Project Technical Report

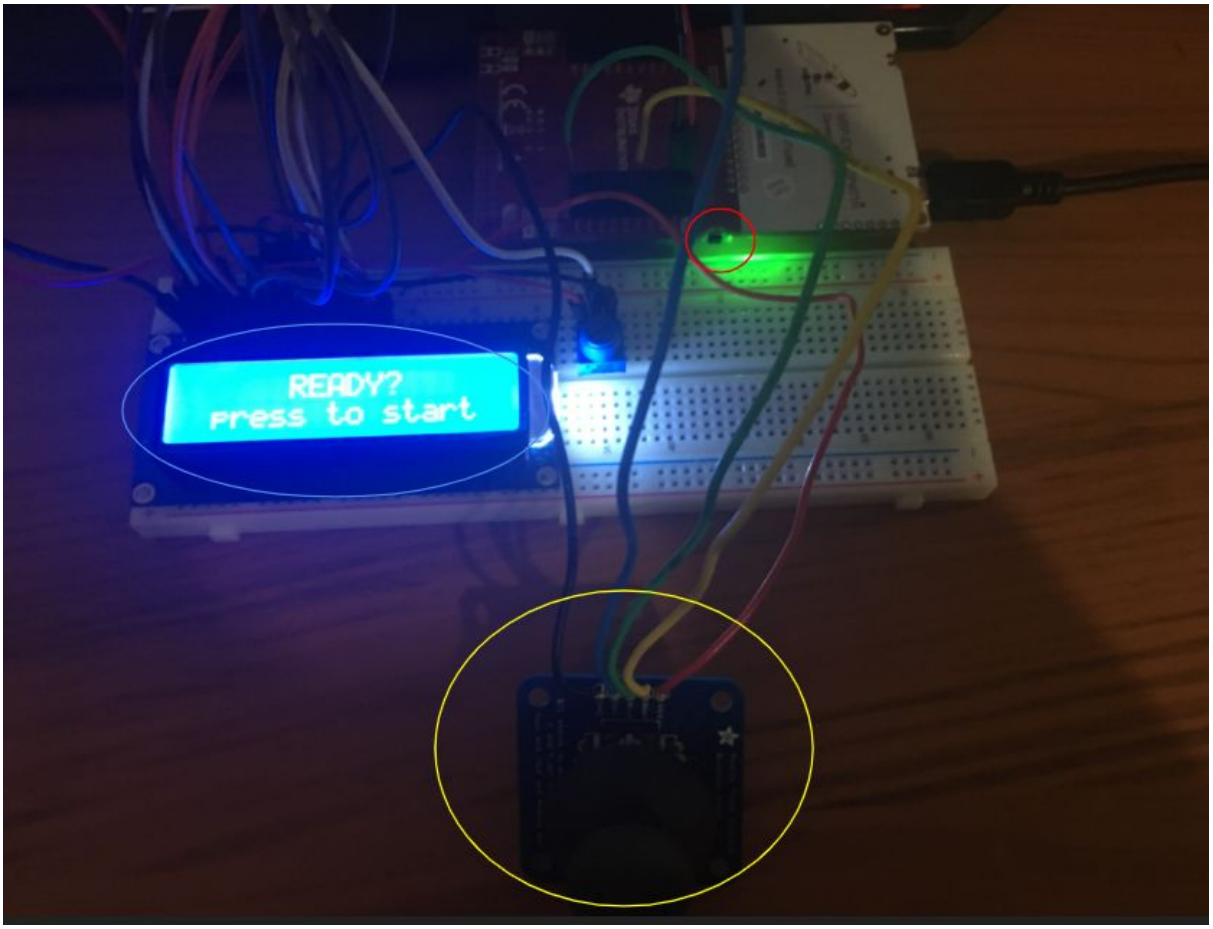
Poject Summary:

The system is an embedded gaming console. The console functions similar to an arcade device in which only power is needed for the system to function, and the game is built into to the processing chip of the system. The player will interact with a joystick and two bottoms during gameplay.

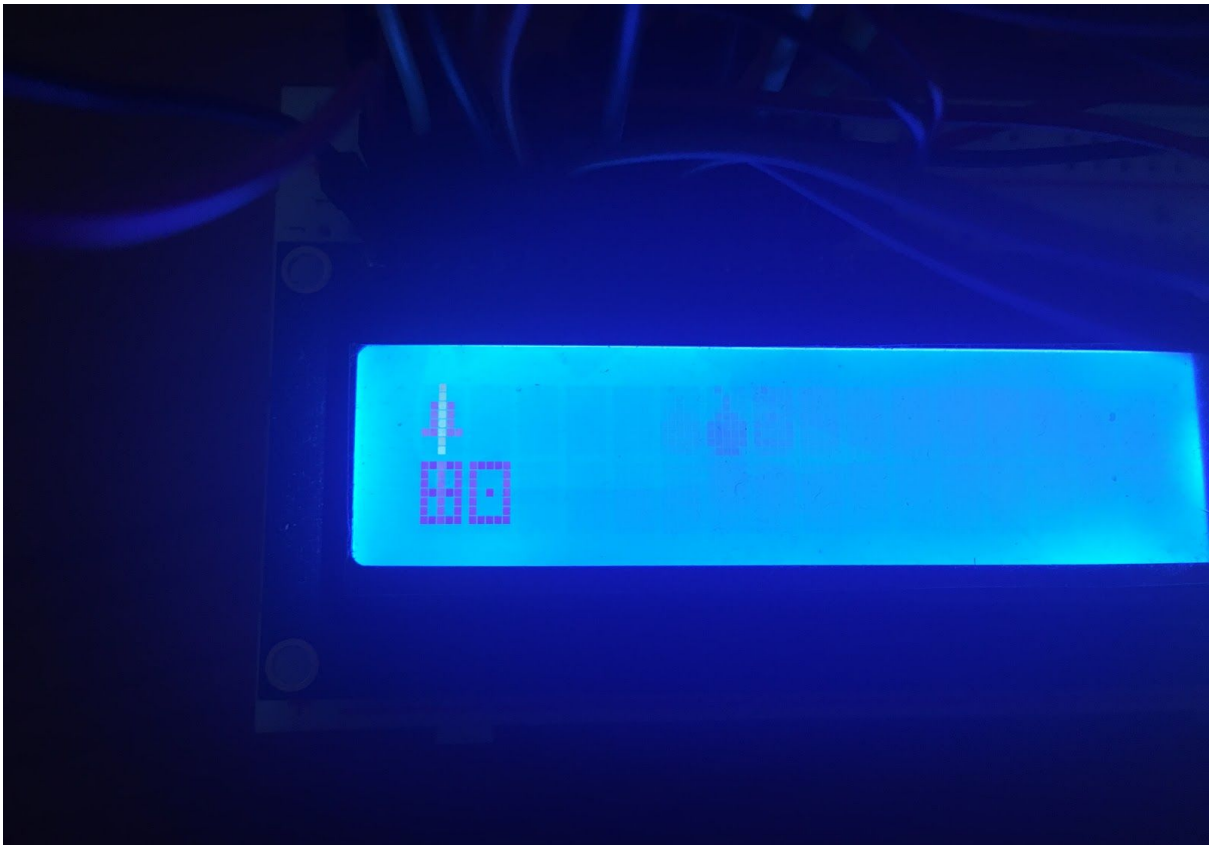
System Functionality:

