Final Project - Tanks

Matt Douglas, Priyanka Makin, Zachary Passarelli, Savio Tran

ECEN 2020

November 28, 2016

Introduction/Abstract

For our final embedded systems project, we decided to code a version of the game Tanks. Tanks is a video game in which the goal is to navigate the playing field and shoot the other player's tank. The first tank to be hit three times loses. Each round, barriers are randomly set on the field to give players an obstacle to maneuver around.

The user interface gives each player their own MSP432 LaunchPad and BoosterPack. The BosterPack's joystick and the lower button are used for turning and movement across the field while the upper button is used to fire bullets at the enemy. The two players' systems would be interfaced via wires.

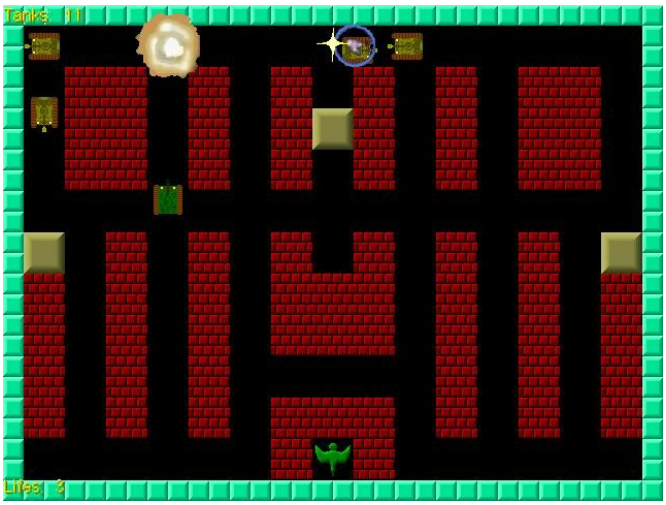

Figure 1 - Example of a Tanks game

Implementation

We started this project by designing a matrix playing field for our game. Our matrix is an enumerated array with 100 elements. We choose to interpret one row of the matrix as 10 elements of the array, which yields a 10 by 10 matrix as shown in Figure 2. The enumerations assign object representations to matrix values like empty spaces, barriers, tanks, and bullets. We make sure to include border spaces to prevent the tanks from moving off the playing field. The tanks' locations are initialized to start in the same spot each round. 10 randomly placed barriers are generated throughout the remaining free space of the playing field to create a new playing environment each round.

Each tank has its own enumerations describing its state. A structure holds the enumerations for the tank's information. The structure hold in formation like an ID variable unique to each player's tank, the direction the tank is facing, the tank's position in the array, and a variable holding the number of times the tank has been hit. A tank is destroyed after 3 hits. There are 8 possible directions the tank could be pointed in. The images below show the matrix setup as well as a visual describing the tank orientation.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

= Empty space
= Border barrier
= Randomly placed barrier
= Tank 1
= Tank 2

Figure 2 - Playing field matrix visualization

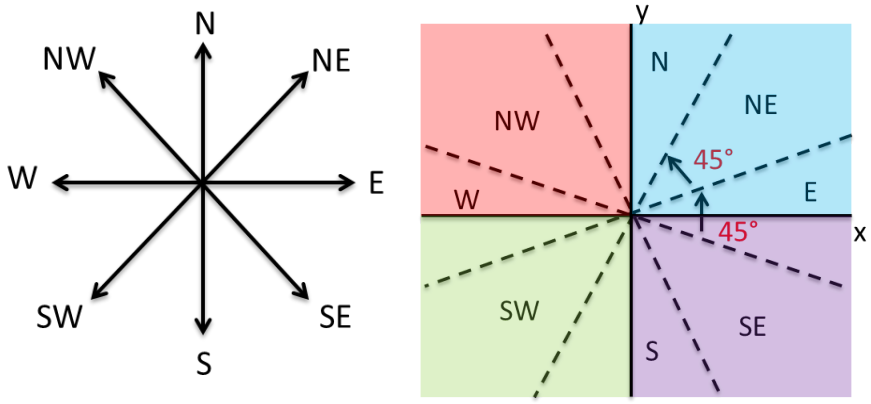

Figure 3 - Tank direction and joystick interpretation

After all this, we configure the pushbuttons and joystick on the Boosterpack, analog-to-digital conversion, timers, and UART like we learned in the previous labs.

The playing field array is created in main.c, however functions we wrote in other Code Composer files need to be able to change the values in the array. C functions call inputs by value, so we create an array of pointers that correspond to each element of the playing field array to pass into functions. Functions can dereference these pointers, which change the values of elements of the actual game array. Similarly, pointers to structures are passed into our functions to allow the functions to change the values in the structures.

The next step is writing the movement functions of the tanks and bullets. To move a tank, we first read its current orientation, configurable with the joystick. Figure 3 shows the coordinate plane used by the joystick and how we will interpret the user's input and convert that to a tank orientation. If an interrupt triggers because the bottom button was pushed, we read the tanks orientation and check the next space available in its line of sight. If it is empty, we move there, if it is occupied by the other tank or a barrier, we can't move. The tank is allowed to occupy a

space that already contains a bullet, but the tank will suffer a hit and the bullet will be destroyed. The next figure shows an example movement for tank 1.

