {
          lastRotateSwitchStatus = LOW;
       }
   }
   // go left switch
         if  (digitalRead(goLeftSwitchPin)  != lastGoLeftSwitchStatus) {
       if (lastGoLeftSwitchStatus == LOW) {
          lastGoLeftSwitchStatus = HIGH;
          goLeftFlag = 0x01;
          return;
       } else {
          lastGoLeftSwitchStatus = LOW;
       }
   }
}

/* hardware hazard - not implemted yet */

/*
 * reads the buttons status and sets the flags accordingly
  * this function is only used by the hardware interrupt
 */
/*void getInput(){

          if(digitalRead(goRigthSwitchPin)== HIGH){
             goRigthFlag = 0x01;
       }
         if(digitalRead(goDownSwitchPin)== HIGH){
             goDownFlag = 0x01;
       }
           if(digitalRead(rotateSwitchPin)== HIGH){
             rotateFlag = 0x01;
       }
          if(digitalRead(goLeftSwitchPin)== HIGH){
             goLeftFlag = 0x01;
       }
}*/

/*
 * the loop event gets the buttons states and interacts
 * with the game according to the button flags
 */
void loop() {
```

```
     /* hardware hazard - not implemted yet
*/
    getInput(); // buttons polling

    // go down check
    if (goDownFlag == 0x01) {
        goDownFlag = 0x00;
        goDown();
    }

    // go rigth check
    if (goRigthFlag == 0x01) {
        goRigthFlag = 0x00;
        goRigth();
    }

    // go left check
    if (goLeftFlag == 0x01) {
        goLeftFlag = 0x00;
        goLeft();
    }

    // rotate check
    if (rotateFlag == 0x01) {
        rotateFlag = 0x00;
        rotate();
    }
}

#pragma vector = TIMER_A0_VECTOR
    __interrupt void timerA0ISR(){
        ISR_update();
      }
```