The system presents a side scroller game in which the player character (the sword) has to survive waves of incoming obstacles (the shields). There are two types of obstacles, type 0 (the shield with a dot in the middle) moves leftwards one block at a time where type 1 (the shield with a cross in the middle) moves leftwards two blocks at a time. When the system is powered, it will start in the starting screen, in which the player will be prompted to press the PUSH1 to start the game. The player character initially start at the left of the screen and the obstacles come out from the right. The movement of the player character is controlled by the attached joystick, where moving the controller up and down will cause the player character to switch rows (the y-axis) appropriate to the input and moving the joystick left and right will case the player character to switch columns (the x-axis) appropriate to the input. Scores increase when an obstacle passes the player character without triggering an game ending event, or when the player character collides with a bonus object. As the score increases, the player will be presented with faster and more frequent obstacles, up to a point where progression will be near impossible. A bonus object is an object that spawns periodically, in the shape of a round fruit. When the bonus object is collected (collides with the player character), in addition to gaining points, the player also gains the ability to fire a projectile (a slash) by pressing PUSH2 (such ability does stack), that travels from the position for the player character all the way towards the right of the screen. The projectile will delete all obstacles that it comes in contact with. A game ending event is triggered when an obstacle comes into contact with the player characters. When this particular event happens, a game over screen will appear on the screen displaying the character “YOU DIED” along with the amount of points the player has gained during gameplay. In which the player can press PUSH1 to go back to the starting screen.

The game starting screen, showing PUSH1 (red), the screen (blue) and the joystick (yellow)



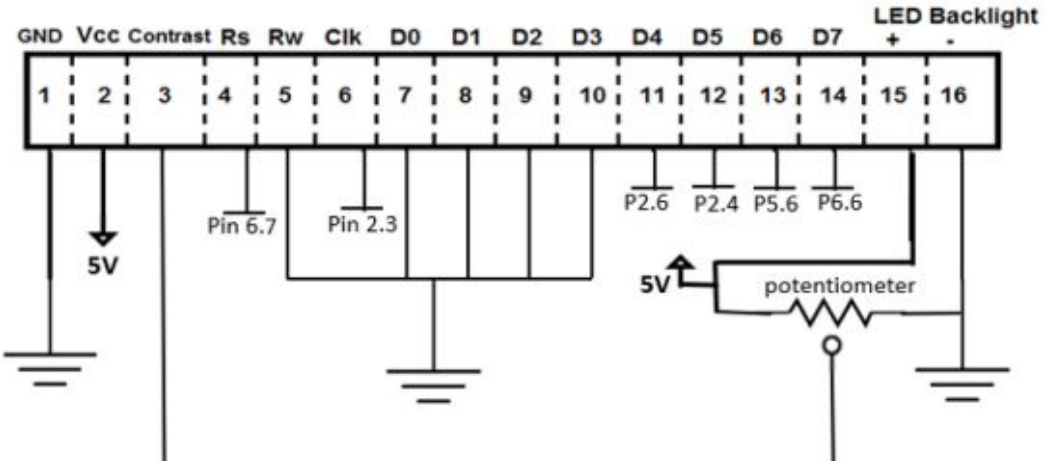
During game play. Showing player character (top left), type 1 obstacle (bot left), type 0 obstacle (bot right) and the bonus object (top right).



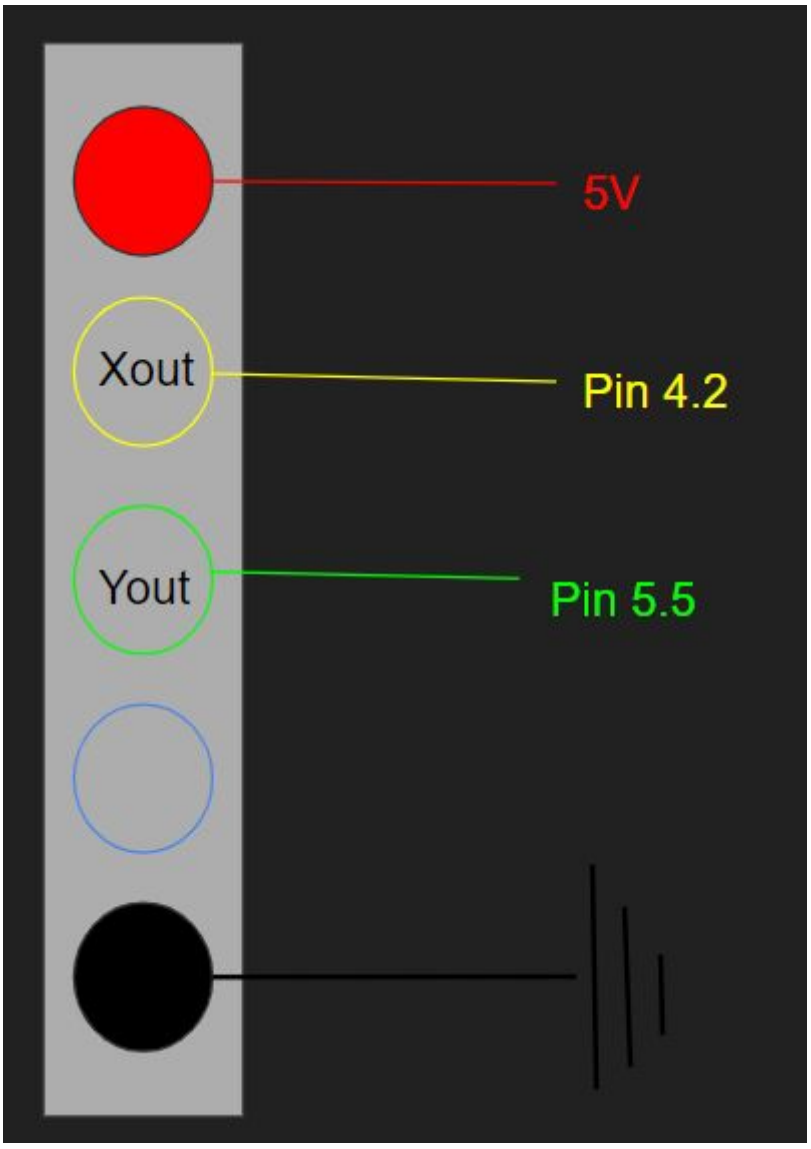
Game over screen, showing “YOU DIED” (top) and the pointe gained during gameplay (bot)