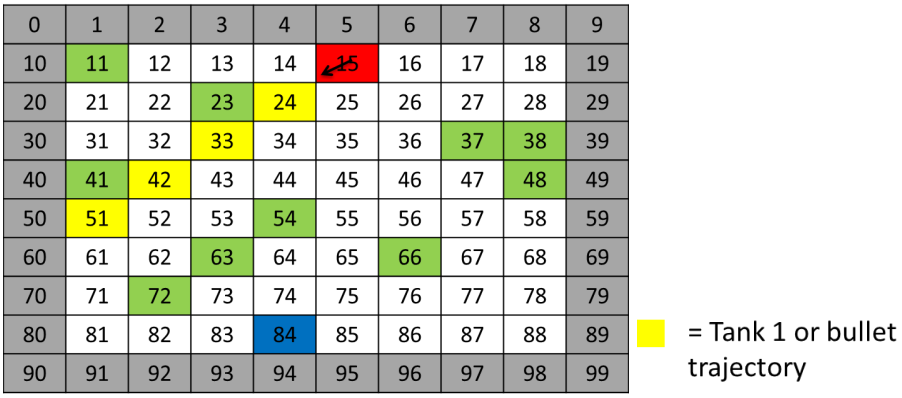

Figure 4 - Tank and/or bullet trajectory based on orientation

In order to move the tank, we add or subtract from the index of the tank, clear its previous location, and update the tank's appropriate structure members. In a separate 8 element array, we specify how many indices to add or subtract to the tank's position based on its orientation. These are put in order to match the direction enumeration as shown in Figure 4 below. So, for example, Index_Array[North] = -10. This value can then be added to the tank position value to find the position the tank wants to occupy.

Array index / Enumeration Value	0	1	2	3	4	5	6	7
Direction	N	NE	E	SE	S	SW	W	NW
Index_Array[direction]	-10	-9	1	11	10	9	-1	-11

Figure 5 - Index_Array visualization

To recap, in order to move, a user needs to hold the joystick in the direction they want to move, and press the lower button. The button press calls an interrupt that will increment a global variable from 0 to 1. Main polls to see if this value is 1, and if it is, it calls a function that changes the direction of the tank using an ADC reading. Then, main calls the movement function.

Bullets work in a similar fashion. Like tanks, we use a structure to keep track of a bullet's location and direction. We write a function to create a bullet when the top button is pressed. This function takes the tank's direction and checks that the bullet will be created in an empty space in

front of it. If the space is empty it becomes a bullet. The button interrupt is disabled, so only one bullet can be shot per tank at a time, and a timer interrupt, which will be used to move the bullet across the playing field, is enabled. The bullet structure is updated and the timer interrupt will repeatedly call a separate function to move the bullet in the same direction it is oriented toward. If there happens to be a barrier in the next spot, the bullet is destroyed. If there is a tank, then the tank gets a hit and the bullet is destroyed. Two bullets annihilate each other if they attempt to occupy the same space.

Results

We had a lot of ideas and high hopes for this final project, however time didn't permit for all our dreams to come true. Our final product is a single player Tanks game that logs tank movements and states via UART on RealTerm. Originally, we wanted to print our whole playing field matrix on RealTerm and to update it real time as players moved and shot, but it proved difficult to manipulate an array in C to print in matrix form. Also, we weren't quite sure how the logging statements, which were a requirement, would affect game play if they were dispersed between our playing field updating on RealTerm.

However, before our demo, we found some configurations on RealTerm that lets you arrange the data that is being transmitted by number of rows and columns. We wrote #ifdef statements around our logging functions so we could toggle between log statements and our playing field.

Additionally, on demo day, we found that our ADC must have not been configured correctly because the tanks didn't seem to orient the way we thought they would. When moving, the tank would sometimes move into the incorrect position, or it would relocate as though it was occupying a space it was in several moves ago. Also, we faced some button debounce issues because if we fired the bullets quite rapidly, they seemed to run into and destroy each other. They were shot out so rapidly that the timer interrupt was completely disregarded.

Another difficulty we faced involved integrating the second player into the game. We weren't really sure how two different LaunchPads and MSP432s would communicate to each other. However, most of our game play code takes into account the second tank, so all we would need is to configure the peripherals for a second player and figure out how the two MSP432s could write to the same matrix.

Future Ideas and Comments

There were many features we would have liked to add to the game. We would have liked to utilize the BoosterPack to a greater extent by integrating the LCD display to visualize the game and the buzzer for shooting sounds. The game could become wireless by integrating Bluetooth communication as well as a battery-operated power supply. This would create handheld devices that communicate and interact.

The gameplay could have been more interesting too. One entertaining idea involves hidden features, or “Easter Eggs”. For example, you could press all the buttons at the same time to blockade your opponent. The tanks could have more capabilities, too. There could be more attack features, like dropping a time bomb with a lethal blast radius. The playing field could have been improved by increasing resolution to match the pixel density on the LCD screen. Also, a game menu would improve user friendliness. This could include different gameplay modes, settings, and perhaps an information or help section. The possibilities with gameplay features are easy to add in the future, but we found it important to refrain from adding features that would add possible design complications.

Conclusion

As usual with design, there is never enough time. Planning the project consumed a lot of time, which took away from design and prototyping. In order to plan the project, we came up with all the enumerations, structures, and functions that would be necessary, and decided how they would interact before actually writing their definitions. Dividing the code into different chunks or “modules” helped organize what turned out to be a complex project and kept all our functions straight.

One of the most difficult aspects of this was ensuring that all variables keeping track of game data such as our game matrix and structures could be accessed by interrupts. Another thing that was quite challenging was getting all our peripherals configured correctly, like the joystick and button interrupts. This project, unlike many of our labs, involved a lot more software, which made it feel more applicable, as embedded systems concepts coupled with software created a game.

