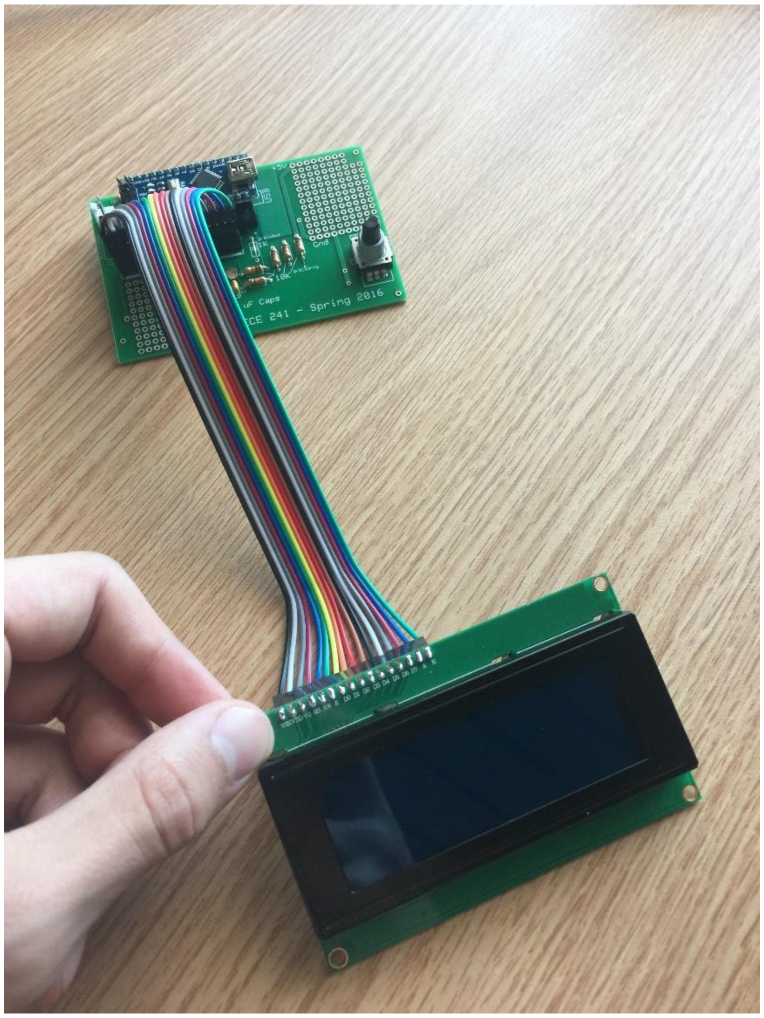
ECE 241 **Final Project Report** **Spring 2018**
Seth Boxberger / Graham Wood **Lab Section: Friday 12:30pm**

Objective: The purpose of this project was to utilize our knowledge of the Liquid Crystal library and LCD functionality through the Arduino Nano board and programming IDE to develop 2 “games” played using the LCD display and the encoder/button knob attached to the board. The games are controlled using the twist functionality of the knob, while menu settings include the integration of a button press to reset or switch between games, those being short and long presses (greater than half second hold) respectively. We developed and manipulated code from our 10th and final lab to have the games include a “scrolling” display with which our games utilize to bring a sense of “play”.

Hardware Used: The only external hardware we purchased was a 4x20 LCD display, which was essential in broadening our field of play for the two created games. A set of wire connections and pins were included to connect to the Arduino board, exact in replication of the state of our LCD displays used all semester (2x16). A bit of soldering was required to attach a set of pins to the LCD, but aside from that, the connections were straightforward. A picture below shows the board connected as a whole.



Software: Since this project was heavily based on programming knowledge, the software was key in making the two games work effectively. To begin with, we used code from Lab 10 which gave us the basics of the scrolling display using a 2-dimensional array which represented each value the display could hold. We had a random number generator put an asterisk in a certain row 20% of the time, and using a couple nested "for" loops, we had the code go through each position and increment values that held an asterisk. This process essentially comes out to look like a "scrolling" motion, and the values moved across the screen indefinitely.

