```c
/*
 * tank.c
 * Taylor Cowley and Andrew Okazaki
 */

#include <stdint.h>
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "sound/sound.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "util.h"
#include "interface.h"      // enable to take life afaw from tank
#include "bunkers.h"        // tank shell to hit bunker
#include "aliens.h"         // required to call collision detection function
#include "mother_ship.h"    // required to collition detection to kill her.
#include "tank.h"
#define TANK_HEIGHT     8       // Tank is 8 pixels high
#define TANK_DEATH_HEIGHT 16    // height of tank death sprite
#define TANK_DEATH_WIDTH 26     // width of tank death sprite
#define TANK_DEATH_TIME 400
#define TANK_WIDTH      15      // Tank is 15 pixels wide
#define TANK_INIT_ROW   210     // Tank starts at row 210
#define TANK_INIT_COL   160     // Tank starts at col 160
#define SHELL_LENGTH 3          // Shell is 3 pixels long
#define SHELL_COL_OFFSET 7      // Shell is 7 pixels offset from the tank
#define SHELL_DEATH_ROW 20
#define EXPLOSION_ROW_OFFSET -1 // tank explosion row offset
#define EXPLOSION_COL_OFFSET -4 // tank explosion column offset

#define GREEN 0x0000FF00        // Hex value for green
#define BLACK 0x00000000        // Hex value for black
#define WHITE 0xFFFFFFFF        // Hex value for white

// Packs each horizontal line of the figures into a single 32 bit word.
#define packword15(b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0)  \
        ((b14 << 14) | (b13 << 13) | (b12 << 12) | (b11 << 11) | (b10 << 10) | \
        (b9  << 9 ) | (b8  << 8 ) | (b7  << 7 ) | (b6  << 6 ) | (b5  << 5 ) | \
        (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ) )


#define
packWord26(b25,b24,b23,b22,b21,b20,b19,b18,b17,b16,b15,b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,
b4,b3,b2,b1,b0) \
((b25 << 25) | (b24 << 24) |                              \
 (b23 << 23) | (b22 << 22) | (b21 << 21) | (b20 << 20) | (b19 << 19) | (b18 << 18) | (b17
 << 17) | (b16 << 16) |                          \
 (b15 << 15) | (b14 << 14) | (b13 << 13) | (b12 << 12) | (b11 << 11) | (b10 << 10) | (b9
<< 9 ) | (b8  << 8 ) |                           \
 (b7  << 7 ) | (b6  << 6 ) | (b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1
<< 1 ) | (b0  << 0 ) )
static const int tank_15x8[TANK_HEIGHT] = {     // This is how we
        packword15(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0),  // Store the tank
        packword15(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0),  // drawing data
        packword15(0,0,0,0,0,0,1,1,1,0,0,0,0,0,0),
```

```
        packword15(0,1,1,1,1,1,1,1,1,1,1,1,1,1,0),
        packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword15(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)};
static const int tankDeath1[TANK_DEATH_HEIGHT] = {
    packWord26(0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packWord26(0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packWord26(0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0),
    packWord26(0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,0,0),
    packWord26(0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,1,1,0,0,1,1,0,0),
    packWord26(0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,1,1,0,0,1,1,0,0),
    packWord26(0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,0),
    packWord26(0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,0),
    packWord26(1,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0),
    packWord26(1,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0),
    packWord26(0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
    packWord26(0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
    packWord26(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    packWord26(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),};
static const int tankDeath2[TANK_DEATH_HEIGHT] = {
    packWord26(1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(1,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packWord26(0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packWord26(0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0),
    packWord26(0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0),
    packWord26(1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0),
    packWord26(1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0),
    packWord26(0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1),
    packWord26(0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1),
    packWord26(0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0),
    packWord26(0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0),
    packWord26(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    packWord26(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),};

#define WORD_WIDTH 15

struct tank{        // The struct for our tank
    int row;        // Tank's row
    int col;        // Tank's column
}tank;

struct tank_shell{  // The struct that stores the tank's bullet data
    int row;        // Shell's row
    int col;        // Shell's column
    bool alive;     // Whether it is alive
}tank_shell;

// ------------------------------------------
// Our declaration of functions to be used
void tank_kill_bullet(uint32_t * framePointer);
// Ending declaration of internal functions
// ------------------------------------------
```

```
// This initializes our tank at its proper location
void tank_init(){
    tank.row = TANK_INIT_ROW;       // Tank starts at this row
    tank.col = TANK_INIT_COL;       // and column
}
uint32_t * frame; // frame pointer
// This draws (or erases, via the erase bool) an entire tank.