Appendix

Bill of Materials:

- TI MSP432 LaunchPad Microcontroller
- TI MKII BoosterPack
- Micro USB Cable

The following pages are the code files created and used for the project:

- Main C file
- Struct header and enumerations header files
- Logging header and C files
- Configure header and C files
- Initialize game header and C files
- Tank and bullet functions header and C files
- Interrupts header and C files
- UART header and C files

```c
//******************************************************************************
//
// MSP432 main.c template - Empty main
//
//******************************************************************************
#include "msp.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <stdint.h>

#include "enumerations.h"
#include "structs.h"

//declares functions
#include "logging.h"
#include "configure.h"
#include "initializegame.h"
#include "bulletandtank.h"
#include "interrupts.h"
#include "uart.h"

uint8_t endgame = 1; //Global counters
uint8_t mov_req = 0;
uint8_t bul_req = 0;
uint8_t bul_mov_req = 0;


void main(void){

    WDTCTL = WDTPW | WDTHOLD;
    configure_clock();
    configure_timer();
    configure_port();
    configure_serial_port();
    configure_ADC();
    __enable_interrupts();

    objects matrix[100] = {emptyspace}; //define a 10 by ten matrix of zeros
    objects * mat_ptr[100]; //array of pointers to use in functions
    int i;
    for (i=0; i<100; i++){
        mat_ptr[i] = &matrix[i];
    } //assign each element of the array of pointers to the corresponding matrix element
    init_matrix(mat_ptr);  //initialize matrix

    tankstate tank1state; //initialize structs for the two tanks
    tankstate tank2state;
    init_tanks(&tank1state, &tank2state);

    bulletstate tank1bullet;
    bulletstate tank2bullet; //each tank will have a structure for their bullet

    movement index_array[8] = {
            MV_NORTH,
            MV_NORTHEAST,
            MV_EAST,
```

```
58              MV_SOUTHEAST,
59              MV_SOUTH,
60              MV_SOUTHWEST,
61              MV_WEST,
62              MV_NORTHWEST}; //INDEXARRAY
63
64
65      uart_putchar_n(matrix,100); //display game field
66      log_game_start(); //logs start of game
67      while(endgame == 1){
68          ADC14->CTL0 |= ADC14_CTL0_SC; //Start joystick ADC conversion
69          //check counters for control requests
70          if(mov_req>0){ //move request
71              mov_req--; //decrement counter
72              change_dir(&tank1state); //call change direction function
73              movetank(mat_ptr, &tank1state, index_array); //call move tank function
74              uart_putchar_n(matrix,100); //update game field
75          }
76          if(bul_req>0){ //shoot request
77              bul_req--; //decrement counter
78              create_bullet(index_array, mat_ptr, &tank1state, &tank1bullet); //call create
  bullet
79              uart_putchar_n(matrix,100); //update game field
80          }
81          if(bul_mov_req>0){ //bullet movement request
82              bul_mov_req--; //decrement counter
83              move_bullet(mat_ptr,&tank1bullet,index_array,&tank1state,&tank2state); //call move
  bullet
84              uart_putchar_n(matrix,100); //update game field
85          }
86
87      }
88  }
89
```

```c
/*
 * enumerations.h
 *
 *  Created on: Nov 16, 2016
 *      Author: Savio
 */

#ifndef ENUMERATIONS_H_
#define ENUMERATIONS_H_

typedef enum objects_in_matrix_t{
    emptyspace = 0,
    tank1 = 1,
    tank2 = 2,
    bullet = 3,
    barrier = 4,

} objects;

typedef enum direction_t{
    north = 0,
    northeast = 1,
    east = 2,
    southeast = 3,
    south = 4,
    southwest = 5,
    west = 6,
    northwest = 7,
} direction;

typedef enum pos_move_t{
       MV_NORTH = -10,
       MV_NORTHEAST = -9,
       MV_EAST = 1,
       MV_SOUTHEAST = 11,
       MV_SOUTH = 10,
       MV_SOUTHWEST = 9,
       MV_WEST = -1,
       MV_NORTHWEST = -11,
    } movement;

#endif /* ENUMERATIONS_H_ */

```

```
1 /*
2  * structs.h
3  *
4  *  Created on: Nov 28, 2016
5  *      Author: Savio
6  */
7
8 #ifndef STRUCTS_H_
9 #define STRUCTS_H_
10
11 #include <stdint.h>
12
13 typedef struct tank_state_t{
14     volatile uint8_t hits; //tracks how many times the tank has been hit
15     volatile direction dir; // tracks what direction the tank is facing
16     volatile uint8_t position; // tracks which element of the matrix the tank is in
17     objects tanknumber; // identifies the tank (so we can log which tank moved/ got hit)
18 }tankstate;
19 // we need to define this struct twice in main for each tank
20
21 typedef struct bullet_data_t{
22     volatile uint8_t position; //holds current position of bullet
23     direction dir; //gives direction bullet will travel in
24 }bulletstate;
25
26 #endif /* STRUCTS_H_ */
27
```

```c
/*
 * logging.h
 *
 *  Created on: Nov 28, 2016
 *      Author: Savio
 */
// We need to log changes of direction of the tanks,  a tank being moved,
// a bullet being fired, a bullet being destroyed, and a tank being hit.  These are in real time,
// so data should be logged as it comes in. The functions transmit the data through uart
#ifndef LOGGING_H_
#define LOGGING_H_

#include "enumerations.h"
#include "structs.h"

void log_dir_change(tankstate * state); //logs direction changes
void log_movement(tankstate *state); //logs movement
void log_bullet_fired(tankstate * tank); //logs bullet fired
void log_bullet_destroyed(); //logs bullet destroyed
void log_tank_hit(tankstate * tank); //logs tank hit
void log_game_start(void); //logs game start
void log_game_over(tankstate * tank); //logs game over
void log_no_movement(tankstate * state); //logs no movement

#endif /* LOGGING_H_ */