After implementing this code to get a scrolling display, we integrated code to track the encoder position, which represented the player's position, to move up and down the display on the final column based on encoder detent values. These values are key in the games because they essentially keep track of the player's "score" (lives left and asterisks collected). We made sure the player could not move off the screen, and the way the encoder knob moves represents the direction the player moves (since the games are made to be held vertical, moving the knob clockwise moves the player up the screen, and counter-clockwise moves the player down on the screen, to simulate right-left movement).

Button press code was implemented to determine the player's choice to replay the current game or to start an attempt on the other game. The following description describes the process pretty well complete. If the button press is short, reset the current game and display and go again. If the button press is long (i.e. more than half a second hold), switch the game, reset the display and start.

We mention previously that the games hold "score" values – lives left and asterisks collected – which determine the length and successfulness of the gameplay attempt. These were quite easy to implement, just needing a variable to be decremented or incremented based on the type of the game. The life count starts at 3, so once the player hits three asterisks, they are out (when life count = 0). The collect game runs for 60 seconds, which we will talk about later, but similar code was used to increment the "collected" variable once the player "hit" an asterisk. Both of these values are displayed with their respective game, and are reset when switching/resetting the games. For the second game, we had the program run on a time interval of 60 seconds. We had to include code to decrement the code every second, so the game does not run forever, and made sure to have the game run ONLY when that timer is greater than 0. "Score" values are displayed with their respective game, and are reset when their games are.

To make the games more difficult, we were to have them speed up over time. This included keeping a grand total timer variable and keeping track of how long the games had been running. We did this by comparing the timer values to the always-changing millis() variable (how long the Arduino has been running the current program) and having different stages in which we increased the refresh rate of asterisks moving across the screen, essentially making the asterisks move along screen at increasing intervals based on how long the games were running.

Testing: To simplify efforts, we decided to approach this project one game at a time and one use case at a time. Our project can be combined into the efforts of the following test cases.

Case 1: We implemented the scrolling display with the player-tracking code, which gave the screen a dynamic feel and tested to see how well the screen moved with the player moving on top of it (using the encoder knob).

Case 2: Scrolling code was manipulated to decrement the life count variable and display this value on the top-left corner of our display.

Case 3: We integrated end-game code to determine how long the game should run, as well as what it displayed when the game was over. We later implemented a menu screen to help the player determine what to do at this stage.

Debugging: We used trial and error as well as printing values through the serial monitor to track lives and player positions. Once over, we made sure to stop the scrolling display and noticeably change the player icon.

Case 4: We integrated code to make the game speed up over time.

Debugging: Knowing how to properly track the time running and game time. We also tested different intervals of time to see how the screen reacted to the values speeding up.

Game 1 – Asteroid Dodge – is completed.

Case 5: We copied code and manipulated it to instead increment a "collected" variable and implanted the game run state to a length of time, rather than number of lives.

Debugging: We tested different game lengths, and had a bit of trouble getting our code to correctly increment our collected value. We got it all to work eventually. This was the longest part of the project.

Case 6: We tested to make sure our same time intervals worked. After seeing they did, we decided to lower the amount of asterisks printed so that the screen wasn't working too hard and the player didn't feel overwhelmed.

Game 2 – Asteroid Collect – is completed.

Case 7: We decided to go above and beyond with the button press capabilities, and implemented a title screen and game-over screen to be displayed at the VERY beginning (first plug-in / upload) and after each game was over, respectively. The menu simply tells the player what a short press and long press of the button will do at that point in time, and works correctly to reset and switch between games.

Debugging: We had to test a few times to see what exactly we needed to reset. Eventually we came to the conclusion to reset everything and to re-establish all variables to their beginning values.