Circuit Design:

Below is the circuit diagram for the LCD screen. D4-D7 are used by the microcontroller to spend information on what to display the screen. The data a being received on each clock pulse by the controller (pin 2.3). Rs is used by the LCD screen to determine what mode the received data should be, either in instruction or character. Example of instruction mode data can be “move cursor” while example of character mode data can be “0” in which the LCD will print a “0” to the cursor location. The potentiometer is used to adjust the contrast of the screen.



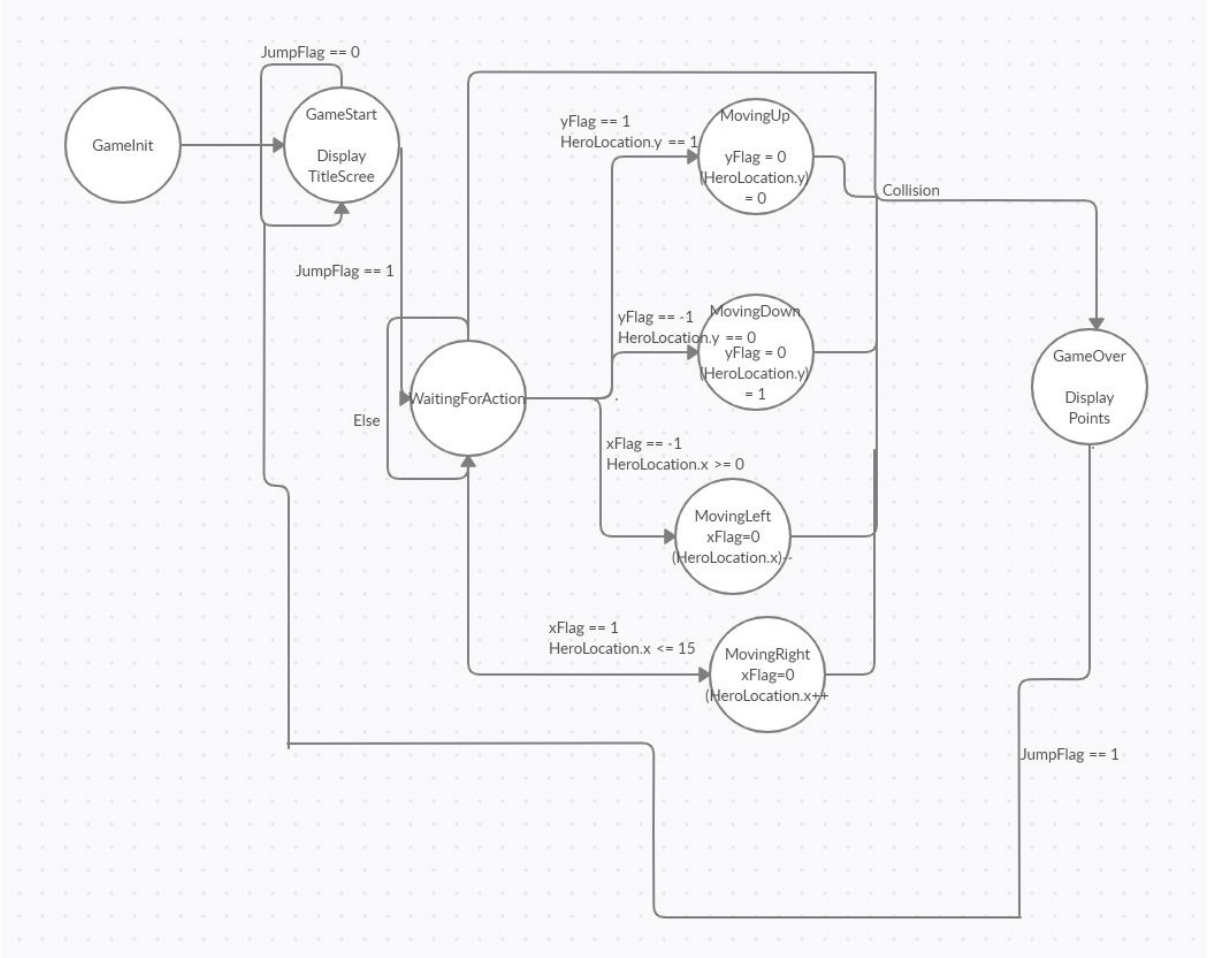
Below is the connection diagram for the joystick. Both Xout and Yout needed to be connected with pins that has analog to digital conversion, since the input is non-binary. The blue pin is the select pin, which is not used in the system due to its functionality being replaced by the buttons built-in on the board. The two buttons (PUSH and PUSH2) will trigger an interrupt service routine when pressed since the two signals should be processed with priority. An interrupt service routine is when the system pauses all current processes to execute the code within the interrupt service routine.