void tank_draw(uint32_t * framePointer, bool erase){
    frame = framePointer;
    int color = erase ? BLACK : GREEN ;     // green or black depending on erase
    int row, col;                           // init loop vars
    for(row=0;row<TANK_HEIGHT;row++){       // Go through tank x pixels
        for(col=0;col<WORD_WIDTH;col++){    // and tank y pixels
            if ((tank_15x8[row] & (1<<(WORD_WIDTH-col-1)))) {   // If a pixel
                //  Draw the pixel
                util_draw_pixel(framePointer, row+tank.row,col+tank.col,color);
            }
        }
    }
}

// moves our tank left by a certain number of pixels
void tank_move_left(uint32_t * framePointer){
#define L_0_GREEN   7   // When moving left,
#define L_2_GREEN   6   // where to
#define L_3_GREEN   1   // draw green
#define L_7_GREEN   0   // pixels based on row

#define L_0_BLACK   8   // When moving left,
#define L_2_BLACK   9   // where to
#define L_3_BLACK   14  // erase pixels
#define L_7_BLACK   15  // based on row

    if(tank.col <= 0){
        return;     // Can't go past edge of the screen
    }

    tank.col --;        // Move our tank left by a pixel
    int row;            // Declare loop var
    for(row = 0; row < TANK_HEIGHT; row++){
        switch (row){   // Depending on the row
        case 0:         // Draw/erase proper pixels
            util_draw_pixel(framePointer,row+tank.row,L_0_GREEN+tank.col,GREEN);
            util_draw_pixel(framePointer,row+tank.row,L_0_BLACK+tank.col,BLACK);
            break;
        case 1: // Cases 1 and 2 are identical
        case 2:         // Keep drawing/erasing pixels
            util_draw_pixel(framePointer,row+tank.row,L_2_GREEN+tank.col,GREEN);
            util_draw_pixel(framePointer,row+tank.row,L_2_BLACK+tank.col,BLACK);
            break;
        case 3:         // Keep drawing/erasing pixels
            util_draw_pixel(framePointer,row+tank.row,L_3_GREEN+tank.col,GREEN);
            util_draw_pixel(framePointer,row+tank.row,L_3_BLACK+tank.col,BLACK);
            break;
        case 4: // Cases 4, 5, 6, and 7 are all identical.
        case 5:
        case 6:
```

```
        case 7:         // Keep drawing/erasing pixels
            util_draw_pixel(framePointer,row+tank.row,L_7_GREEN+tank.col,GREEN);
            util_draw_pixel(framePointer,row+tank.row,L_7_BLACK+tank.col,BLACK);
            break;
        }
    }
}

//moves our tank right by a certain number of pixels
void tank_move_right(uint32_t * framePointer){
#define R_0_GREEN 7     // When moving
#define R_1_GREEN 8     // right,
#define R_2_GREEN 8     // which pixels
#define R_3_GREEN 13    // are
#define R_4_GREEN 14    // to
#define R_5_GREEN 14    // be drawn
#define R_6_GREEN 14    // green
#define R_7_GREEN 14    // based on the row

#define R_0_BLACK 6     // When moving
#define R_1_BLACK 5     // right,
#define R_2_BLACK 5     // which pixels
#define R_3_BLACK 0     // are
#define R_4_BLACK -1    // to
#define R_5_BLACK -1    // be ERASED
#define R_6_BLACK -1    // with black
#define R_7_BLACK -1    // based on the row

    if(tank.col+TANK_WIDTH >= UTIL_SCREEN_WIDTH){
        return;     // Can't go past edge of the screen
    }
    tank.col ++;    // Move our tank right by a single pixel
    int r = 0;      // Start our count pointer
    // Draw and erase the proper pixels for row 0
    util_draw_pixel(framePointer, r+tank.row, R_0_GREEN+tank.col, GREEN);
    util_draw_pixel(framePointer, r+tank.row, R_0_BLACK+tank.col, BLACK);
    r++;            // increment row counter
    // Draw and erase the proper pixels for row 1
    util_draw_pixel(framePointer, r+tank.row, R_1_GREEN+tank.col, GREEN);
    util_draw_pixel(framePointer, r+tank.row, R_1_BLACK+tank.col, BLACK);
    r++;            // increment row counter
    // Draw and erase the proper pixels for row 2
    util_draw_pixel(framePointer, r+tank.row, R_2_GREEN+tank.col, GREEN);
    util_draw_pixel(framePointer, r+tank.row, R_2_BLACK+tank.col, BLACK);
    r++;            // increment row counter
    // Draw and erase the proper pixels for row 3
    util_draw_pixel(framePointer, r+tank.row, R_3_GREEN+tank.col, GREEN);
    util_draw_pixel(framePointer, r+tank.row, R_3_BLACK+tank.col, BLACK);
    r++;            // increment row counter
    // Draw and erase the proper pixels for row 4
    util_draw_pixel(framePointer, r+tank.row, R_4_GREEN+tank.col, GREEN);
    util_draw_pixel(framePointer, r+tank.row, R_4_BLACK+tank.col, BLACK);
    r++;            // increment row counter
    // Draw and erase the proper pixels for row 5
    util_draw_pixel(framePointer, r+tank.row, R_5_GREEN+tank.col, GREEN);
    util_draw_pixel(framePointer, r+tank.row, R_5_BLACK+tank.col, BLACK);
    r++;            // increment row counter
    // Draw and erase the proper pixels for row 6
```

```
    util_draw_pixel(framePointer, r+tank.row, R_6_GREEN+tank.col, GREEN);
    util_draw_pixel(framePointer, r+tank.row, R_6_BLACK+tank.col, BLACK);
    r++;            // increment row counter
    // Draw and erase the proper pixels for row 07
    util_draw_pixel(framePointer, r+tank.row, R_7_GREEN+tank.col, GREEN);
    util_draw_pixel(framePointer, r+tank.row, R_7_BLACK+tank.col, BLACK);
}

// This creates a shell and initially draws it to the screen
void tank_fire(uint32_t * framePointer){
    if(!tank_shell.alive){          // Only go on if our shell is dead
        sound_init_tank(); // tank shoot sound
        tank_shell.col = tank.col;  // give it
        tank_shell.row = tank.row;  // a location
        tank_shell.alive = true;    // make it alive!

        // Tank bullet is 3 pixels long.
        int row;
        // So go through all 3 pixels and draw them to the screen!
        for(row = tank_shell.row-1;row>tank_shell.row-SHELL_LENGTH;row--){
            util_draw_pixel(framePointer,row,SHELL_COL_OFFSET+tank_shell.col,WHITE);
        }
    }
}

// This moves the shell up the screen
void tank_update_bullet(uint32_t * framePointer){
    if(!tank_shell.alive){
        return;                 // Do nothing if no living bullet
    }


    if(tank_shell.row<SHELL_DEATH_ROW){         // If shell is off the screen
        tank_kill_bullet(framePointer);
    }else if(bunkers_detect_collision(tank_shell.row-SHELL_LENGTH,
            tank_shell.col+SHELL_COL_OFFSET, false)){
        tank_kill_bullet(framePointer);
    } else if(aliens_detect_collision(tank_shell.row-SHELL_LENGTH,
            tank_shell.col+SHELL_COL_OFFSET)){
        tank_kill_bullet(framePointer);
    } else if(mother_ship_detect_collision(tank_shell.row-SHELL_LENGTH,
            tank_shell.col+SHELL_COL_OFFSET)){
        tank_kill_bullet(framePointer);
    } else {        // Don't do anything if it's dead
        tank_shell.row -= 1;            // move it up
        // Erase the lowest pixel, and draw one higher up.