Conclusion: The LCD game project was created using a 4 x 20 LCD screen and an Arduino Nano. The project implemented two games, one where the user would avoid asterisks and one where the user would pick them up. The software portion of the project was the most time consuming and required modifying code used in previous labs as well implementing new code. A considerable amount of time was also spent on debugging and balancing the game so that the speeds and difficulty were set to a desired rate. The completed program fully implements two games that are well balanced and bug free, that can be played by a user.

CODE FOR FINAL PROJECT:

```
// ----------- Moving Screen Vars -------------------
unsigned long Timer;    //declaring timer variables
unsigned long g1Timer;
unsigned long g2Timer;
unsigned long trackTimer;
int numAst;                   // to keep track of number of asterisks
char Values[4][20] = { // [row] [column]
  {' ', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
  {' ', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
  {' ', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '},
  {' ', ' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '}
}; // this array symbolizes all the

//-------------- Player Vars ----------------
char Player = '<';  // the Player's character
int playerPosition;  // variable to track where the player is
int livesLeft;        // variable to track player's lives

int collected;      // variable to track how many points the player collected
int g2time;         // variable that tracks how much time is left for game 2

// ---------Button Debounce Code------------------
enum state{
  Idle,
  Wait,
  Low
};
state ButtonState;
enum game{
  dodge,
  collect,
  neither
};
game Game;
boolean gameOver;
unsigned long ButtonTimer;

int ButtonNextState(int input){   // inserted the button debounce code
  switch(ButtonState){
    case Idle:
      if (input == LOW){
        ButtonTimer = millis();
        ButtonState = Wait;
        digitalWrite(13, HIGH);
      }
      break;
    case Wait:
      if (input == HIGH){
        ButtonState = Idle;
      }
      else if (millis()-ButtonTimer >= 5){
        ButtonState = Low;
        digitalWrite(13, LOW);
        return 1;
      }
      break;
    case Low:
      if(input == HIGH){
        if (millis() - ButtonTimer < 500){
          ButtonState = Idle;
          return 2;  // Short Press
          break;
        }
        else{
          ButtonState = Idle;
          return 3; // Long Press
          break;
        }

      }
  }
}

//-------------- LCD Library/Setup --------------
#include <LiquidCrystal.h>
LiquidCrystal LCD(11,9,5,6,7,8);

// -------------- Encoder Setup ------------------
volatile int EncoderValue;
volatile int EncoderTracker;

void MonitorA(){
  if (digitalRead(2) == digitalRead(3))      // code to track encoder
position,
    EncoderValue ++;                         // which is used to track player
poition
  else
    EncoderValue --;
}
void MonitorB(){
  if (digitalRead(2) == digitalRead(3))
    EncoderValue --;
  else
    EncoderValue ++;
}
```

```
void resetDisplay(){        // this function sets all values on the
                            // display back to a space, essentially clearing
it
  for(int i = 0; i < 4; i ++){
    for(int j = 0; j < 20; j++)
      Values[i][j] = ' ';
  }
}                                          // this is the increment function
for game 1
void incrementValues1(){                   // loops through the values on the
display
  for (int i = 3; i >= 0; i --){           // and if the value holds an
asterisk, move it
    for(int j = 19; j >= 0; j --){         // to the next space. if the player
also happens to be
      if(j == 19 || j == 18 ){             // in the same row as the asterisk
as it hits the last column,
        if (j == 19){                      // decrement the lives left
variable
          if(Values[i][j] == '*'){
           Values[i][j] = ' ';
          }
        }
        else{
          if(Values[i][j] == '*'){
            if(playerPosition == i)    // checks player position to decrement
life count
              livesLeft --;
            Values[i][j] = ' ';
            Values[i][j+1] = '*'; // increment the asterisk
          }
        }
    }

      else{
        if(Values[i][j] == '*'){  // increment the asterisk
         Values[i][j] = ' ';
         Values[i][j+1] = '*';
        }
      }
    }
  }
}
void incrementValues2(){               // this is the increment function for
game 2
  for (int i = 3; i >= 0; i --){       // loops through the values on the
display
    for(int j = 19; j >= 0; j --){     // and if the value holds an asterisk,
move it
      if(j == 19 || j == 18 ){         // to the next space. if the player
also happens to be
        if (j == 19){                  // in the same row as the asterisk as
it hits the last column,
          if(Values[i][j] == '*'){     // increment the collected variable
           Values[i][j] = ' ';
          }
        }
        else{
          if(Values[i][j] == '*'){
            if(playerPosition == i) // checks player position to increment
collected variable
              collected ++;
            Values[i][j] = ' ';
            Values[i][j+1] = '*';   // increment the asterisk
          }


        }
    }

      else{
        if(Values[i][j] == '*'){
         Values[i][j] = ' ';
         Values[i][j+1] = '*';    // increment asterisk
        }
      }
    }
  }
}

void printValues1(){                 // the printvalues1 function loops through
the display values and
  LCD.setCursor(0,0);                // prints what should be there, as well as
printing the life count
  LCD.print(livesLeft);              // in the top left corner
  for(int i = 0; i < 4; i ++){
    for(int j = 0; j < 20; j++){
      if((i + j) != 0)
        LCD.setCursor(j,i);
        LCD.print(Values[i][j]);
    }
  }
}

void printValues2(){                 // similar to printvalues1, printvalues2
loops and prints what should
  LCD.setCursor(0,0);                // be where, along with the game timer on
the top left and the collected
  LCD.print(g2time);                 // number on the bottom left
  for(int i = 0; i < 4; i ++){
    for(int j = 0; j < 20; j++){
      if(i == 0 && j >= 2)
        LCD.setCursor(j,i);
      else if (i == 3 && j >= 2)
        LCD.setCursor(j,i);
        LCD.print(Values[i][j]);

    }
  }
  LCD.setCursor(0,3);
  LCD.print(collected);
}
```