Embedded Control Design:

The following is the state machine used to control the system. Several flags were used to help the system to decide which action it shall take next. When PUSH1

is pressed, jumpFlag will be set to one. Allowing the game to start or restart (depending on current state). The jumpFlag is constantly being reset to avoid the chance that PUSH1 is being pressed outside of its intended states. X and Y flags are set using the X and Y analog inputs of the joystick. When the user moves the joystick in a certain direction pass a threshold, the respective flags will be set in order for the screen accurately reflect user inputs. The joystick inputs are only being read if the game has started, hence the necessity of the waitingForAction state. It also helps eliminate the “busy waiting” problem, in which the processor constantly checks for user input despite there might be none. During each movement states, after the position of the player character has been changed appropriately, the respective flag needs to be reset inorder to prevent the player character from moving in one direction indefinitely when no input has been read.



Enhancements:

Multiple obstacles are implemented by initializing a new struct, containing information such as the position of the obstacle, if the obstacle is active, what type of obstacle is it and if it has been checked for scores or not (this exact struct is also used to implement the bonus and the projectile, more on those later). Then an array of obstacles can be initiated with the size of an array being the absolute maximum amount of obstacles on the screen. When the game starts, numerous obstacles will be activated over time, keeping track of the amount of active obstacles. The game will stop making obstacles active after the current active obstacles equals with the current maximum obstacles. After that, the code checks for all active obstacles and increments their position appropriately, and deletes them when they either gone out of the left of the screen, or comes in contact with a projectile. The current maximum obstacles will be incremented over time as score progresses, up until it has the same value as the absolute maximum amount of obstacles.

The current maximum amount of obstacles and the speed of the obstacles are incremented under the same logic: two integer are initialized, one incrementer and one threshold. The incrementer gets incremented when the player gets a score, and when the incrementer gets pass the threshold, the current max and the speed gets incremented. After that, the incrementer gets reset and the threshold is also reset to a random number, so that the time between each time the game gets harder is different.

The bonus object is implemented using rather similar logic as above. On each cycle (certain amount of time), a variable is being incremented a random amount, and when the variable surpasses a certain fixed threshold, a bonus object is being created, resetting the incrementer. Along side that, on every clock cycle, the code checks for active bonus objects and moves it across the screen, and deleting it when it either exits the left side of the screen, or comes in constant with the player, in which case two points will be granted along with the ability to fire the projectile.

The projectile is implemented by having an interrupt reading the PUSH2 input, and setting a flag when it is pressed. Then on each clock cycle, the code checks for if the button is pressed, and if the player has resources available to fire the projectile. If both are true, a projectile is created in front of the player character. Along side with detecting button presses and creating the projectile, the code also looks for active projecties (max 1) and moves it towards the right of the screen.

Appendix:

