Monster Hunter

Custom Project Final Report

Joseph DiCarlantonio

July 25, 2019

Table of Contents

Introduction....................3

  User Guide....................3

Hardware....................4

  Parts List....................4

  Pinout....................4

Software....................5

  State Machines....................5

  Code (Github Link)....................8

Used Technologies/Components....................8

  Complexities/Build-upons....................8

YouTube Link....................8

Known Bugs and Shortcomings....................9

Future Work....................9

Introduction

Monster Hunter is a single player side scrolling game played on a 16x2 LCD screen. The player controls a character which shoots at oncoming monsters and saves friendly non-player characters (NPC) by ducking out of their way as they are running for their lives. The objective is to shoot as many monsters as possible and to ensure each friendly NPC is saved. The player' s score is determined by how many monsters are killed. Moreover, failure to save a friendly NPC will result in a one point deduction from the player's total score.

User Guide

Objective: Kill as many monsters as possible while allowing friendlies to pass by.

Controls: There are a total of three buttons. Two to control the player and one to reset/start.

How to Play:

The top right button is used to fire bullets at oncoming entities. Shooting a monster once kills it and increments your score. However, shooting a friendly will have no effect.

The bottom right button is used to duck. You must duck to allow the friendly NPCs to pass. Failure to duck for friendlies will result in a loss of points. You may also duck from monsters if you feel you will not be able to shoot it. No points will be awarded in this scenario.

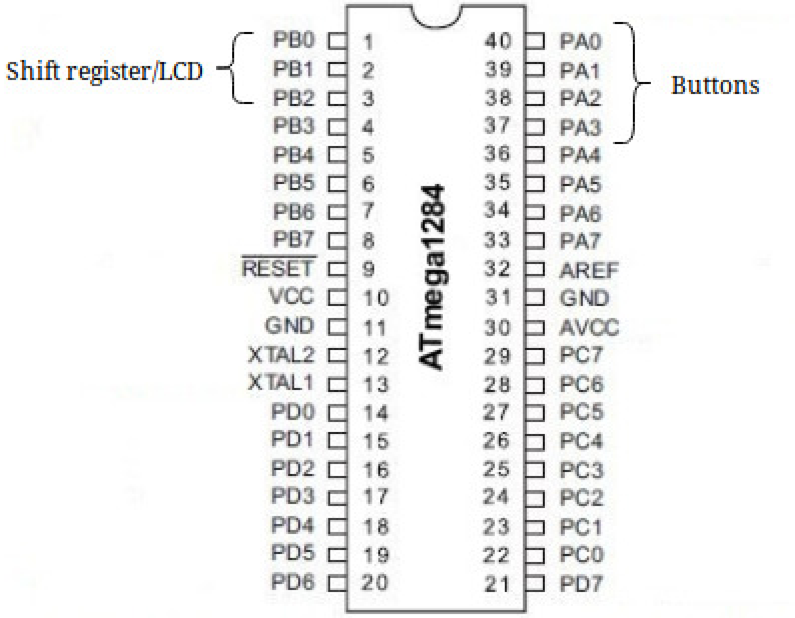
The remaining button is used as a soft reset and start button. Pressing it during the menu will start the game. Pressing it at any time during the game or on the game over display will take you back to the menu.

Hardware

Parts List

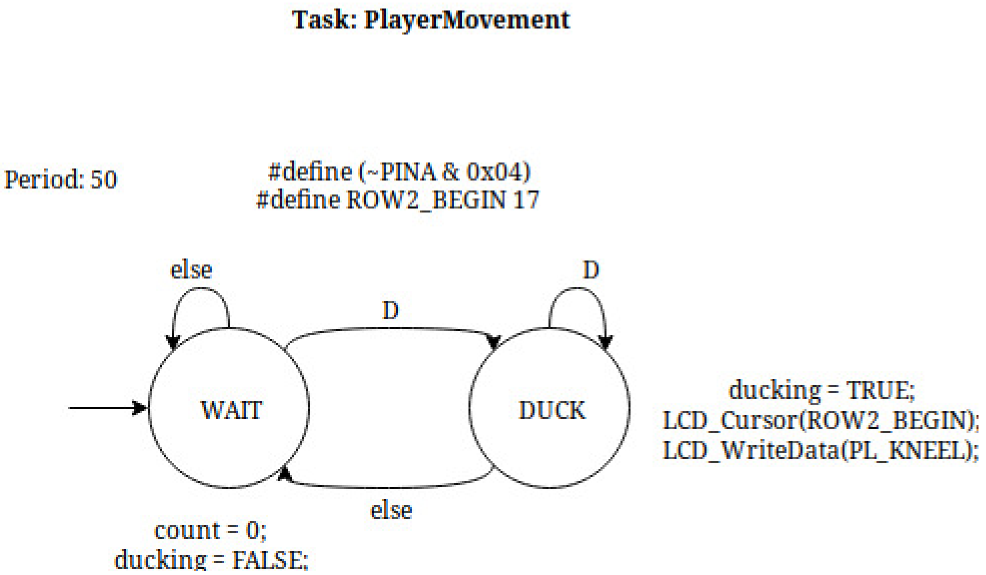
- ATMega1284 Microcontroller
- 16x2 LCD Display
- TI SN74HC595N 8-bit Shift Register
- Buttons

Pinout



Software

State Machines



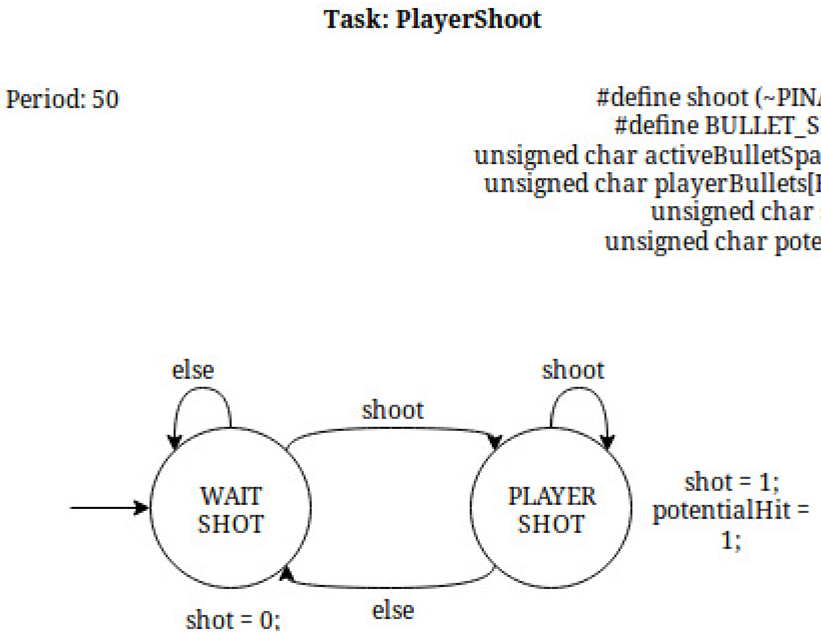
Task: PlayerMovement

Period: 50

```
#define (~PINA & 0x04)
#define ROW2_BEGIN 17
```

```
ducking = TRUE;
LCD_Cursor(ROW2_BEGIN);
LCD_WriteData(PL_KNEEL);
```

```
count = 0;
ducking = FALSE;
```



Task: PlayerShoot

Period: 50

```
#define shoot (~PINA & 0x02)
#define BULLET_SPACE 15
unsigned char activeBulletSpace = BULLET_SPACE;
unsigned char playerBullets[BULLET_SPACE + 1];
unsigned char shot;
unsigned char potentialHit;
```

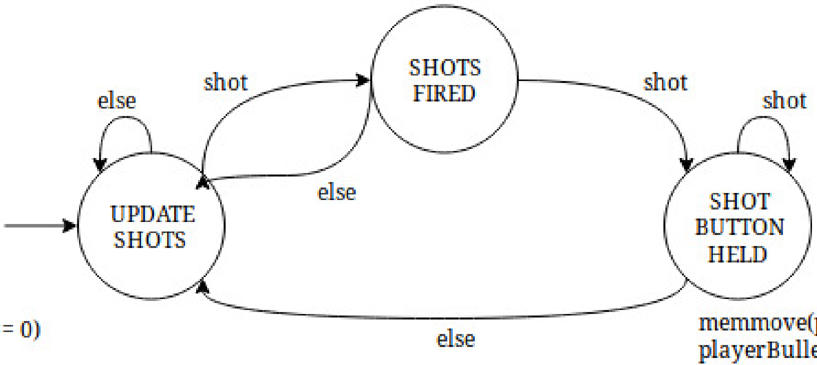
```
shot = 1;
potentialHit =
1;
```

```
shot = 0;
```

Task: Update Shots

Period: 100

```
playerBullets[0] = '-';
```



```
if(inMenu == 0)
{
   memmove(playerBullets + 1, playerBullets, 14);
   playerBullets[0] = ' ';
   for(int i = 0; i < activeBulletSpace; ++i)
   {
      LCD_Cursor(i + 2);
      LCD_WriteData(playerBullets[i])
   }
}
```

```
memmove(playerBullets + 1, playerBullets, 14);
playerBullets[0] = ' ';
for(int i = 0; i < activeBulletSpace; ++i)
{
   LCD_Cursor(i + 2);
   LCD_WriteData(playerBullets[i])
}
```

Task: Game Task

Period: 250

```
#define start (~PINA & 0x08)
#define EEPROM_SCORE_LOC 5
#define LCD_ROW_SIZE 15
#define OBSTACLE_BEGIN 2
#define OBSTACLE_SIZE 35

unsigned char currentScore;
unsigned char obstacle[OBSTACLE_SIZE] = {/* Insert obstacle here
*/};
```

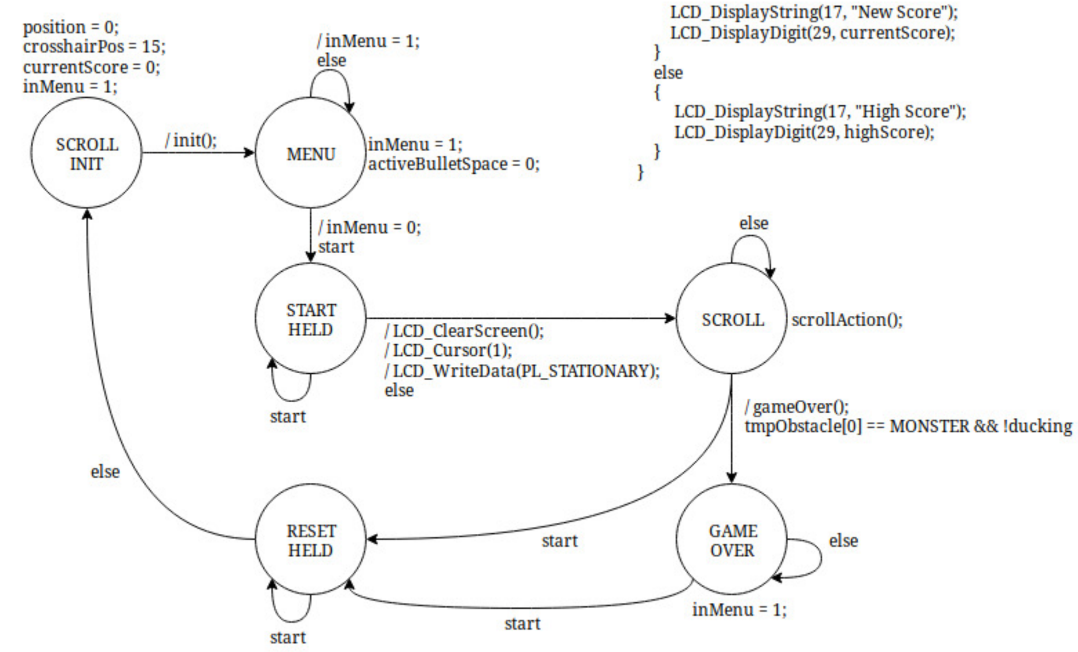
```
void init()
{
   if(eeprom_read_byte(EEPROM_SCORE_LOC) >= 255)
   {
      highScore = 0;
   }
   else
   {
      highScore = eeprom_read_byte(EEPROM_SCORE_LOC);
   }

   LCD_DisplayString(1, "Monster Shooter Press Start");
}
```

```
void gameOver()
{
   LCD_ClearScreen();
   LCD_DisplayString(1, "Game Over");

   if(currentScore > highScore)
   {
      eeprom_update_byte(EEPROM_SCORE_LOC, currentScore);

      LCD_DisplayString(17, "New Score");
      LCD_DisplayDigit(29, currentScore);
   }
   else
   {
      LCD_DisplayString(17, "High Score");
      LCD_DisplayDigit(29, highScore);
   }
}
```



```
void scrollAction()
{
          inMenu = 0;
          crosshairPos = getNextSprite(tmpObstacle, LCD_ROW_SIZE);
          if(potentialHit && (tmpObstacle[crosshairPos] == MONSTER))
          {
              potentialHit = 0;
              currentScore++;
              tmpObstacle[crosshairPos] = ' ';
          }
          else
          {
              potentialHit = 0;
          }

          if(tmpObstacle[0] == FRIENDLY && !ducking)
          {
              if(currentScore > 0)
              {
                  currentScore--;
              }
          }

          activeBulletSpace = getNextSprite(tmpObstacle, LCD_ROW_SIZE);
          for(i = 0; i < LCD_ROW_SIZE; ++i)
          {
              LCD_Cursor(i + OBSTACLE_BEGIN);
              LCD_WriteData(tmpObstacle[i]);
          }

          if(position < OBSTACLE_SIZE)
          {
              memmove(tmpObstacle, tmpObstacle + 1, LCD_ROW_SIZE - 1);
              tmpObstacle[LCD_ROW_SIZE - 1] = obstacle[position];

              ++position;
          }
          else
          {
              position = 0;
          }

          LCD_DisplayDigit(31, currentScore);
}
```