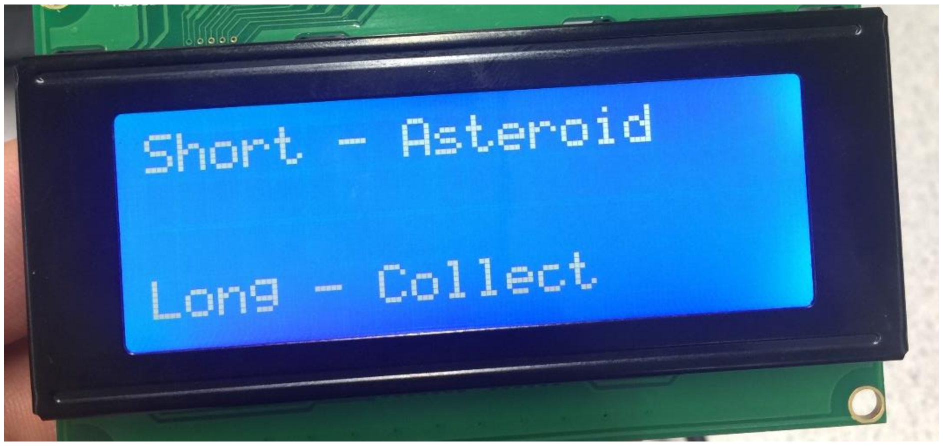
```
void trackPlayer(){                              // the trackplayer function does
exactly that, using the encoder
  if(EncoderValue - EncoderTracker >= 4){   // tracking code to determine
where the player is, and not allowing the
   playerPosition --;                            // player to move off the screen
    if (playerPosition < 0)
      playerPosition = 0;

   EncoderTracker += 4;
  }
  else if ((EncoderValue - EncoderTracker) <= -4){

    playerPosition ++;
   if (playerPosition > 3)
      playerPosition = 3;

    EncoderTracker -= 4;
  }
  LCD.setCursor(19, playerPosition);  // prints the plaer in the last column
of the display,
  LCD.print(Player);                   // at the corresponding row value as
determined by the enocder position
}

// ---------------------- Game Stages ----------------------
/* The following code is all very similar to each other. as all it is doing
 *  is running the different "phases" of the game based on how long the game
has been
 *  running and speeding it up over time.
 */

void g1Stage1(){
  if (millis()-Timer >= 500){          //if 500 millisec have passed,
    incrementValues1();
      for(int k = 0; k < 4; k ++){
        if(random(0,100) < 20){
          if(numAst == 0){              // this if statement makes sure there is
only 1 asterisk printed per column
            Values[k][0] = '*';
            numAst ++;
          }
        }
      }
   printValues1();                      // print the values and
   numAst = 0;                         // reset the asterisk count per colum to
0
   Timer += 500;//update timer
  }
}

void g2Stage1(){
  if (millis()-Timer >= 500){            //if 500 millisec have passed,
    incrementValues2();
      for(int k = 0; k < 4; k ++){
        if(random(0,100) < 10){
          if(numAst == 0){               // this if statement makes sure there
is only 1 asterisk printed per column
```

```
            Values[k][0] = '*';
            numAst ++;
          }
        }
      }
   printValues2();                    // print the values and
   numAst = 0;                        // reset the asterisk count per colum to
0
   Timer += 500;//update timer
  }
}

void g1Stage2(){
  if (millis()-Timer >= 350){            //if 350 millisec have passed,
    incrementValues1();
      for(int k = 0; k < 4; k ++){
        if(random(0,100) < 20){
          if(numAst == 0){               // this if statement makes sure there
is only 1 asterisk printed per column
            Values[k][0] = '*';
            numAst ++;
          }
        }
      }
   printValues1();                     // print the values and
   numAst = 0;                         // reset the asterisk count per colum to
0
   Timer += 350;//update timer
  }
}

void g2Stage2(){
  if (millis()-Timer >= 350){          //if 350 millisec have passed,
    incrementValues2();
      for(int k = 0; k < 4; k ++){
        if(random(0,100) < 10){
          if(numAst == 0){            // this if statement makes sure there is
only 1 asterisk printed per column
            Values[k][0] = '*';
            numAst ++;
          }
        }
      }
   printValues2();                   // print the values and
   numAst = 0;                       // reset the asterisk count per colum to 0
   Timer += 350;//update timer
  }
}

void g1Stage3(){