All the code for the project are attached for detail

```
#include <LiquidCrystal.h>
#include <OneMsTaskTimer.h>

int p = 0;

xy OldBonusPosition;

void setupBonus()
{
  bonus.position.x = 16;
  bonus.position.y = 0;
  OldBonusPosition = bonus.position;
  Serial.begin(9600);
}

void loopBonus()
{

  while(BonusThreadFlag == 0)
  {
    delay(10);
  }
  BonusThreadFlag = 0; //bonus is on a faster clock compared to the obsticals, means it will move a lot faster than the obsticals
  //Serial.print("(Bonus Thread Wokring) ");

  if(PAS!=gameOver && PAS!=Gamestart && PAS!=GameInit) // if the game is not over and not at the starting screen
  {
    srand(millis());
    p = (rand()%200);

    if(PAS!=gameOver && PAS!=Gamestart)
    {
      if(count >= 10000)  //designed to create a bonus every 10 seconds (on average);
      {
        //Serial.print("Creating bonus");

        createBonus();
        count = 0;

        //Serial.print("Finish Creating Bonus");
      }
    }

    if(bonus.active == 1)
    {
      advanceBonus(); //move the bonus across the screen
      deleteBonus(); //deleting the bonus when it is off the screen
    }

    if(HeroLocation.x==bonus.position.x && HeroLocation.y==bonus.position.y) //collision
    {
      //Serial.print("Bonus Collected");

      nukeCount++;  //inrement bonus count
      points = points + 2; //bonus also awards points, two of them
      delaycnt = delaycnt + 2; //since it awards point, it should also reduce the delay

      lcd.setCursor((HeroLocation.x+1), HeroLocation.y); //indicate that the bonus has been collected
      lcd.print(nukeCount); //this is fine, I wanted the nuke count of flash instead of permenant but this is fine
      delay(100);
      lcd.print(" ");

      eraseBonus();
      bonus.active = 0; //make the bonus inactive
      bonus.position.x = 16; //reset position
```

```
66
67          srand(millis());
68          bonus.position.y = rand()%2; //random the y position of the bonus, between 0 and
            1, for some reason it feels really broken
69        }
70
71        count = count + p;
72      }
73    }
74
75    void createBonus()
76    {
77      if(bonus.active == 0) //only draws when the bonus isn't already on the screen, can
        also check for active in the main loop, but here is fine
78      {
79        bonus.active = 1;
80        lcd.setCursor(bonus.position.x, bonus.position.y);
81        lcd.write(byte(3));
82      }
83    }
84
85    void advanceBonus()
86    {
87      eraseBonus();
88      drawBonus();
89    }
90
91    void eraseBonus()
92    {
93      lcd.setCursor(OldBonusPosition.x, OldBonusPosition.y); //these code are given
94      lcd.print(" ");
95    }
96
97    void drawBonus()
98    {
99        bonus.position.x = bonus.position.x - 1;
100       lcd.setCursor(bonus.position.x, bonus.position.y);
101       lcd.write(byte(3));
102       OldBonusPosition = bonus.position;
103   }
104
105   void deleteBonus()
106   {
107     if(bonus.position.x < -1) //-1 to prevent the object disappearing before making
        collision
108       {
109         eraseBonus();
110
111         bonus.active = 0; //make the bonus inactive
112         bonus.position.x = 16; //reset position
113
114         srand(millis());
115         bonus.position.y = rand()%2; //random the y position of the bonus, between 0 and
            1, for some reason it feels really broken
116
117         Serial.print("Bonus Y: ");
118         Serial.println(bonus.position.y);
119       }
120   }
121
```

```
// include the library code:
#include <LiquidCrystal.h>
#include <OneMsTaskTimer.h>

int points = 0;
int pointsx = 15;

OneMsTaskTimer_t timerTask = {100, playActionTimerISR, 0, 0};

typedef struct xy_struct
{
  int x;
  int y;
}xy;

typedef struct obstical
{
  xy position;
  bool active = 0; //check if the obstical is active
  int type = 0; //type of the obstical
  int checked = 0; //if the obstical has already been checked
} obstical;

obstical bonus; //reusing the obstical struct, the "type" memeber is probably not going to be used
obstical nuke;

int nukeCount = 0;
int count = 0; //incrementer
int delaycnt = 0; //incrementer
int shieldsInUse = 0; //amount of obsticals in use
int maxShields = 2; //max amount of obsticals
int deelaay = 1000; //initial delay between each obstical appearing

byte hero[8] =
{
  B00100,
  B00100,
  B01110,
  B01110,
  B01110,
  B11111,
  B00100,
  B00100,
};

byte sticc[8] =
{
  B11111,
  B10001,
  B10001,
  B10101,
  B10001,
  B10001,
  B11111,
  B00000, //all obsticals are floating since it looks better
};

byte slash[8] =
{
  B11000,
  B00100,
  B00010,
  B00001,
  B00001,
  B00010,
  B00100,
  B11000,
};
```

```
69
70  byte RoundLookingThing[8] =
71  {
72    B01000,
73    B00100,
74    B00100,
75    B01110,
76    B11111,
77    B11111,
78    B11111,
79    B01110,
80  };
81
82  byte special[8] =
83  {
84    B11111,
85    B10101,
86    B10101,
87    B11111,
88    B10101,
89    B10101,
90    B11111,
91    B00000,
92  };
93
94  int const obstcount = 10;
95  obstical obsticals[obstcount]; //a bunch of obsticals
96  obstical oldObsticals[obstcount];
97
98  xy HeroLocation;
99  xy OldLocation;
100 xy sticcLocation = {15,1};
101 xy oldSticcLocation;
102
103 enum PlayerActionStates{GameInit, Gamestart, WaitingForAction, MoveForwards,
    MoveBackwards, MoveUp, MoveDown, gameOver};
104 PlayerActionStates PAS;
105
106 bool PlayerActionFlag = 0;
107 bool BonusThreadFlag = 0;
108 bool NukeThreadFlag = 0;
109 bool ScreenThreadFlag = 0;
110
111 // initialize the library with the numbers of the interface pins
112 LiquidCrystal lcd(P6_7, P2_3, P2_6, P2_4, P5_6, P6_6);
113
114 void setup()
115 {
116   lcd.begin(16, 2);
117   lcd.createChar(0, hero);
118   lcd.createChar(1, sticc);
119   lcd.createChar(2, special);
120   lcd.createChar(3, RoundLookingThing);
121   lcd.createChar(4, slash);
122   Serial.begin(9600);
123
124   OneMsTaskTimer::add(&timerTask);
125   OneMsTaskTimer::start();
126
127   for(int i=0; i<obstcount; i++)
128   {
129     obsticals[i].position.x = 16;
130     obsticals[i].active = 0;
131     obsticals[i].type = 0;
132     obsticals[i].checked = 0;
133   }
134
135   for(int i=0; i<(int)(obstcount/2); i++) //half of the obsticals will be on top
136   {
```

```
137       obsticals[i].position.y = 0;
138     }
139
140     for(int i=(int)(obstcount/2); i<obstcount; i++) //half of the obsticals will be on
        the bottom, screw random numbers
141     {
142       obsticals[i].position.y = 1;
143     }
144
145     for(int i=0; i<(int)(obstcount/2); i=i+2) //half of the obsticals on top will be of
        type 0
146     {
147       obsticals[i].type = 0;
148     }
149
150     for(int i=(int)(obstcount/2); i<obstcount; i=i+2) //half of the obsticals on top will
        be of type 1
151     {
152       obsticals[i].type = 1;
153     }
154
155     obsticals[0].type = 0;
156
157   }
158
159   void loop() {
160     delay(100);
161     //Serial.print("(LCD Wokring)");
162   }
163
164   void eraseShield(int i)
165   {
166     lcd.setCursor(oldObsticals[i].position.x, oldObsticals[i].position.y); //these code
        are given, just added an input parameter to help navigation
167     lcd.print(" ");
168   }
169
170   void playActionTimerISR()
171   {
172     PlayerActionFlag = 1;
173     BonusThreadFlag = 1;
174     NukeThreadFlag = 1;
175     ScreenThreadFlag = 1;
176   }
177
```

```
#include <LiquidCrystal.h>
#include <OneMsTaskTimer.h>

bool nukeing = 0;
int nukePin = PUSH2;
xy OldNukePosition;

void setupNuke()
{
  pinMode(nukePin, INPUT_PULLUP);
  nuke.active = 0;
  nuke.position.x = 16;
  nuke.position.y = 0;
  OldNukePosition = nuke.position;
  Serial.begin(9600);
  attachInterrupt(digitalPinToInterrupt(nukePin), nukeISR, FALLING);
}

void loopNuke()
{
  while(NukeThreadFlag == 0) //almost everything is on this clock
  {
    delay(10);
  }
  NukeThreadFlag = 0;

  //Serial.print("(Nuke Thread Wokring) ");

  if(PAS!=gameOver && PAS!=Gamestart && PAS!=GameInit) // if the game is not over and not at the starting screen
  {
    if(nukeing == 1 && nukeCount > 0 && nuke.active == 0) //the bottum has beem pressed and the player has resources left
    {
      nukeing = 0;
      nukeCount--;

      createNuke(); //logics here are similar to bonus

      Serial.print("Creating Nuke");
    }
    else if(nukeing == 1 && nukeCount <= 0 && nuke.active == 0)
    {
      Serial.print("Error: Not Enough Nukes");
    }

    if(nuke.active == 1)
    {
      advanceNuke(); //move the nuke across the screen
      deleteShield1(); //detect collision on each step
      deleteNuke(); //deleting the nuke when it is off the screen
    }

    nukeing = 0; //the flag should still be reset even if nothing happens
  }
}

void nukeISR()
{
  Serial.println("ISR - Nuke");
  nukeing = 1;
}


void createNuke()
{
  if(nuke.active == 0) //only runs when the nuke isn't already on the screen
  {
    nuke.position.x = HeroLocation.x + 1; //this might cause timing issues, but I'm too
```

```
      tired to fix it
      nuke.position.y = HeroLocation.y; //Set the nuke to appear one unit infront of the PC
      nuke.active = 1;
      OldNukePosition = nuke.position;

      lcd.setCursor(nuke.position.x, nuke.position.y);
      lcd.write(byte(4));
    }
  }

  void advanceNuke()
  {
    eraseNuke();
    drawNuke();
  }

  void eraseNuke()
  {
    lcd.setCursor(OldNukePosition.x, OldNukePosition.y); //these code are given
    lcd.print(" ");
  }

  void drawNuke()
  {
      nuke.position.x = nuke.position.x + 1; //march towards the right
      lcd.setCursor(nuke.position.x, nuke.position.y);
      lcd.write(byte(4));
      OldNukePosition = nuke.position;
  }

  void deleteNuke()
  {
    if(nuke.position.x < 0  || nuke.position.x > 15) //if the player wants to fire it off
    on the far right for some reason
      {
        eraseNuke();

        nuke.active = 0; //make the nuke inactive
        nuke.position.x = 16; //reset position
      }
  }

  void deleteShield1()
  {
    for(int i=0; i<obstcount; i++) //check every element of the array
    {
      if(((obsticals[i].position.x == nuke.position.x) && (obsticals[i].position.y ==
      nuke.position.y)) || ((obsticals[i].position.x == (nuke.position.x-1)) &&
      (obsticals[i].position.y == nuke.position.y))) //if the obstical meets the nuke,
      the obstical gets deleted
      {
        /*Serial.print("Deactivating element: ");
        Serial.println(i);*/

        eraseShield(i);

        if(obsticals[i].position.x <= 15 && nuke.position.x <= 15) //prevent collision
        off screen from messing with things
        {
          shieldsInUse--;
          Serial.print("Shield Count Updated: ");
          Serial.println(shieldsInUse);
        }

        obsticals[i].active = 0; //make the obstical inactive
        obsticals[i].position.x = 16; //reset position
        obsticals[i].checked = 0;
      }
    }
```

```
131     }
132
```

```
#include <OneMsTaskTimer.h>

int xOutPin = P4_2;
int yOutPin = P5_5;
int xVal;
int yVal;

int selectPin = PUSH1;

int xFlag = 0;
int yFlag = 0;
bool jumpFlag = 0;

int cnt = 0;

void setupPlayerActions()
{
  pinMode(selectPin, INPUT_PULLUP);
  Serial.begin(9600);
  HeroLocation.x = 0;
  HeroLocation.y = 1;

  attachInterrupt(digitalPinToInterrupt(selectPin), jumpISR, FALLING);
}

void loopPlayerActions()
{

  while(PlayerActionFlag == 0)
  {
    delay(10);
  }
  PlayerActionFlag = 0;

  //Serial.print("(PA Thread Wokring) ");


  /*Serial.println(jumpFlag);
  delay(1000);
  Serial.println(jumpFlag);*/

  if(PAS!=gameOver && PAS!=Gamestart && PAS!=GameInit) // detect player movements only if the game is in progress
  {
    xVal = analogRead(xOutPin);
    setXflag(xVal);

    yVal = analogRead(yOutPin);
    setYflag(yVal);
  }

  /*Serial.println(xVal);*/
  //Serial.println(xFlag);

  //Serial.println(yVal);
  //Serial.println(yFlag);

  PlayerStateProgress();

  /*Serial.print("Hero position: ");
  Serial.print(HeroLocation.x);
  Serial.print(" ");
  Serial.println(HeroLocation.y);*/
  /*Serial.print("Current Game State: ");
  Serial.println(PAS);*/

  delay(10);

}
```

```
69
70  void jumpISR()
71  {
72    Serial.println("ISR - Jump");
73    jumpFlag = 1;
74  }
75
76  void setXflag(int xVal)
77  {
78    if(xVal > 730 && xVal < 830)
79    {
80      xFlag = 0;
81    }
82    else if(xVal>900)
83    {
84      xFlag = 1;
85    }
86    else if(xVal<533)
87    {
88      xFlag = -1;
89    }
90  }
91
92
93  void setYflag(int yVal)
94  {
95    if(yVal > 730 && yVal < 830)
96    {
97      yFlag = 0;
98    }
99    else if(yVal>900)
100   {
101     yFlag = 1;
102   }
103   else if(yVal<533)
104   {
105     yFlag = -1;
106   }
107 }
108
109 void PlayerStateProgress()
110 {
111   //switch statements
112   switch(PAS)
113   {
114     case(GameInit):
115       PAS = Gamestart;
116       break;
117     case(Gamestart):
118       if(jumpFlag)
119       {
120         points = 0;
121         jumpFlag = 0;
122         lcd.clear();
123         PAS = WaitingForAction;
124       }
125       break;
126     case(WaitingForAction):
127       //Serial.println("Transition Waiting for Action");