```

```c
/*
 * logging.c
 *
 *  Created on: Dec 4, 2016
 *      Author: Savio
 */
#include "enumerations.h"
#include "structs.h"
#include "logging.h"
#include "uart.h"

void log_dir_change(tankstate * state){
    direction dir_tank = state->dir;
    if(state->tanknumber == tank1){
        switch (dir_tank) {
        case north :
            uart_putchar_n("Tank 1 now facing North\n", 25);
            break;
        case northeast :
            uart_putchar_n("Tank 1 now facing Northeast\n", 9);
            break;
        case east :
            uart_putchar_n("Tank 1 now facing East\n", 24);
            break;
        case southeast :
            uart_putchar_n("Tank 1 now facing southeast\n", 29);
            break;
        case south :
            uart_putchar_n("Tank 1 now facing South\n", 25);
            break;
        case southwest :
            uart_putchar_n("Tank 1 now facing southwest\n", 29);
            break;
        case west :
            uart_putchar_n("Tank 1 now facing west\n", 24);
            break;
        case northwest :
            uart_putchar_n("Tank 1 now facing Northwest\n", 29);
            break;
        }
    }
    else if(state->tanknumber == tank2){
        switch(dir_tank){
        case north :
            uart_putchar_n("Tank 2 now facing North", 23);
            break;
        case northeast :
            uart_putchar_n("Tank 2 now facing Northeast", 27);
            break;
        case east :
            uart_putchar_n("Tank 2 now facing East", 22);
            break;
        case southeast :
            uart_putchar_n("Tank 2 now facing Southeast", 27);
            break;
        case south :
            uart_putchar_n("Tank 2 now facing South", 23);
```

```
58              break;
59          case southwest :
60              uart_putchar_n("Tank 2 now facing Southwest", 27);
61              break;
62          case west :
63              uart_putchar_n("Tank 2 now facing West", 22);
64              break;
65          case northwest :
66              uart_putchar_n("Tank 2 now facing Northwest", 27);
67              break;
68          }
69      }
70  }
71
72  void log_no_movement(tankstate * state){
73      if(state->tanknumber == tank1){
74          uart_putchar_n("Tank 1 cannot move\n", 20);
75      }
76      else if(state->tanknumber == tank2){
77          uart_putchar_n("Tank 2 cannot move\n", 20);
78      }
79  }
80
81  void log_movement(tankstate * state){
82      direction dir_tank = state->dir;
83      if(state->tanknumber == tank1){
84          switch(dir_tank){
85          case north :
86              uart_putchar_n("Tank 1 moved North\n", 20);
87              break;
88          case northeast :
89              uart_putchar_n("Tank 1 moved Northeast\n", 24);
90              break;
91          case east :
92              uart_putchar_n("Tank 1 moved East\n", 19);
93              break;
94          case southeast :
95              uart_putchar_n("Tank 1 moved Southeast\n", 24);
96              break;
97          case south :
98              uart_putchar_n("Tank 1 moved South\n", 25);
99              break;
100         case southwest :
101             uart_putchar_n("Tank 1 moved Southwest\n", 29);
102             break;
103         case west :
104             uart_putchar_n("Tank 1 moved west\n", 19);
105             break;
106         case northwest :
107             uart_putchar_n("Tank 1 moved Northwest\n", 24);
108             break;
109         }
110     }
111     else if(state->tanknumber == tank2){
112         switch(dir_tank){
113         case north :
114             uart_putchar_n("Tank 2 moved North", 18);
```

```
115             break;
116         case northeast :
117             uart_putchar_n("Tank 2 moved Northeast", 22);
118             break;
119         case east :
120             uart_putchar_n("Tank 2 moved East", 17);
121             break;
122         case southeast :
123             uart_putchar_n("Tank 2 moved Southeast", 22);
124             break;
125         case south :
126             uart_putchar_n("Tank 2 moved East", 18);
127             break;
128         case southwest :
129             uart_putchar_n("Tank 2 moved southwest", 22);
130             break;
131         case west :
132             uart_putchar_n("Tank 2 moved west", 17);
133             break;
134         case northwest :
135             uart_putchar_n("Tank 2 moved Northwest", 22);
136             break;
137         }
138     }
139 }
140
141
142 void log_bullet_fired(tankstate * tank){
143     if (tank->tanknumber == tank1){
144         uart_putchar_n("Tank 1 fired a bullet\n", 23);
145     }
146     else if (tank->tanknumber == tank2){
147         uart_putchar_n("Tank 2 fired a bullet\n", 23);
148     }
149 }
150 void log_bullet_destroyed(void){
151     uart_putchar_n("Bullet has been destroyed\n", 27);
152 }
153
154 void log_tank_hit(tankstate * state){
155
156     if (state->tanknumber == tank1){
157         uart_putchar_n("Tank 1 has been hit\n", 21);
158     }
159     else if (state->tanknumber == tank2){
160         uart_putchar_n("Tank 2 has been hit\n", 21);
161     }
162 }
163
164 void log_game_start(void){
165     uart_putchar_n("Game has been started\n", 23);