  if (millis()-Timer >= 200){            //if 200 millisec have passed,
    incrementValues1();
      for(int k = 0; k < 4; k ++){
        if(random(0,100) < 20){
          if(numAst == 0){               // this if statement makes sure there
is only 1 asterisk printed per column
            Values[k][0] = '*';
            numAst ++;
          }
        }
      }
   printValues1();                    // print the values and
   numAst = 0;                        // reset the asterisk count per colum to 0
   Timer += 200;//update timer
  }
}

void g2Stage3(){
  if (millis()-Timer >= 200){         //if 200 millisec have passed,
    incrementValues2();
      for(int k = 0; k < 4; k ++){
        if(random(0,100) < 10){
          if(numAst == 0){          // this if statement makes sure there is
only 1 asterisk printed per column
            Values[k][0] = '*';
            numAst ++;
          }
        }
      }
   printValues2();                    // print the values and
   numAst = 0;                       // reset the asterisk count per colum to 0
   Timer += 200;//update timer
  }
}

void g1Stage4(){
  if (millis()-Timer >= 100){           //if 100 millisec have passed,
    incrementValues1();
      for(int k = 0; k < 4; k ++){
        if(random(0,100) < 20){
          if(numAst == 0){          // this if statement makes sure there is
only 1 asterisk printed per column
            Values[k][0] = '*';
            numAst ++;
          }
        }
      }
   printValues1();                    // print the values and
   numAst = 0;                       // reset the asterisk count per colum to 0
   Timer += 100;//update timer
  }
}
void g2Stage4(){
  if (millis()-Timer >= 100){         //if 100 millisec have passed,
    incrementValues2();
      for(int k = 0; k < 4; k ++){
        if(random(0,100) < 10){
          if(numAst == 0){            // this if statement makes sure there is
only 1 asterisk printed per column
            Values[k][0] = '*';
            numAst ++;
          }
        }
      }
   printValues2();                  // print the values and
   numAst = 0;                       // reset the asterisk count per colum to 0
   Timer += 100;//update timer
  }
}

void makeChoice(){       // determine choice based on button press
    switch (ButtonNextState(digitalRead(4))){
    case 2:
      if(Game == dodge){           // if  game is dodge and short press, reset
game 1
        resetGame1();
        resetDisplay();
      }
      else if (Game == collect){  // if game is collect and short press,
reset game 2
        resetGame2();
        resetDisplay();
      }
      else if(Game == neither){   // for the beginning, if short, start game
1
        Game = dodge;
        resetDisplay();
        resetGame1();
      }
      break;
    case 3:
      if(Game == dodge){            // if  game is dodge and long press,
switch to game 2
        resetDisplay();
        resetGame2();
        Game = collect;
      }
      else if (Game == collect){  // if game is collect and long press,
switch to game 1
        resetDisplay();
        resetGame1();
        Game = dodge;
      }
      else if (Game == neither){  // for the beginning, if long, start game 2
        Game = collect;
        resetDisplay();
        resetGame2();
      }
      break;
  }
}

void runGame1(){  //  this function tracks the time and runs the game while
the life count