        util_draw_pixel(framePointer,tank_shell.row-SHELL_LENGTH,SHELL_COL_OFFSET+tank_sh
ell.col, WHITE);
        util_draw_pixel(framePointer,tank_shell.row,SHELL_COL_OFFSET+tank_shell.col,
BLACK);
    }
}

// This just erases the bullet.
void tank_kill_bullet(uint32_t * framePointer){
#define BULLET_PIXEL_1 -1
#define BULLET_PIXEL_2 -2
#define BULLET_PIXEL_3 -3
```

```
    tank_shell.alive = false;   // Kill it
    util_draw_pixel(framePointer,tank_shell.row+BULLET_PIXEL_1,
            SHELL_COL_OFFSET+tank_shell.col, BLACK);            // Black
    util_draw_pixel(framePointer,tank_shell.row+BULLET_PIXEL_2,
            SHELL_COL_OFFSET+tank_shell.col, BLACK);            // Out all
    util_draw_pixel(framePointer,tank_shell.row+BULLET_PIXEL_3,
            SHELL_COL_OFFSET+tank_shell.col, BLACK);            // 3 pixels
}

// If something hit our tank?
bool tank_detect_collision(uint32_t row, uint32_t col){
    if(row == tank.row && col > tank.col && col < tank.col+TANK_WIDTH){
        interface_kill_tank();
        tank_die();
        sound_init_explosion(); // tank expode
        return true;
    }
    return false;
}

// Kills our tank. Also, seizes hold of the program so nothing else happens
void tank_die(){
    uint32_t row, col, i;   // init loop vars
    for(i = 0; i < TANK_DEATH_TIME ; i++){
        for(row=0;row<TANK_DEATH_HEIGHT;row++){     // Go through tank x pixels
            for(col=0;col<TANK_DEATH_WIDTH;col++){  // and tank y pixels
                if ((tankDeath1[row] & (1<<(TANK_DEATH_WIDTH-col-1)))) {    // If a pixel
                    util_draw_pixel(frame,
row+tank.row+EXPLOSION_ROW_OFFSET,col+tank.col+EXPLOSION_COL_OFFSET,GREEN); //  Draw the
pixel
                }
                else{
                    util_draw_pixel(frame,
row+tank.row+EXPLOSION_ROW_OFFSET,col+tank.col+EXPLOSION_COL_OFFSET,BLACK); //  Draw the
pixel
                }
            }
        }
        for(row=0;row<TANK_DEATH_HEIGHT;row++){     // Go through tank x pixels
            for(col=0;col<TANK_DEATH_WIDTH;col++){  // and tank y pixels
                if ((tankDeath2[row] & (1<<(TANK_DEATH_WIDTH-col-1)))) {    // If a pixel
                    util_draw_pixel(frame,
row+tank.row+EXPLOSION_ROW_OFFSET,col+tank.col+EXPLOSION_COL_OFFSET,GREEN); //  Draw the
pixel
                }
                else{
                    util_draw_pixel(frame,
row+tank.row+EXPLOSION_ROW_OFFSET,col+tank.col+EXPLOSION_COL_OFFSET,BLACK); //  Draw the
pixel
                }
            }
        }
    }
    for(row=0;row<TANK_DEATH_HEIGHT;row++){     // Go through tank x pixels
        for(col=0;col<TANK_DEATH_WIDTH;col++){  // and tank y pixels
            if ((tankDeath2[row] & (1<<(TANK_DEATH_WIDTH-col-1)))) {    // If a pixel
                util_draw_pixel(frame,
```

```
row+tank.row+EXPLOSION_ROW_OFFSET,col+tank.col+EXPLOSION_COL_OFFSET,BLACK); //  Draw the
pixel
                util_draw_pixel(frame,
row+tank.row+EXPLOSION_ROW_OFFSET,col+tank.col+EXPLOSION_COL_OFFSET,BLACK); //  Draw the
pixel
            }
        }
    }
    tank_draw(frame, false);    // Releases the program and redraws the tank.
}
```