166 }
167
168 void log_game_over(tankstate * tank){
169     //input loser
170     if (tank->tanknumber == tank1){
171         uart_putchar_n("Tank 1 won the game.  Press reset button for new game\n", 55);
```

```
172     }
173     else if (tank->tanknumber == tank2){
174         uart_putchar_n("Tank 2 won the game.  Press reset button for new game\n", 55);
175     }
176 }
177
```

```
 1 /*
 2  * configure.h
 3  *
 4  *  Created on: Dec 7, 2016
 5  *      Author: Zach
 6  */
 7 //Functions for configuring msp432 functionality
 8 #ifndef CONFIGURE_H_
 9 #define CONFIGURE_H_
10
11 void configure_clock(void);
12 void configure_timer(void);
13 void configure_port(void);
14 void configure_ADC(void);
15
16 #endif /* CONFIGURE_H_ */
17
```

```c
/*
 * configure.c
 *
 *  Created on: Dec 4, 2016
 *      Author: Savio
 */
#include "msp.h"
#include "configure.h"

void configure_clock(void){
    //Configure clock
    CS->KEY = 0x695A; //Unlock CS module for register access
    CS->CTL0 = 0; //Reset tuning parameters
    CS->CTL0 = CS_CTL0_DCORSEL_1; //Setup DCO clock (3 MHz)
    //Select ACLK = REFO, SMCLK = MCLK = DCO
    CS->CTL1 = CS_CTL1_SELA_2 | CS_CTL1_SELS_3 | CS_CTL1_SELM_3;
    CS->KEY = 0; //Lock CS module for register access
}

void configure_timer(void){
    //Timer source: SMCLK; Up mode; Prescalar: 1/8
    TIMER_A0->CTL |= TIMER_A_CTL_SSEL__SMCLK | TIMER_A_CTL_MC__UP | TIMER_A_CTL_ID_3;
    TIMER_A0->R = 0; //reset counter
    TIMER_A0->CCR[0] = 50000; //capture compare value
    NVIC_EnableIRQ(TA0_0_IRQn); //enable timer interrupt
}

void configure_port(void){
     //pin 5.1 (upper button) configuration
     P5DIR &= BIT1; //input direction
     P5OUT |= BIT1; //pullup resistor
     P5REN |= BIT1;  //enable resistor
     P5IFG &= ~BIT1; //reset flag
     P5IES |= BIT1; //falling edge trigger
     P5IE |= BIT1; //enable interrupt

     //pin 3.5 (lower button) configuration
     P3DIR &= BIT5; //input direction
     P3OUT |= BIT5; //pullup resistor
     P3REN |= BIT5; //enable resistor
     P3IFG &= ~BIT5; //reset flag
     P3IES |= BIT5; //falling edge trigger
     P3IE |= BIT5; //enable interrupt

     NVIC_EnableIRQ(PORT3_IRQn); //enable port interrupts
     NVIC_EnableIRQ(PORT5_IRQn);

    //joystick: pin 4.4(y axis); pin 6.0(x axis)
    P6->SEL0 |= BIT0; //Primary mode (ADC)
    P4->SELC |= BIT4; //Tertiary mode (ADC)
}

void configure_ADC(void){
    //Configure ADC - Pulse sample mode; ADC14SC trigger
    //ADC ON, sample period > 30us
    ADC14->CTL0 |= ADC14_CTL0_SHT0_5 | ADC14_CTL0_ON | ADC14_CTL0_SHP;
    ADC14->CTL0 |= ADC14_CTL0_CONSEQ_1;
```

```
58      //Configure res (10 bit)
59      ADC14->CTL1 |= ADC14_CTL1_RES_1;
60      //Map joystick analog channels to (x)MEM1/MCTL1 and (y)MEM2/MCTL2
61      ADC14->MCTL[0] |= ADC14_MCTLN_INCH_15| ADC14_MCTLN_VRSEL_0;
62      ADC14->MCTL[1] |= ADC14_MCTLN_INCH_9 | ADC14_MCTLN_VRSEL_0;
63      ADC14->CTL0 |= ADC14_CTL0_ENC; //Enable conversions
64      NVIC_EnableIRQ(ADC14_IRQn); //Enable ADC interrupts
65  }
66
```

```
1 /*
2  * initializegame.h
3  *
4  *  Created on: Nov 28, 2016
5  *      Author: Savio
6  */
7
8 #ifndef initializegame_H_
9 #define initializegame_H_
10
11 #include "enumerations.h"
12 #include "structs.h"
13
14 void init_matrix(objects * mat_ptr[100]);
15 //takes an array of pointers to the matrix, and initializes the mat_ptr with barriers around
   the edges
16 //Also places tanks in the matrix
17 //Adds 10 random barriers onto the playing field
18
19 void init_tanks(tankstate * tank1state, tankstate * tank2state);
20 // sets each tank's hit value to 0
21 //also starts the first player facing south and the second player facing north
22
23 #endif /* initializegame_H_ */
24
```

```c
/*
 * initializegame.c
 *
 *  Created on: Dec 4, 2016
 *      Author: Savio
 */
#include "initializegame.h"
#include "msp.h"
#include <time.h>
#include <stdlib.h>
#include "enumerations.h"
#include "structs.h"


void init_matrix(objects * mat_ptr[100]) {
    int i;
    for(i=0; i<10; i++){ //set out edge of playing field to barriers
        *mat_ptr[i]=barrier;
    } //top row barriers
    for(i=10; i<100; i+=10){
        *mat_ptr[i]= barrier;
    } //left side barriers
    for(i=19; i<100; i=i+10){
        *mat_ptr[i] = barrier;
    } //right side barriers
    for(i=91; i<99; i++){
        *mat_ptr[i] = barrier;
    } //bottom barriers

    *mat_ptr[15]= tank1; //initialize tank 1
    *mat_ptr[84]= tank2; //initialize tank 2
    srand(time(NULL)); //initalize for rand
    for(i=0; i<10;i++){
        int pos= rand() % 100; //generate random number
        while (*mat_ptr[pos] != emptyspace) {
            pos= rand() % 100; //reroll if space is occupied
        }
            *mat_ptr[pos]= barrier; // randomly place ten barriers
    }
}

void init_tanks(tankstate * tank1_ptr, tankstate * tank2_ptr) {
    tank1_ptr->tanknumber = tank1;
    tank2_ptr->tanknumber = tank2; //tank number designation
    tank1_ptr->hits = 0;
    tank2_ptr->hits = 0; //both tanks start with 0 hits
    tank1_ptr->dir = south;
    tank2_ptr->dir = north; //initializes direction of tanks
    tank1_ptr->position = 14;
    tank2_ptr->position = 84;// initializes tank's position
}