128       if(xFlag == 1 && HeroLocation.x <= 15)
129       {
130         PAS = MoveForwards;
131         //Serial.println("Moving Forwards");
132         break;
133       }
134       else if(xFlag == -1 && HeroLocation.x >= 0)
135       {
136         PAS = MoveBackwards;
137         break;
```

```
        }
        else if(yFlag == -1 && HeroLocation.y == 0)
        {
          Serial.println("Moving Down");
          PAS = MoveDown;
          break;
        }
        else if(yFlag == 1 && HeroLocation.y == 1)
        {
          Serial.println("Moving Up");
          PAS = MoveUp;
          break;
        }
        else
        {
          PAS = WaitingForAction;
          break;
        }
      case(MoveForwards):
          PAS = WaitingForAction;
          break;
      case(MoveBackwards):
          PAS = WaitingForAction;
          break;
      case(MoveDown):
          PAS = WaitingForAction;
          break;
      case(MoveUp):
          PAS = WaitingForAction;
          break;
      default:
          PAS = WaitingForAction;
          break;
      case(gameOver):

          //Serial.println("Game Over");

          if(jumpFlag)
          {
            Serial.println(jumpFlag);
            jumpFlag = 0;
            lcd.clear();
            PAS = Gamestart;
          }
          break;
    }

    //state actions
    switch(PAS)
    {
      case(Gamestart):
        lcd.setCursor(5, 0);
        lcd.print("READY?");
        lcd.setCursor(1, 1);
        lcd.print("press to start");
        HeroLocation.x=0;
        HeroLocation.y=1;
        break;

      case(MoveForwards):
        //Serial.println("Move Forward State");
        xFlag=0;
        (HeroLocation.x) = (HeroLocation.x) + 1;
        break;

      case(MoveBackwards):
        //Serial.println("Move Backward State");
        xFlag=0;
        (HeroLocation.x)--;
```

```
207         break;
208
209       case(MoveUp):
210         yFlag = 0;
211         (HeroLocation.y) = 0;
212         break;
213
214       case(MoveDown):
215         yFlag = 0;
216         (HeroLocation.y) = 1;
217         break;
218
219       case(gameOver):
220         for(int i=0; i<obstcount; i++)
221         {
222           obsticals[i].position.x = 16; //reset every obstical on game over
223           obsticals[i].active = 0;
224           obsticals[i].checked = 0;
225         }
226
227         bonus.position.x = 16; //also reset everything else
228         bonus.position.y = 0;
229         bonus.active = 0;
230         nukeCount = 0;
231         count = 0;
232         shieldsInUse = 0;
233         maxShields = 2;
234         deelaay = 1000;
235
236         //pointsx = 15; //also reset points
237         points = 0;
238         break;
239     }
240
241   }
242
```

```
#include <LiquidCrystal.h>

void setupScreen()
{
  //lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0, 1);
  lcd.write(byte(0));
  Serial.begin(9600);
}

void loopScreen()
{
  while(ScreenThreadFlag == 0)
  {
    delay(10);
  }
  ScreenThreadFlag = 0;

  //Serial.print("(Screen Thread Wokring) ");

  if(PAS!=gameOver && PAS!=Gamestart && PAS!=GameInit) // if the game is not over and not at the starting screen
  {
    jumpFlag = 0;

    eraseHero();
    drawHero();

    //detect collision
    for(int i=0; i<obstcount; i++)
    {
      if(HeroLocation.x==obsticals[i].position.x && HeroLocation.y==obsticals[i].position.y)
      {
        Serial.print("Game Over");

        Serial.print("Hero Position: ");
        Serial.print(HeroLocation.x);
        Serial.print(" ");
        Serial.println(HeroLocation.y);

        Serial.print("Obstical Position: ");
        Serial.print(obsticals[i].position.x);
        Serial.print(" ");
        Serial.println(obsticals[i].position.y);

        gameOva();
      }
    }
  }

  /*Serial.print("Sticc position: ");
  Serial.print(sticcLocation.x);
  Serial.print(" ");
  Serial.println(sticcLocation.y);*/

}

void eraseHero()
{
  lcd.setCursor(OldLocation.x, OldLocation.y);
  lcd.print(" ");
}

void drawHero()
{
  lcd.setCursor(HeroLocation.x, HeroLocation.y);
  lcd.write(byte(0));
```

```
  OldLocation = HeroLocation;
}

void eraseSticc()
{
  lcd.setCursor(oldSticcLocation.x, oldSticcLocation.y);
  lcd.print(" ");//erase the sticc
}

void drawSticc()
{
  srand(millis());
  if(sticcLocation.x < 0)//resetting the sticc
  {
    sticcLocation.x = rand()%10+8;
  }

  //increment the sticc
  sticcLocation.x = sticcLocation.x-1;

  //draw new sticc
  lcd.setCursor(sticcLocation.x, sticcLocation.y);
  lcd.write(byte(1));
  /*Serial.println(sticcLocation.x);
  Serial.println(sticcLocation.y);
  Serial.println("Wrong");*/

  oldSticcLocation = sticcLocation;
}

void gameOva()
{
  PAS = gameOver;

  lcd.clear();
  lcd.setCursor(4, 0);
  lcd.print("YOU DIED");
  lcd.setCursor(4, 1);
  lcd.print("Points: ");
  lcd.print((points-1)); //taking out the points that got counted when the player died

}
```

```
#include <LiquidCrystal.h>
#include <OneMsTaskTimer.h>

int delaytrigger = 3;
int const delaymax = 250; //minimum delay

void setupShield()
{
  //lcd.begin(16, 2);
  Serial.begin(9600);
}

void loopShield()
{

  if(PAS!=gameOver && PAS!=Gamestart && PAS!=GameInit)
  {
    createShield(); //try to create an obstical, designed to work some of the time
    advanceShield(); //increment all obstical positions by the appropriate amount
    //overLap(); //if two obstical over lap, delete the one thas has a lower type value
    deleteShield(); //mark off the obsticals that are no longer on the screen, can be intergrated into other fucntions but I'm not going to
  }

  if(PAS!=gameOver && PAS!=Gamestart && PAS!=GameInit)
  {
    for(int i=0; i<obstcount; i++) //counting points
      {
        if((HeroLocation.x == obsticals[i].position.x && obsticals[i].checked == 0) ||
        (HeroLocation.x > obsticals[i].position.x && obsticals[i].checked == 0))
        {
          points++;
          delaycnt++;

          Serial.print("Points updated: ");
          Serial.println(points);

          obsticals[i].checked = 1; //prevent a sigular obstical to grant multiple points
        }
      }
   }

  if(PAS!=gameOver && PAS!=Gamestart && PAS!=GameInit)
  {
    if(delaycnt >= delaytrigger && deelaay > delaymax)
    {
      if(maxShields < obstcount)
      {
        maxShields = maxShields + 1; //increment the amount of shields that can be in use when reducing delay
        Serial.print("MaxShields updated: ");
        Serial.println(maxShields);
      }

      deelaay = deelaay - (delaycnt * 20); //reducing the delay
      delaycnt = 0;

      srand(millis());