Code (Github Link)

Github Link: https://github.com/jdicarlantonio/CS120B_Project

There are four different tasks used in this project with their implementations defined in the header directory:

PlayerMovement (playerMovement.h)
This task manages the control of the player's movement. The player will either be ducking or stationary.

PlayerShoot and **UpdateShots** (playerShots.h)
PlayerShoot manages the actual shooting of the player while UpdateShots will handle the logic of when a player actually hits a monster

Game (game.h)
The game task handles the flow of the game. The player is either in the start menu, playing the game, or in the game over menu. This task oversees this logic.

In addition, the customChar.h file contains the data used to draw custom characters on the LCD display. A function was added to the provided io.h file in order to draw the custom characters.

Used Technologies/Components

The AVR-GCC toolchain was used to program the logic onto the ATMega1284. This included the AVR-GCC compiler, GDB, and the AVR C Library.

A TI shift register was used to reduce the amount of pins needed by the 16x2 LCD display.

Complexities/Build-upons

1. Using a shift register to lessen the pin out of the LCD
2. Drawing custom characters on the LCD
3. Storing a high score in EEPROM
4. Game logic

YouTube Link

https://www.youtube.com/watch?v=zOmHwxTv-bw

Known Bugs and Shortcomings

Bullet drawing bug:

There are a few bugs that arise when the player shoots bullets. Upon firing a bullet, the monster will be eliminated before the bullet reaches the monster. Furthermore, after the monster is eliminated the bullet continues to draw until reaching the next sprite. A possible fix for this is to have a variable to track the position of the bullet. That way, it should be possible to remove the monster’s sprite as soon as the bullet’s position is equal to the sprite’s position, and then halt the bullet drawing after.

Future Work

Aside from fixing the known bugs, there is some more work that can be done on this project. On the software side there could be a leveling system added with varying obstacles and increasing speeds on each level. There could also be a feature that negatively effects the player for shooting any of the friendly NPCs. In addition to the leveling system, there could also be new custom sprites added for different monsters that take more hits to kill.

On the hardware side, a larger LCD display, say a 16x4, could be used for more complex obstacles and better display. An additional LCD display could also be used for the display of rules and instructions as players begin the game. Furthermore, a casing could also be used to morph the system into a full handheld video game.