```

```
/*
 * bullet.h
 *
 *  Created on: Dec 1, 2016
 *      Author: Savio
 */

#ifndef BULLETANDTANK_H_
#define BULLETANDTANK_H_

//all of these functions will log what they do via uart

//Note, the matrix is 10 by 10, so it is an array with 100 elements.
//So, we just need to interpret the array as a matrix.  So, positions
//0 through 9 would be the first row of the matrix, and 10 through 19
//make up the second row.  This goes on until the last row, of positions 90
//through 99.  So, moving in the matrix is equivalent to changing positions
//in the array

#include "enumerations.h"
#include "structs.h"

void movetank(objects * mat_ptr[100], tankstate * tank, movement index_array[8]);
//this function checks the tankstate structure which tank is moving to move the proper tank.
//Then, it checks which direction and moves the tank accordingly.  To move
// we need to check that the new position is an empty space, then, if it is
//the function sets the new space equal to the tank, and makes it's former position
//an empty space.  It also updates the position variable.

void change_dir(tankstate * tank);
//takes a direction that the tank will change to and a pointer to the tank (that wants to change direction) structure
uint16_t xpos; //data holders for joystick ADC conversions
uint16_t ypos;

void create_bullet(movement index_array[8], objects * mat_ptr[100], tankstate * tank, bulletstate * bul);
//creates a bullet in the matrix using the index array, the matrix of pointers to all the array elements, and
// struct pointers to the tank making the bullet and a bullet structure.
// this function will turn on timer interrupt if the bullet is created in an empty space

void move_bullet(objects * mat_ptr[100], bulletstate * bul, movement index_array[8], tankstate * tank_1, tankstate * tank_2);
//must put pointer to tank 1 and tank 2 in proper order
// takes the index array, the pointer matrix, the bullet's structure, and both tank's structures to move the bullet
//note, this function will propagate the bullet one position each time it is called by the timer interrupt.
//However, once a bullet is destroyed, it will disable the timer interrupt, and enable the button interrupt

#endif /* BULLETANDTANK_H_ */

```

```c
/*
 * tank_functions.c
 *
 *  Created on: Dec 4, 2016
 *      Author: Savio
 */
#include "msp.h"
#include <stdint.h>
#include "enumerations.h"
#include "structs.h"
#include "bulletandtank.h"
#include "logging.h"
#define logging

#define THIRD (1023/3) //define area for joystick interpretation
extern uint8_t endgame;

void movetank(objects * mat_ptr[100], tankstate * tank, movement index_array[8]){
    movement index = index_array[tank->dir]; //this index value allows us to move based on the tank's direction
    objects objecttype = *mat_ptr[tank->position + index];  //finds what is in the space tank wants to be move into
    //tank moves into empty space
    if (objecttype == emptyspace ){
        if (tank->tanknumber == tank1){
            *mat_ptr[tank->position] = emptyspace; //set old position to empty
            *mat_ptr[tank->position + index] = tank1; //set new position to tank
#ifdef logging
            log_movement(tank);
#endif
        }
        else if (tank->tanknumber == tank2){
            *mat_ptr[tank->position] = emptyspace;
            *mat_ptr[tank->position + index] = tank2;
#ifdef logging
            log_movement(tank);
#endif
        }
        tank->position += index; //update position in tank struct
    }
    //tank moves into bullet
    else if (objecttype == bullet){
        if (tank->hits < 2){
            tank->hits++; //register as a hit
            TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIE; //disable timer interrupt
            P5IE |= BIT1;   //turn on button interrupt
#ifdef logging
            log_tank_hit(tank);
#endif
            if (tank->tanknumber == tank1){
                *mat_ptr[tank->position] = emptyspace;
                *mat_ptr[tank->position + index] = tank1;
#ifdef logging
                log_movement(tank);
#endif
            }
            else if (tank->tanknumber == tank2){
```

```
56                  *mat_ptr[tank->position] = emptyspace;
57                  *mat_ptr[tank->position + index] = tank2;
58#ifdef logging
59                  log_movement(tank);
60#endif
61              }
62          }
63          else if (tank->hits == 2){
64#ifdef logging
65              log_game_over(tank);
66#endif
67              endgame=0; //global to end game loop in main
68          }
69      }
70      //tank moves into barrier or other tank
71      else if (objecttype == barrier | objecttype == tank1 | objecttype == tank2) {
72#ifdef logging
73          log_no_movement(tank);
74#endif
75      }
76}
77
78void change_dir(tankstate * tank){
79      ADC14->CTL0 |= ADC14_CTL0_SC; //start ADC conversion
80      while(ADC14->CTL0 & ADC14_CTL0_BUSY); //wait for conversion
81      xpos = ADC14->MEM[0]; //get data from conversion
82      ypos = ADC14->MEM[1];
83      //find joystick position using ADC results and update tank direction in struct
84      if(ypos >= 2*THIRD){
85          if(xpos >= 2*THIRD){
86              //northeast
87              tank->dir = northeast;
88          }
89          else if(xpos >= THIRD){
90              //north
91              tank->dir = north;
92          }
93          else{
94              //northwest
95              tank->dir = northwest;
96          }
97      }
98      else if(ypos >= THIRD){
99          if(xpos >= 2*THIRD){
100             //east
101             tank->dir = east;
102         }
103         else{
104             //west
105             tank->dir = west;
106         }
107     }
108     else{
109         if(xpos >= 2*THIRD){
110             //southeast
111             tank->dir = southeast;
112         }
```

```
113         else if(xpos >= THIRD){
114             //south
115             tank->dir = south;
116         }
117         else{
118             //southwest
119             tank->dir = southwest;
120         }
121     }
122 #ifdef logging
123     log_dir_change(tank);
124 #endif
125 }
126
```

```c
/*
 * bullet.c
 *
 *  Created on: Dec 4, 2016
 *      Author: Savio
 */
#include "msp.h"
#include <stdint.h>
#include "bulletandtank.h"
#include "logging.h"
#define logging

extern uint8_t endgame;

void create_bullet(movement index_array[8], objects * mat_ptr[100], tankstate * tank, bulletstate * bul){
    movement index = index_array[tank->dir];  //This will later be added to the position of the tank to get the bullet position
    //tank and bullet in same direction, so tank->dir just gives direction of the bullet
    objects objecttype = *mat_ptr[tank->position + index];  //finds what is in the space bullet wants to be created
        //bullet created on empty space
        if (objecttype == emptyspace){
                *mat_ptr[tank->position + index] = bullet; //set location as bullet
                bul->dir = tank->dir; //bullet in same direction as tank
                bul->position = tank->position + index; //setting up the bullet struct for movement
                TIMER_A0->CCTL[0] |= TIMER_A_CCTLN_CCIE; //Enable timer interrupt for bullet movement
#ifdef logging
                log_bullet_fired(tank);
#endif
        }
        //bullet created on a barrier
        else if (objecttype == barrier ){
#ifdef logging
            log_bullet_destroyed(bul);
#endif
        }
        //bullet created on another bullet
        else if (objecttype == bullet ){
#ifdef logging
            log_bullet_destroyed(bul);
#endif
        }
        //bullet created on tank 1
        else if (objecttype == tank1 ){
            if (tank->hits < 2){
                tank->hits++; //register as a hit
                tankstate tankhit;
                tankhit.tanknumber = tank1;
#ifdef logging
                log_tank_hit(&tankhit);
#endif
            }
            else if (tank->hits == 2){
                tankstate tankhit;
```

```
53                  tankhit.tanknumber = tank2;
54#ifdef logging
55                  log_game_over(&tankhit);
56#endif
57                  endgame = 0; //global to end game loop in main
58              }
59
60          }
61          //bullet created on tank 2
62          else if (objecttype == tank2 ){
63              if (tank->hits < 2){
64                  tank->hits++; //register as a hit
65                  tankstate tankhit;
66                  tankhit.tanknumber = tank2;
67#ifdef logging
68                  log_tank_hit(&tankhit);
69#endif
70              }
71              else if (tank->hits == 2){
72                  tankstate tankhit;
73                  tankhit.tanknumber = tank1;
74#ifdef logging
75                  log_game_over(&tankhit);
76#endif
77                  endgame = 0; //global to end game loop in main
78              }
79
80          }
81}
82
83void move_bullet(objects * mat_ptr[100], bulletstate * bul, movement index_array[8], tankstate
  * tank_1, tankstate * tank_2){
84      movement index = index_array[bul->dir];  //This will later be added to the position of the
  bullet to get the new bullet position
85      objects objecttype = *mat_ptr[bul->position + index];  //finds what is in the space bullet
  wants to be move into
86          //bullet moves into empty space
87          if (objecttype == emptyspace ){
88                  *mat_ptr[bul->position + index] = bullet; //set new position as bullet
89                  *mat_ptr[bul->position] = emptyspace; //clear old position
90                  bul->position += index; //setting up the bullet struct for movement
91          }
92          //bullet moves into barrier
93          else if (objecttype == barrier ){
94              TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIE; //disable timer interrupt
95              *mat_ptr[bul->position] = emptyspace; //remove bullet
96              P5IE |= BIT1; //enable shoot button
97#ifdef logging
98              log_bullet_destroyed();
99#endif
100
101         }
102         //bullet moves into another bullet
103         else if (objecttype == bullet ){
104             TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIE; //disable timer interrupt
105             *mat_ptr[bul->position] = emptyspace; //remove bullet
106             P5IE |= BIT1; //enable shoot button
```

```
107 #ifdef logging
108             log_bullet_destroyed();
109 #endif
110         }
111         //bullet moves into tank 1
112         else if (objecttype == tank1 ){
113             *mat_ptr[bul->position] = emptyspace; //remove bullet
114             if (tank_1->hits < 2){
115                 tank_1->hits ++; //register as a hit
116                 TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIE; //disable timer interrupt
117                 P5IE |= BIT1; //enable shoot button
118 #ifdef logging
119                 log_tank_hit(tank_1);
120 #endif
121
122             }
123             else if (tank_1->hits == 2){
124 #ifdef logging
125                 log_game_over(tank_2); //tank 2 wins
126 #endif
127                 endgame = 0; //global to end game loop in main
128             }
129
130         }
131         //bullet moves into tank 2
132         else if (objecttype == tank2 ){
133             *mat_ptr[bul->position] = emptyspace; //remove bullet
134             if (tank_2->hits < 2){
135                 tank_2->hits ++; //register as a hit
136                 TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIE; //disable timer interrupt
137                 P5IE |= BIT1; //enable shoot button
138 #ifdef logging
139                 log_tank_hit(tank_2);
140 #endif
141             }
142             else if (tank_2->hits == 2){
143 #ifdef logging
144                 log_game_over(tank_1); //tank 1 wins
145 #endif
146                 endgame = 0; //global to end game loop in main
147             }
148         }
149 }
150
```

```
1 /*
2  * configure_button_interrupts.h
3  *
4  *  Created on: Nov 16, 2016
5  *      Author: Savio
6  */
7 #ifndef CONFIGURE_BUTTON_INTERRUPTS_H_
8 #define CONFIGURE_BUTTON_INTERRUPTS_H_
9 // All handlers will increment request counters when interrupts occur.
10 // Request counters are read in main and appropriate functions will be called.
11 void PORT5_IRQHandler(void);
12 // Increases bullet request counter. Game loop will call create bullet function.
13 // This needs to enable a timer interrupt that will then control the movement
14 // of the bullet.  This should disable port 5 button interrupts, and then
15 // once the bullet is destroyed, we can enable this button interrupt again
16
17 void PORT3_IRQHandler(void);
18 // Increases movement request counter. Game loop will call change direction and move functions.
19
20 void TA0_0_IRQHandler(void);
21 // Increases bullet movement request counter. Game loop will call move bullet function.
22
23 #endif /* CONFIGURE_BUTTON_INTERRUPTS_H_ */
24
```

```c
/*
 * buttoninterrupts.c
 *
 *  Created on: Dec 4, 2016
 *      Author: Savio
 */
#include "msp.h"
#include <stdint.h>
#include "interrupts.h"

extern uint8_t bul_req; //global request counters from main
extern uint8_t mov_req;
extern uint8_t bul_mov_req;

void PORT5_IRQHandler(void){
    //pin 5.1 (upper button)
    if(P5IFG & BIT1){
        P5IE &= BIT1; //disable button interrupt.  Note, this will be enabled again once the bullet is destroyed
        bul_req++; //increment create bullet request counter
        P5IFG &= ~BIT1; //clear flag
    }
}

void PORT3_IRQHandler(void){
    //pin 3.5 (lower button)
    if(P3IFG & BIT5){
        P3IFG &= ~BIT5; //clear flag
        mov_req++; //increment move request counter
    }
}

void TA0_0_IRQHandler(void){
    if(TIMER_A0->CCTL[0] & TIMER_A_CCTLN_CCIFG){
        TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG; //clear flag
        bul_mov_req++; //increment bullet move request counter
    }
}

```

```c
/*
 * uart.h
 *
 *  Created on: Dec 5, 2016
 *      Author: Zach
 */
#ifndef UART_H_
#define UART_H_

#include <stdint.h>

void uart_putchar(uint8_t tx_data);
//Transmits a single character over UART interface
// by loading data onto TX buffer

void uart_putchar_n(uint8_t *data, uint32_t length);
//Transmits multiple characters over UART interface
// by iterating through data and using single character function

void configure_serial_port(void);
//msp432 configuration for UART

#endif /* UART_H_ */

```

```c
/*
 * uart.c
 *
 *  Created on: Dec 5, 2016
 *      Author: Zach
 */
#include "uart.h"
#include <stdint.h>
#include "msp.h"

void uart_putchar(uint8_t tx_data){
    while(!(UCA0IFG & UCTXIFG)); //Wait for transmitter ready
    UCA0TXBUF = tx_data; //Load data onto buffer for transmission
}

void uart_putchar_n(uint8_t *data, uint32_t length){
    uint32_t i;
    for(i = 0; i< length;i++){ //Iterate through data
        //Transmit 1 character per iteration using putchar function
        uart_putchar(data[i]);
    }
}

void configure_serial_port(void) {
    //Configure UART pins: primary function
    P1SEL0 |= BIT2 | BIT3;
    //Configure UART: SMCLK src
    UCA0CTLW0 |= UCSWRST; //Put eUSCI in reset
    UCA0CTLW0 |= UCSSEL_2;
    UCA0BRW = 26; //Set Baud Rate (115200)
    UCA0CTLW0 &= ~UCSWRST; //Initialize eUSCI
}

```