      delaytrigger = rand()%4 + 2;
      Serial.print("Delay reduced: ");
      Serial.println(deelaay);
    }
  }

  delay(deelaay);
}

void overLap() //this functionality is no longer needed, but since I spent so much time writing it, I will keep it here
```

```
66  {
67    for (int i = 0; i<obstcount; i++)
68    {
69      for(int j = obstcount-1; j>i; j--)
70      {
71        if((obsticals[i].position.x == obsticals[j].position.x) &&
          (obsticals[i].position.y == obsticals[j].position.y))
72        {
73
74          Serial.print("i: ");
75          Serial.print(i);
76          Serial.print("j: ");
77          Serial.println(j);
78
79          Serial.print("x[i]: ");
80          Serial.print(obsticals[i].position.x);
81          Serial.print(" y[i]: ");
82          Serial.println(obsticals[i].position.y);
83
84          Serial.print("x[j]: ");
85          Serial.print(obsticals[j].position.x);
86          Serial.print(" y[j]: ");
87          Serial.println(obsticals[j].position.y);
88
89          if(obsticals[i].type < obsticals[j].type)
90          {
91            Serial.print("Overlap detected 1");
92            obsticals[i].position.x = 16;
93            obsticals[i].active = 0;
94
95            lcd.setCursor(oldObsticals[i].position.x, oldObsticals[i].position.y);
96            lcd.print(" ");
97          }
98          else if(obsticals[i].type > obsticals[j].type)
99          {
100           Serial.println("Overlap detected 2");
101           obsticals[j].position.x = 16;
102           obsticals[j].active = 0;
103
104           lcd.setCursor(oldObsticals[i].position.x, oldObsticals[i].position.y);
105           lcd.print(" ");
106         }
107         else if(obsticals[i].type == obsticals[j].type)
108         {
109           Serial.println("Overlap detected 3");
110           obsticals[j].position.x = 16;
111           obsticals[j].active = 0;
112
113           lcd.setCursor(oldObsticals[i].position.x, oldObsticals[i].position.y);
114           lcd.print(" ");
115         }
116       }
117     }
118   }
119 }
120
121 void createShield()
122 {
123   int r;
124   int l;
125   srand(millis());
126   l = rand()%8;
127
128   if(shieldsInUse < maxShields)
129   {
130     for(int i=0; i<l; i++) //generates obsticals at pseudo-random intervals
131     {
132       srand(millis());
133       r = rand()%obstcount; //generates a seed each loop
```

```
134
135         if(obsticals[r].active == 0) //try to find an inactive obstical
136         {
137           obsticals[r].active = 1; //make it active
138
139           /*Serial.print("Activating element: ");
140           Serial.println(r);*/
141
142           lcd.setCursor(obsticals[r].position.x, obsticals[r].position.y); //set curser
              to location
143
144           /*Serial.print("Drawing: x: ");
145           Serial.print(obsticals[r].position.x);
146           Serial.print(" y: ");
147           Serial.print(obsticals[r].position.y);
148           Serial.print(" State: ");
149           Serial.println(obsticals[r].active);*/
150
151           if(obsticals[r].type == 0) //draw the appropriate obstical
152           {
153             lcd.write(byte(1));
154           }
155           else if(obsticals[r].type == 1)
156           {
157             lcd.write(byte(2));
158           }
159           shieldsInUse++;
160
161           Serial.print("Shield Count Updated: ");
162           Serial.println(shieldsInUse);
163           break; //exits the loop when exactly one obstical has been draw to screen
164         }
165         else
166         {
167           //Serial.println("Nothing to be drawn"); //nothing really needs to be here, but
              I'm having it print something anyway
168         }
169       }
170     }
171   }
172
173   void advanceShield()
174   {
175     for (int i = 0; i<obstcount; i++) //go through the entire array
176     {
177       //logics are put in place to ensure that an obstical of a higher type are always on
          top
178       if (obsticals[i].active == 1 && obsticals[i].type == 0) //find active obsticals
          that's of type 0
179       {
180         /*Serial.print("Current Active Elements: ");
181         Serial.println(i);*/
182
183         eraseShield(i); //advance the appropiate obstical
184         drawShield(i);
185       }
186     }
187
188     for (int i = 0; i<obstcount; i++) //go through the entire array
189     {
190       if (obsticals[i].active == 1 && obsticals[i].type == 1) //find active obsticals
          that's of type 1
191       {
192         /*Serial.print("Current Active Elements: ");
193         Serial.println(i);*/
194
195         eraseShield(i); //advance the appropiate obstical
196         drawShield(i);
197       }
```

```
198     }
199   }
200
201   void drawShield(int i)
202   {
203     if(obsticals[i].type == 0)
204     {
205       obsticals[i].position.x = obsticals[i].position.x - 1; //type 0 obsticals march
        left 1 unit at a time
206       lcd.setCursor(obsticals[i].position.x, obsticals[i].position.y);
207       lcd.write(byte(1));
208       //overLap();
209       oldObsticals[i].position = obsticals[i].position; //remember the position
210     }
211     else if(obsticals[i].type == 1) //if else works fine here, but if the type gets
      numerous, switch statements are better
212     {
213       obsticals[i].position.x = obsticals[i].position.x - 2; //type 2 obsticals march
        left 2 unit at a time
214       lcd.setCursor(obsticals[i].position.x, obsticals[i].position.y);
215       lcd.write(byte(2));
216       //overLap();
217       oldObsticals[i].position = obsticals[i].position; //remember the position
218     }
219   }
220
221   void deleteShield()
222   {
223     for(int i=0; i<obstcount; i++) //check every element of the array
224     {
225       if(obsticals[i].position.x < 0 ) //still erase the obstical if it is offscreen, but
        wait for one more clock cycle for collision detection
226       {
227         eraseShield(i);
228       }
229
230       if(obsticals[i].position.x < -1 ) //-1 to prevent the object disappearing before
        making collision
231       {
232         /*Serial.print("Deactivating element: ");
233         Serial.println(i);*/
234
235         obsticals[i].active = 0; //make the obstical inactive
236         obsticals[i].position.x = 16; //reset position
237         obsticals[i].checked = 0;
238
239         shieldsInUse--;
240         Serial.print("Shield Count Updated: ");
241         Serial.println(shieldsInUse);
242       }
243     }
244   }
245
```