                  // is greater than 0. once done, the player turns into an
"X" on the screen
  while(livesLeft != 0){
        trackPlayer();
        if((millis() - g1Timer) < 10000){
          g1Stage1();
        }
        else if ((millis() - g1Timer) < 15000){
          g1Stage2();
        }
        else if ((millis() - g1Timer) < 30000){
          g1Stage3();
        }
        else if((millis() - g1Timer) > 30000){
          g1Stage4();
        }
      }
      LCD.setCursor(19, playerPosition);
      LCD.print('X');
}

void resetGame1(){  // clears the display, resets the timers, and sets life
count back to 3
  LCD.clear();
  g1Timer = millis();
  Timer = millis();
  resetDisplay();
  livesLeft = 3;
}

void resetGame2(){  // clears the display, resets the timers, and sets
collected count to 0
  LCD.clear();
  Timer = millis();
  trackTimer = millis();
  g2Timer = millis();
  resetDisplay();
  g2time = 60;
  collected = 0;
}

void runGame2(){ // runs the second game while the time interval hasnt hit 0.
once it does, it stops
  while(g2time >= 0){
        if(trackTimer - millis() >= 1000){  // decrements the g2time
variable, which determines how long the game
          g2time --;                         // has left to run
          trackTimer += 1000;
        }
        trackPlayer();
        if((millis() - g2Timer) < 10000){
          g2Stage1();
        }
        else if ((millis() - g2Timer) < 20000){
          g2Stage2();
        }
        else if ((millis() - g2Timer) < 40000){
          g2Stage3();
        }
        else if((millis() - g2Timer) > 40000){
          g2Stage4();
        }
  }
}
```

```
void setup() { // initializes variables, sets up ISR's
  pinMode(2, INPUT);
  pinMode(3, INPUT);
  attachInterrupt(0, MonitorA, CHANGE);
  attachInterrupt(1, MonitorB, CHANGE);
  EncoderTracker = EncoderValue;

  Timer = millis();      // set up timer
  ButtonTimer = millis();
  g1Timer = millis();
  g2Timer = millis();
  trackTimer = millis();
  LCD.begin(20,4);  // set up the serial port (20,4)
  LCD.setCursor(0,0);
  numAst = 0;
  playerPosition = 2;
  livesLeft = 3;
  collected = 0;
  Game = neither;
  g2time = 60;
  gameOver = false;

}

void loop() {
  if(Game == neither){              // at the very beginning only, displays a
menu telling the user what they can
    LCD.setCursor(0,0);             // or should do to start a game
    LCD.print("Short - Asteroid");
    LCD.setCursor(0,3);
    LCD.print("Long - Collect");
    makeChoice();                   // reads in the user's choice based on
the button press
  }


  if(Game == dodge && livesLeft !=0){  // runs game 1
    runGame1();
    gameOver = true;  // display game over screen
  }

  makeChoice(); // have the player choose what they want to do


  if (Game == collect){  // runs game 2
     runGame2();
     gameOver = true;  // display game over screen
  }

  if(gameOver){                       // this is the game over screen,
prompting the player to either reset the current
    LCD.setCursor(0,0);               // game or to hold a long press to
switch to the other game
    LCD.print("Short - Reset");
    LCD.setCursor(0,3);
    LCD.print("Long - Switch Game");
    makeChoice();                     // reads in the user's choice based on
the button press
    gameOver = false;                 // 'turns off' the game over display
  }
}
```

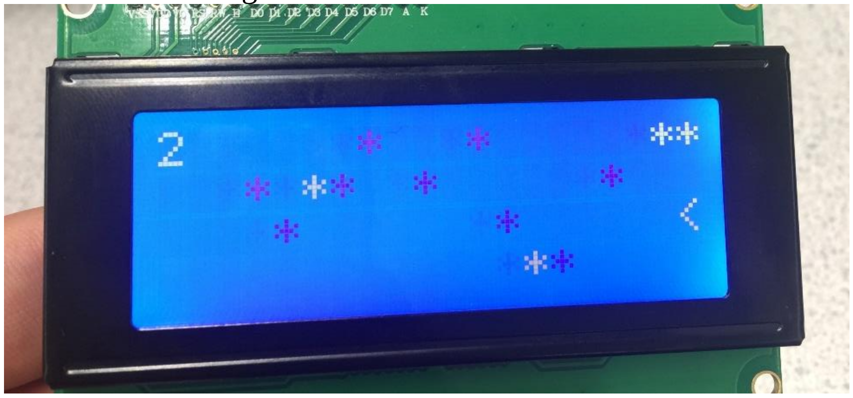
<u>VISUALIZATIONS OF PROGRAM:</u>

Title Menu:



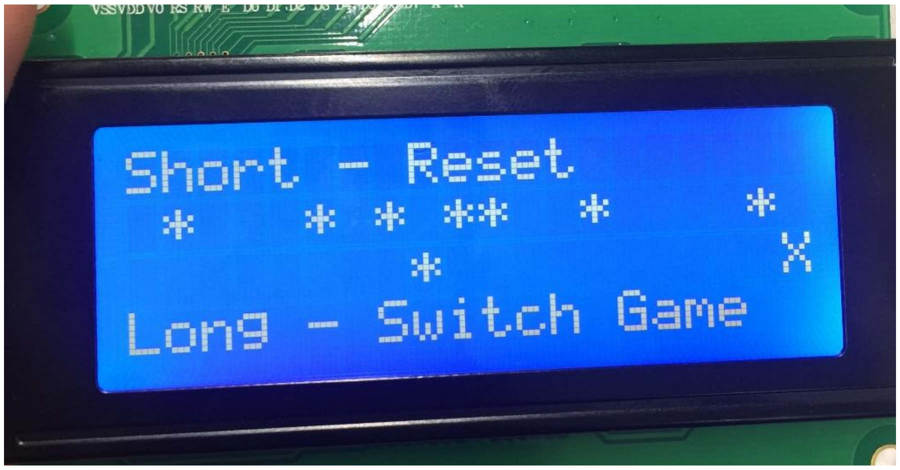
```
Short - Asteroid

Long - Collect
```

Mid-Asteroid Dodge:



```
2       *   *       **
   * **   *      *
    *       *       <
              **
```

Game-Over:



```
Short - Reset
 *    * * **  *    *
         *         X
Long - Switch Game
```

Mid- Asteroid Collect: