Lab 4 - Taylor Cowley and Andrew Okazaki

Chapter 2: Game Console and Engine

- Section 2.1: Game Console
    - 2.1.a: Diligent ATLYS Board
    - 2.1.b: Xilinx Spartan-6 and Microblaze
    - 2.1.c: System Organization
- Section 2.2: Game Engine
    - 2.2.a: Game Engine (Main Game Loop)
    - 2.2.b: Meeting the Game Specifications
- Section 2.3: Application Programming Interfaces

Section 2.1: Game Console

2.1.a: Diligent ATLYS Board

Xilinx's *ATLYS* board "is a complete, ready-to-use digital circuit development platform based on a Xilinx® Spartan®-6 LX45 FPGA." [store.digilentinc.com] it has a large number of peripherals built-in, including switches, buttons, LEDs, several HDMI ports, Ethernet, sound, USB, and a VHDC connector for GPIO. It is a "legacy product"; replaced by the *Nexys Video* product line.

2.1.b: Xilinx Spartan-6 and Microblaze

Xilinx's *Spartan-6* FPGAs are marketed as a low-cost FPGA, "Spartan®-6 devices are the most cost-optimized FPGAs, offering industry leading connectivity features such as high logic-to-pin ratios, small form-factor packaging, and a diverse number of supported I/O protocols." [www.xilinx.com] They are therefore good for applications with low-power necessities and high-volume. The *Spartan-6* product line is currently being replaced by Xilinx's *Artix-7* line.

Xilinx developed a soft microprocessor core called the *MicroBlaze* for their FPGAs. Being a soft microprocessor, it is implemented entirely in the general logic of the FPGA. Hence it can be very customizable for the specific application. It is a RISC-based architecture, and with few exceptions can issue a new instruction every cycle, maintaining single-cycle throughput most of the time.

2.1.c: System Organization

For our game space invaders on the Digilent ATLYS Board we were running our program from the DDR. This enabled us to use more memory because we ran out of space in the BRAMs. The resolution of our game was VGA resolution (640 x 480) pixels and was outputted via a HDMI cable to a screen. To input and communicate with the user we used the UART. With the UART we were able to input from the keyboard commands to either move the tank, fire bullets or any on screen action. By the end of the lab we were able to use an interrupt and buttons to be able to control all user I/O. The left button on the board was able to move the tank left and right button moved the tank

right. The tank fired when the center button was pushed.

Section 2.2: Game Engine

2.2.a: Game Engine (Main Game Loop)

In the game Space invaders there are a many objects that we split into separate files. Those files included:

Tank

In the tank file we had full control over the movement which was comprised of panning left and right. The tank would as well shoot a bullet. The bullets position was stored in the tank file and could be shared with the aliens file so aliens could detect a hit. The bullet would as well store a flag to tell if it was alive or dead because there is only one bullet aloud on screen. The tank could be shot as well when the tank was shot there is an animation stored in the file. This was the implementation of the tank file

Aliens

In the alien file the aliens would move. The aliens could move left and right as well when they hit the right or left side of the screen move down. Aliens could fire four bullets and these bullet locations were saved in the aliens file. When the aliens would die an animation would show and the score would be updated. This was the implementation in the alien file.

Bunkers

Bunkers would initialize which would place four bunkers to the screen.

Bunkers would not move but would degrade if they were hit. As well they would be destroyed if the bunkers were hit by the alien. These were the implementations of the bunker file.

Mother ship

In the game a mother ship would randomly show up and move across the top of the top of the playing screen. If the mother ship was hit it would show a random point value and add it to the games total points. These were the implementations of the mother ship file.

Interface

The interface file was tasked with drawing the games interface such as the score and the tank life indication. It would as well draw the game over screen and the win screen. These were the implementations of the Interface file.

Main

In the main file the program was able to call the many functions from these files to build a running game. In main we also implemented an interrupt and the boards buttons.

2.2.b: Meeting the Game Specifications

To pass and meet the game specification we had to pay close attention to our game. Some requirements that we were looking at was that the game started and ran smoothly with no hieroglyphs, flickering and was not slow. We also paid close attention where we were drawing objects and were we were deleting objects. The specs that a little easier to see was that our game performed like the game Space Invaders with a

score and tank firing and aliens dyeing.

Section 2.3: Application Programming Interfaces

Space invaders has several files of code talking to each other. This is evident in looking at the .h files. When the aliens move and when the aliens and tank update their bullets, they call detect_collision in bunkers, aliens, and tank to see if they hit them.

Also, every file of code draws pixels to the screen, so there is a utils.h file that has a “draw_pixel” function that performs the proper logic for increasing the resolution to the big screen.

The game engine uses built-in microblaze functions to initialize, activate, and read interrupts from the timer and buttons. When the game is over, either when the player fails or wins, the program calls exit().

Timing and Memory Report

To determine cpu usage, a counter was set to run in our while(1) loop for 40 seconds while no game logic was executing. This gives a base estimate of how often the cpu ticks during that time period. Then the same counter was left running, but the interrupts and all the game logic were also running. The counter with the game logic only went up to 75.03% of the game-logic-free counter, showing 75% idle cpu. So we determine that Space Invaders has 25% CPU usage.

For memory usage, when Space Invaders builds, it outputs this report:

Invoking: MicroBlaze Print Size
mb-size spaceInvadersLab4.elf |tee “spaceInvadersLab4.elf.size”

text	data	bss	dec	hex	filename
50838	1496	6570	58904	e618	spaceInvadersLab4.elf

Bug Report

Lab four brought similar bugs that we faced in lab 3. Some of those similar errors were assigning the correct screen position to our objects. Such as the bunkers, when they were hit it erodes that square of the bunker. Finding the location of that square gave us problems. However we were able to see that we were shifting squares causing bugs within the bunkers.

An error that took us a longer time to figure out was the process of drawing an image for a short period of time then taking it away. This process would happen whenever an alien was shot; an explosion would be drawn to the screen then shortly deleted. In the first place we would always draw black to the screen on the next alien move. This was the wrong way however, because when we would reach the bunkers there would be a black box drawn where dead aliens intersected with the bunkers. To solve this we built a flag with in the alien struct to tell if the alien was exploding. This enabled us to draw black only if the alien was exploding.

```c
/*
 * helloworld.c: simple test application
 * Currently used to test lab 3 for Space Invaders.
 * Taylor Cowley and Andrew Okazaki
 */

#include <stdio.h>
#include <stdint.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "tank.h"
#include "interface.h"
#include "aliens.h"
#include "bunkers.h"
#include "mother_ship.h"
#include "util.h"

#include "xgpio.h"
#include "mb_interface.h"
#include "xintc_l.h"

#define DEBUG

#define SCREEN_RES_X 640    // Our screen resolution is 640 * 480
#define SCREEN_RES_Y 480    // Our screen resolution is 640 * 480
#define BLACK 0x00000000    // Hex value for black
#define BLUE  0x2222FF

#define ONE_SECOND  100     // 100 ticks in a second
#define HALF_SECOND 50      // 50 ticks in half a second
#define QUARTER_SECOND 25   // 25 ticks in a quarter second
#define EIGHTH_SECOND 12    // 12 ticks in an eigth second
#define TENTH_SECOND 10     // 10 ticks in a tenth second
#define TWENTIETH_SECOND 5  // 5 ticks in a twentieth second
#define SUPER_FAST 2        // super fast

#define MOTHER_SHIP_SPEED TENTH_SECOND      // Mother ship moves slowly
#define MOTHER_SHIP_SPAWN_CONSTANT 1000     // Mother ship spawns infrequently
#define ALIEN_SHOT_SPAWN_CONSTANT 100       // Aliens shoot frequently
#define ALIEN_MOVE_SPEED HALF_SECOND        // aliens move very slowly

#define BUTTON_UP       0x10    // Constants for button masks
#define BUTTON_DOWN     0x4
#define BUTTON_LEFT     0x8
#define BUTTON_RIGHT    0x2
#define BUTTON_CENTER   0x1

void print(char *str);          // print exists!

#define FRAME_BUFFER_0_ADDR 0xC1000000  // Starting location in DDR

//-----------------------------
void timer_interrupt_handler();
void pb_interrupt_handler();
```

```
void interttupt_handler_dispatcher();
//-----------------------------


XGpio gpLED;  // This is a handle for the LED GPIO block.
XGpio gpPB;   // This is a handle for the push-button GPIO block.
uint32_t* framePointer0 = (uint32_t*) FRAME_BUFFER_0_ADDR;
int32_t currentButtonState;     // Current button being pressed
int32_t mother_ship_points;


void timer_interrupt_handler(){
    static uint32_t timerCount;                 // Timer for timing
    static uint32_t mother_ship_move_counter;   // Timer for mother ship
    tank_update_bullet(framePointer0);          // update all bullets
    aliens_update_bullets(framePointer0);       // update all bullets

    timerCount++;                               // Increment all counters
    mother_ship_move_counter++;
    mother_ship_points++;

    int32_t r = rand();
    if(r%ALIEN_SHOT_SPAWN_CONSTANT == 0){
        alien_missle(framePointer0);    // Make the aliens fire
    }
    if(r%MOTHER_SHIP_SPAWN_CONSTANT == 0){
        mother_ship_spawn();            // mother ship spawns!
    }
    if(mother_ship_move_counter >= MOTHER_SHIP_SPEED){  // MS moves
        mother_ship_move_counter = 0;
        mother_ship_move();
    }
    if(mother_ship_points > TENTH_SECOND){
        mother_ship_points = 0;         // Mother ship points will display
        mother_ship_points_blink();
    }
    if(timerCount >= 5 ){
        timerCount = 0;
        aliens_move(framePointer0); // move the aliens
    }

    // Now to check the buttons.
    if(currentButtonState & BUTTON_LEFT){
        tank_move_left(framePointer0);      // Moving the tank left
    }
    if(currentButtonState & BUTTON_RIGHT){
        tank_move_right(framePointer0);     // Moving the tank right
    }
    if(currentButtonState & BUTTON_CENTER){
        tank_fire(framePointer0);           // Fire the tank!
    }
    if(currentButtonState & BUTTON_UP){     // Not functional yet
    }
}
void pb_interrupt_handler(){
    XGpio_InterruptGlobalDisable(&gpPB);    // Can't be interrupted by buttons
    xil_printf("Button Interrupt\n\r");
    currentButtonState = XGpio_DiscreteRead(&gpPB, 1);
```

```
    // Time to clear the interrupt and reenable GPIO interrupts
    XGpio_InterruptClear(&gpPB, 0xFFFFFFFF);
    XGpio_InterruptGlobalEnable(&gpPB);
}

// Main interrupt handler, queries interrupt controller to see what peripheral
// fired the interrupt and then dispatches the corresponding interrupt handler.
// This routine acks the interrupt at the controller level but the peripheral
// interrupt must be ack'd by the dispatched interrupt handler.
// Question: Why is timer_interrupt_handler() called after ack'ing controller
// but pb_interrupt_handler() is called before ack'ing the interrupt controller?
void interrupt_handler_dispatcher(void* ptr) {
    int intc_status = XIntc_GetIntrStatus(XPAR_INTC_0_BASEADDR);
    // Check the FIT interrupt first.
    if (intc_status & XPAR_FIT_TIMER_0_INTERRUPT_MASK){
        XIntc_AckIntr(XPAR_INTC_0_BASEADDR, XPAR_FIT_TIMER_0_INTERRUPT_MASK);
        timer_interrupt_handler();  // It was a timer interrupt! call that fn
    }
    // Check the push buttons.
    if (intc_status & XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK){
        pb_interrupt_handler();     // It was a button interrupt!
        XIntc_AckIntr(XPAR_INTC_0_BASEADDR, // Acknowledge the interrupt
                XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK);
    }
}

void init_interrupts(void){
    int32_t success;
    print("\n\rHello . Let's have a fun \e[31m\e[1mtime \e[21m\e[0m\n\r");
    success = XGpio_Initialize(&gpPB, XPAR_PUSH_BUTTONS_5BITS_DEVICE_ID);
    // Set the push button peripheral to be inputs.
    XGpio_SetDataDirection(&gpPB, 1, 0x0000001F);
    // Enable the global GPIO interrupt for push buttons.
    XGpio_InterruptGlobalEnable(&gpPB);
    // Enable all interrupts in the push button peripheral.
    XGpio_InterruptEnable(&gpPB, 0xFFFFFFFF);
    // Register the interrupt handler
    microblaze_register_handler(interrupt_handler_dispatcher, NULL);
    // And enable interrupts
    XIntc_EnableIntr(XPAR_INTC_0_BASEADDR,
            (XPAR_FIT_TIMER_0_INTERRUPT_MASK |
                    XPAR_PUSH_BUTTONS_5BITS_IP2INTC_IRPT_MASK));
    // Master the enable
    XIntc_MasterEnable(XPAR_INTC_0_BASEADDR);
    // And enable again
    microblaze_enable_interrupts();
}

int main() {
    init_platform();                    // Necessary for all programs.
    init_interrupts();
    int Status;                         // Keep track of success/failure of system
function calls.
    XAxiVdma videoDMAController;
    // There are 3 steps to initializing the vdma driver and IP.
    // Step 1: lookup the memory structure that is used to access the vdma driver.
    XAxiVdma_Config * VideoDMAConfig = XAxiVdma_LookupConfig(XPAR_AXI_VDMA_0_DEVICE_ID);
    // Step 2: Initialize the memory structure and the hardware.
```

```
    if(XST_FAILURE == XAxiVdma_CfgInitialize(&videoDMAController,
VideoDMAConfig, XPAR_AXI_VDMA_0_BASEADDR)) {
        xil_printf("VideoDMA Did not initialize.\r\n");
    }
    // Step 3: (optional) set the frame store number.
    if(XST_FAILURE ==  XAxiVdma_SetFrmStore(&videoDMAController, 2, XAXIVDMA_READ)) {
        xil_printf("Set Frame Store Failed.");
    }
    // Initialization is complete at this point.

    // Setup the frame counter. We want two read frames. We don't need any write frames
but the
    // function generates an error if you set the write frame count to 0. We set it to 2
    // but ignore it because we don't need a write channel at all.
    XAxiVdma_FrameCounter myFrameConfig;
    myFrameConfig.ReadFrameCount = 2;
    myFrameConfig.ReadDelayTimerCount = 10;
    myFrameConfig.WriteFrameCount =2;
    myFrameConfig.WriteDelayTimerCount = 10;
    Status = XAxiVdma_SetFrameCounter(&videoDMAController, &myFrameConfig);
    if (Status != XST_SUCCESS) {
        xil_printf("Set frame counter failed %d\r\n", Status);
        if(Status == XST_VDMA_MISMATCH_ERROR)
            xil_printf("DMA Mismatch Error\r\n");
    }
    // Now we tell the driver about the geometry of our frame buffer and a few other
things.
    // Our image is 480 x 640.
    XAxiVdma_DmaSetup myFrameBuffer;
    myFrameBuffer.VertSizeInput = 480;    // 480 vertical pixels.
    myFrameBuffer.HoriSizeInput = 640*4;  // 640 horizontal (32-bit pixels).
    myFrameBuffer.Stride = 640*4;         // Dont' worry about the rest of the values.
    myFrameBuffer.FrameDelay = 0;
    myFrameBuffer.EnableCircularBuf=1;
    myFrameBuffer.EnableSync = 0;
    myFrameBuffer.PointNum = 0;
    myFrameBuffer.EnableFrameCounter = 0;
    myFrameBuffer.FixedFrameStoreAddr = 0;
    if(XST_FAILURE == XAxiVdma_DmaConfig(&videoDMAController, XAXIVDMA_READ,
&myFrameBuffer)) {
        xil_printf("DMA Config Failed\r\n");
    }
    // We need to give the frame buffer pointers to the memory that it will use. This
memory
    // is where you will write your video data. The vdma IP/driver then streams it to the
HDMI
    // IP.
    myFrameBuffer.FrameStoreStartAddr[0] = FRAME_BUFFER_0_ADDR;
    myFrameBuffer.FrameStoreStartAddr[1] = FRAME_BUFFER_0_ADDR + 4*640*480;

    if(XST_FAILURE == XAxiVdma_DmaSetBufferAddr(&videoDMAController, XAXIVDMA_READ,
            myFrameBuffer.FrameStoreStartAddr)) {
        xil_printf("DMA Set Address Failed Failed\r\n");
    }
    // Print a sanity message if you get this far.
    xil_printf("Woohoo! I made it through initialization.\n\r");
    // Now, let's get ready to start displaying some stuff on the screen.
    // The variables framePointer and framePointer1 are just pointers to the base address
```

```
    // of frame 0 and frame 1.
    uint32_t* framePointer0 = (uint32_t*) FRAME_BUFFER_0_ADDR;
    // Just paint some large red, green, blue, and white squares in different
    // positions of the image for each frame in the buffer (framePointer0 and
framePointer1).
    int row=0, col=0;
    for( row=0; row<SCREEN_RES_Y; row++) {
        for(col=0; col<SCREEN_RES_X; col++) {
            framePointer0[row*SCREEN_RES_X + col] = BLACK;
        }
    }

    bunkers_init(framePointer0);            // Init the bunkers
    tank_init();                            // initialize the tank
    tank_draw(framePointer0, false);        // draw the tank
    interface_init_board(framePointer0);    // draw the tanks at the top
    aliens_init(framePointer0);             // initialize aliens
    mother_ship_init(framePointer0);        // Init the mother ship

    // This tells the HDMI controller the resolution of your display (there must be a
better way to do this).
    XIo_Out32(XPAR_AXI_HDMI_0_BASEADDR, 640*480);

    // Start the DMA for the read channel only.
    if(XST_FAILURE == XAxiVdma_DmaStart(&videoDMAController, XAXIVDMA_READ)){
        xil_printf("DMA START FAILED\r\n");
    }
    int frameIndex = 0;
    // We have two frames, let's park on frame 0. Use frameIndex to index them.
    // Note that you have to start the DMA process before parking on a frame.

    if (XST_FAILURE == XAxiVdma_StartParking(&videoDMAController, frameIndex,
XAXIVDMA_READ)) {
        xil_printf("vdma parking failed\n\r");
    }
    char input;
    srand((unsigned)time( NULL ));

    xil_printf("Are we getting here?\n\r");
    while(1){       // This doesn't need to be here no more
        //aliens_move(framePointer0);   // move the aliens
        tank_update_bullet(framePointer0);  // update all bullets
        aliens_update_bullets(framePointer0);   // update all bullets
        //interface_increment_score(framePointer0,0);
        input = getchar();
        switch(input){
        case '4':
            tank_move_left(framePointer0);      // move the tank left
            break;
        case '6':
            tank_move_right(framePointer0);     // move the tank right
            break;
        case '8':
            mother_ship_spawn();
            break;
        case '2':
            interface_kill_tank();
            interface_increment_score(1);
```

```
            //aliens_kill(framePointer0);   // Kill an alien
            break;
        case '5':
            tank_fire(framePointer0);       // Make the tank fire
            break;
        case '3':
            alien_missle(framePointer0);    // Make the aliens fire
            break;
        case'9':
            mother_ship_move();
            break;
        case '7':
            break;
        }
    }
    cleanup_platform();
    return 0;
}
```

```c
/*
 * aliens.h
 *  Taylor Cowley and Andrew Okazaki
 */

#include <stdbool.h>
#include <stdint.h>
#ifndef ALIENS_H_
#define ALIENS_H_


#endif /* ALIENS_H_ */

void aliens_init(uint32_t * framePointer);  // Initializes the aliens
void aliens_move(uint32_t * framePointer);  // Moves the aliens
void aliens_left(uint32_t * framePointer);  // Moves aliens left
void aliens_right(uint32_t * framePointer); // Move aliens right
void aliens_kill(uint32_t * framePointer);  // Kills a random alien
void alien_missle(uint32_t * framePointer); // Shoots an alien bullet
void aliens_update_bullets(uint32_t * framePointer);    // Updates the bullets
bool aliens_detect_collision(uint32_t row,uint32_t col);// Detect collision w me
```

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

#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "util.h"
#include <stdbool.h>
#include <stdint.h>
#include "bunkers.h"
#include "tank.h"           // required to tell if our bullets hit him.
#include "interface.h"      //required to update the score
#define ALIEN_HEIGHT 8      // Aliens are 8 pixels tall
#define ALIEN_WIDTH 12      // Aliens are 12 pixels wide
#define ALIEN_COLUMNS 11    // 11 columns of aliens
#define BULLET_HEIGHT 5     // Bullets are 5 pixels tall
#define TOP_TOTAL 11        // 11 aliens in top group
#define LOC_ALIEN_ONE 50    // Pixel where the first alien is
#define MIDDLE_TOTAL 22     // There are 22 total middle aliens
#define BOTTOM_TOTAL 22     // There are 22 total bottom aliens
#define ALIEN_NUM_BULLETS 4 // Aliens can have up to 4 bullets at a time
#define ALIEN_NUM_BULLET_TYPES 2// Aliens have 2 types of bullets to choose from
#define BAD_ADDRESS -1      // Nothing exists at screen address -1
#define MOVE_DOWN_PIXELS 15 // When the aliens move down, they do so 15 pixels
#define LEFT_BOUNDRY    11  // Aliens cannot go more left than this
#define RIGHT_BOUNDRY   307 // Aliens cannot go more right than this
#define BULLET_COL_OFFSET 6 // Bullets appear 11 more right than their alien
#define BULLET_ROW_OFFSET 11// Bullets appear more down than their alien
#define SCREEN_LENGTH   320 // Our screen is 320 pixels wide
#define SCREEN_HEIGHT   240 // Our screen is 240 pixels tall
#define SCREEN_RES_X    640 // Our screen RESOLUTION is 640 pixels wide
#define SCREEN_RES_Y    480 // Our screen RESOLUTION is 480 pixels tall
#define WHITE 0xFFFFFFF     // These
#define BLACK 0x0000000     // are colors
#define RED 0xFFF0000
#define WORD_WIDTH 12
#define TOP_POINTS  40      // top alien amount of points given if killed
#define MIDDLE_POINTS 20    // middle alien amount of points given if killed
#define BOTTOM_POINTS 10    // bottom alien amount of points given if killed

// Packs each horizontal line of the figures into a single 32 bit word.
#define
packWord32(b31,b30,b29,b28,b27,b26,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) \
        ((b31 << 31) | (b30 << 30) | (b29 << 29) | (b28 << 28) | (b27 << 27) | (b26 <<
26) | (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 ) )
```

```
#define packword12(b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0) \
        ((b11 << 11) | (b10 << 10) | (b9  << 9 ) | (b8  << 8 ) | (b7  << 7 ) | (b6  << 6
 ) \
                | (b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) |
(b0  << 0 ) )

// -------------------------------------------------
// The following static const ints define the aliens
// We have 3 types of aliens with 2 poses each
const int deadAlien[ALIEN_HEIGHT] =
{
        packword12(0,0,0,0,0,1,0,1,0,0,0,0),
        packword12(0,1,0,1,0,0,0,1,0,0,1,0),
        packword12(0,0,1,0,0,1,0,0,0,1,0,0),
        packword12(0,0,0,0,0,0,0,0,0,0,0,0),
        packword12(0,1,1,1,0,1,0,1,1,1,0,0),
        packword12(0,0,0,0,0,0,0,0,0,0,1,0),
        packword12(0,0,1,0,0,1,0,0,1,0,0,0),
        packword12(0,1,0,1,0,1,0,0,0,1,0,0) };
static const int32_t alien_top_in_12x8[ALIEN_HEIGHT] = {
        packword12(0,0,0,0,0,1,1,0,0,0,0,0),
        packword12(0,0,0,0,1,1,1,1,0,0,0,0),
        packword12(0,0,0,1,1,1,1,1,1,0,0,0),
        packword12(0,0,1,1,0,1,1,0,1,1,0,0),
        packword12(0,0,1,1,1,1,1,1,1,1,0,0),
        packword12(0,0,0,1,0,1,1,0,1,0,0,0),
        packword12(0,0,1,0,0,0,0,0,0,1,0,0),
        packword12(0,0,0,1,0,0,0,0,0,1,0,0,0) };
static const int32_t alien_top_out_12x8[ALIEN_HEIGHT] = {
        packword12(0,0,0,0,0,1,1,0,0,0,0,0),
        packword12(0,0,0,0,1,1,1,1,0,0,0,0),
        packword12(0,0,0,1,1,1,1,1,1,0,0,0),
        packword12(0,0,1,1,0,1,1,0,1,1,0,0),
        packword12(0,0,1,1,1,1,1,1,1,1,0,0),
        packword12(0,0,0,0,0,1,0,0,1,0,0,0,0),
        packword12(0,0,0,1,0,1,1,0,1,0,0,0),
        packword12(0,0,1,0,1,0,0,1,0,1,0,0) };
static const int32_t alien_middle_in_12x8[ALIEN_HEIGHT] = {
        packword12(0,0,0,1,0,0,0,0,0,1,0,0),
        packword12(0,0,0,0,1,0,0,0,1,0,0,0),
        packword12(0,0,0,1,1,1,1,1,1,1,0,0),
        packword12(0,0,1,1,0,1,1,1,0,1,1,0),
        packword12(0,1,1,1,1,1,1,1,1,1,1,1),
        packword12(0,1,1,1,1,1,1,1,1,1,1,1),
        packword12(0,1,0,1,0,0,0,0,0,1,0,1),
        packword12(0,0,0,0,1,1,0,1,1,0,0,0) };
static const int32_t alien_middle_out_12x8[] = {
        packword12(0,0,0,1,0,0,0,0,0,1,0,0),
        packword12(0,1,0,0,1,0,0,0,1,0,0,1),
        packword12(0,1,0,1,1,1,1,1,1,1,0,1),
        packword12(0,1,1,1,0,1,1,1,0,1,1,1),
        packword12(0,1,1,1,1,1,1,1,1,1,1,1),
        packword12(0,0,1,1,1,1,1,1,1,1,1,0),
        packword12(0,0,0,1,0,0,0,0,0,1,0,0),
        packword12(0,0,1,0,0,0,0,0,0,0,1,0) };
static const int32_t alien_bottom_in_12x8[ALIEN_HEIGHT] = {
        packword12(0,0,0,0,1,1,1,1,0,0,0,0),
        packword12(0,1,1,1,1,1,1,1,1,1,1,0),
```

```
        packword12(1,1,1,1,1,1,1,1,1,1,1,1),
        packword12(1,1,1,0,0,1,1,0,0,1,1,1),
        packword12(1,1,1,1,1,1,1,1,1,1,1,1),
        packword12(0,0,1,1,1,0,0,1,1,1,0,0),
        packword12(0,1,1,0,0,1,1,0,0,1,1,0),
        packword12(0,0,1,1,0,0,0,0,1,1,0,0) };
static const int32_t alien_bottom_out_12x8[] = {
        packword12(0,0,0,0,1,1,1,1,0,0,0,0),
        packword12(0,1,1,1,1,1,1,1,1,1,1,0),
        packword12(1,1,1,1,1,1,1,1,1,1,1,1),
        packword12(1,1,1,0,0,1,1,0,0,1,1,1),
        packword12(1,1,1,1,1,1,1,1,1,1,1,1),
        packword12(0,0,0,1,1,0,0,1,1,0,0,0),
        packword12(0,0,1,1,0,1,1,0,1,1,0,0),
        packword12(1,1,0,0,0,0,0,0,0,0,1,1) };
// End of the const ints that define the alien pixels
// --------------------------------------------------

// ------------------------------------------------------------
// These are our internal methods, used only by ourselves
// Draws the aliens on the screen - top, middle, and bottom aliens
void build_tops(uint32_t * framePointer, const int32_t alien_middle[], bool erase);
void build_middle(uint32_t * framePointer, const int32_t alien_middle[], bool erase);
void build_bottom(uint32_t * framePointer, const int32_t alien_bottom[], bool
forceUpdate);
// Fire a bullet from either a top, middle, or bottom alien
int32_t fire_bottom(uint32_t * framePointer, int32_t r);
int32_t fire_middle(uint32_t * framePointer, int32_t r);
int32_t fire_top(uint32_t * framePointer, int32_t r);
// Checks to see whether our aliens are currently capable of shooting
bool can_aliens_shoot();
// Draws a bullet on the screen
void draw_bullet(uint32_t * framePointer, int32_t bullet, uint32_t color);
// We like our aliens black
void aliens_blacken(uint32_t * framePointer, uint32_t row, uint32_t col);
// Have the aliens destroyed us?
void aliens_detect_game_over();
// End internal method declarations
// ------------------------------------------------------------

// These structs hold all of our aliens.
struct top { // Struct for our top aliens
    int32_t row;
    int32_t col;bool alive; // alien has row, column, and alive?
    bool exploding;
} top[TOP_TOTAL];

struct middleAlien { // Struct for our middle aliens
    int32_t row;
    int32_t col;bool alive; // alien has row, column, and alive?
    bool exploding;
} middleAlien[MIDDLE_TOTAL];

struct bottomAlien { // Struct for our bottom aliens
    int32_t row;
    int32_t col;bool alive; // alien has row, column, and alive?
    bool exploding;
} bottomAlien[MIDDLE_TOTAL];
```

```
// aliens can have two types of bullet: cross and lightning
// cross 0 and 3 are identical
typedef enum {
    cross0, cross1, cross2, cross3, lightning0, lightning1
} bullet_type;
struct alien_bullet { // Struct that holds our aliens' bullets
    int32_t row;
    int32_t col;bool alive; // Bullets have coordinates and alive?
    bullet_type bullet_type; // Bullets also have a type.
} alien_bullet[ALIEN_NUM_BULLETS];

int32_t alien_count; // a count of how many aliens are alive
int32_t how_many_aliens_left;
uint32_t * frame; // framePointer
//initialize all of the aliens by setting values contained in struct's and printing
aliens to the screen
void aliens_init(uint32_t * framePointer) {
#define ALIEN_TOP_ROW_INIT 30               // Where
#define ALIEN_MIDDLE_ROW_INIT 45            // the
#define ALIEN_MIDDLE2_ROW_INIT 60           // aliens
#define ALIEN_BOTTOM_ROW_INIT 75            // are
#define ALIEN_BOTTOM2_ROW_INIT 90           // initialized to
#define ALIEN_SPACING 15                    // Spacing between aliens
    //local variables, loc is the starting location of alien one on the screen
    int32_t i, loc = LOC_ALIEN_ONE;
    frame = framePointer;

    //loops through one row of aliens
    for (i = 0; i < ALIEN_COLUMNS; i++) {
        top[i].row = ALIEN_TOP_ROW_INIT; //set the row of alien tops to 30
        top[i].col = loc;//sets the column of alien tops
        top[i].alive = true;//sets the alien is alive flag
        top[i].exploding = false;

        middleAlien[i].row = ALIEN_MIDDLE_ROW_INIT; //middle aliens
        middleAlien[i].col = loc;//sets column of first row of middle aliens
        middleAlien[i].alive = true;//sets first row of middle aliens to alive
        middleAlien[i].exploding = false;
        middleAlien[i + ALIEN_COLUMNS].row = ALIEN_MIDDLE2_ROW_INIT;//sets middle
        middleAlien[i + ALIEN_COLUMNS].col = loc;//sets column second row middle
        middleAlien[i + ALIEN_COLUMNS].alive = true;//sets second row middle alive
        middleAlien[i + ALIEN_COLUMNS].exploding = false;

        bottomAlien[i].row = ALIEN_BOTTOM_ROW_INIT;//sets bottom aliens
        bottomAlien[i].col = loc;//sets column of first row of bottom aliens
        bottomAlien[i].alive = true;//sets first row of bottom aliens to alive
        bottomAlien[i].exploding = false;
        bottomAlien[i + ALIEN_COLUMNS].row = ALIEN_BOTTOM2_ROW_INIT;//bottom
        bottomAlien[i + ALIEN_COLUMNS].col = loc;//sets column second row bottom
        bottomAlien[i + ALIEN_COLUMNS].alive = true;//sets second row bottom alive
        bottomAlien[i + ALIEN_COLUMNS].exploding = false;
        loc += ALIEN_SPACING; //controls the column spacing in-between alien
    }

    //now that structs are built draw top, middle, and bottom aliens to screen
    build_tops(framePointer, alien_top_in_12x8, false); // Top
    build_middle(framePointer, alien_middle_in_12x8, false); // Middle
```

```
    build_bottom(framePointer, alien_bottom_in_12x8, false); // Bottom

    how_many_aliens_left = TOP_TOTAL + MIDDLE_TOTAL + BOTTOM_TOTAL;
}

// Draws the top aliens on the screen
void build_tops(uint32_t * framePointer, const int32_t alien_top[], bool erase) {
    uint32_t color = erase ? BLACK : WHITE ;
    int32_t row, col, i; // initialize variables
    for (i = 0; i < TOP_TOTAL; i++) { //loop through top column of aliens
        for (row = 0; row < ALIEN_HEIGHT; row++) { //loop top aliens' pixels row
            int32_t currentRow = row + top[i].row;// current pixel row of alien
            for (col = 0; col < WORD_WIDTH; col++) { //loop alien's pixel col
                int32_t currentCol = col + top[i].col; //current col of alien
                if ((alien_top[row] & (1 << (WORD_WIDTH - col - 1)))
                        && top[i].alive) {
                    // If our alien is alive and has a pixel there, draw it
                    util_draw_pixel(framePointer, currentRow, currentCol,
                            color);
                } else if(top[i].alive){ // otherwise, erase it.
                    util_draw_pixel(framePointer, currentRow, currentCol,BLACK);
                } else if(top[i].exploding){
                    top[i].exploding = false;
                    aliens_blacken(framePointer, currentRow, currentCol);
                }
            }
        }
    }
}

// Draws the middle aliens to the screen
void build_middle(uint32_t * framePointer, const int32_t alien_middle[], bool erase) {
    uint32_t color = erase ? BLACK : WHITE ;
    int32_t row, col, i; // declare our variables
    for (i = 0; i < MIDDLE_TOTAL; i++) { // Looping through all the middle aliens
        for (row = 0; row < ALIEN_HEIGHT; row++) { // Pixel y
            int32_t currentRow = row + middleAlien[i].row;//current pixel row
            for (col = 0; col < WORD_WIDTH; col++) {// Pixel x
                int32_t currentCol = col + middleAlien[i].col;// current col alien
                if ((alien_middle[row] & (1 << (WORD_WIDTH - col - 1)))
                        && middleAlien[i].alive) {
                    // If our alien is alive and has a pixel there, draw it
                    util_draw_pixel(framePointer, currentRow, currentCol,
                            color);
                } else if(middleAlien[i].alive){ // otherwise, erase it.
                    util_draw_pixel(framePointer, currentRow, currentCol,BLACK);
                } else if(middleAlien[i].exploding){
                    middleAlien[i].exploding = false;
                    aliens_blacken(framePointer, currentRow, currentCol);
                }
            }
        }
    }
}

// Draws the bottom aliens to the screen
void build_bottom(uint32_t * framePointer, const int32_t alien_bottom[], bool erase) {
    int32_t row, col, i; // Declare vars
```

```
    uint32_t color = erase ? BLACK : WHITE ;
    for (i = 0; i < BOTTOM_TOTAL; i++) { // Looping through all the bottom aliens
        for (row = 0; row < ALIEN_HEIGHT; row++) { // looping through y pixels
            int32_t currentRow = row + bottomAlien[i].row; // current row
            for (col = 0; col < WORD_WIDTH; col++) { // looping through x pixels
                int32_t currentCol = col + bottomAlien[i].col; // current col
                if ((alien_bottom[row] & (1 << (WORD_WIDTH - col - 1)))
                        && bottomAlien[i].alive) {
                    // If our alien is alive and has a pixel here, draw it
                    util_draw_pixel(framePointer, currentRow, currentCol,
                            color);
                } else if(bottomAlien[i].alive){ // otherwise, erase it.
                    util_draw_pixel(framePointer, currentRow, currentCol,BLACK);
                } else if(bottomAlien[i].exploding){
                    bottomAlien[i].exploding = false;
                    aliens_blacken(framePointer, currentRow, currentCol);
                }
            }
        }
    }
}

// Draws a big, black, rectangle over an alien
void aliens_blacken(uint32_t * framePointer, uint32_t row, uint32_t col){
    int32_t r, c;
    for(r=0;r<ALIEN_HEIGHT;r++){
        for(c=0;c<ALIEN_WIDTH;c++){
            util_draw_pixel(framePointer, r+row, c+col,BLACK);
        }
    }
}

// Does the needful to move the aliens left
void aliens_left(uint32_t * framePointer) {
    int32_t i, row; // Declare loop vars
    for (i = 0; i < MIDDLE_TOTAL; i++) { // Move every single alien LEFT
        if (i < TOP_TOTAL) {
            top[i].col--;
        } // Move the top aliens LEFT
        middleAlien[i].col--; // Move the middle aliens LEFT
        bottomAlien[i].col--; // Move the bottom aliens LEFT
    }
    if (alien_count == 0) { // If aliens are out, make them in
        alien_count = 1;
        build_tops(framePointer, alien_top_in_12x8, false); // Draw top aliens
        build_middle(framePointer, alien_middle_in_12x8, false); // Draw mid aliens
        build_bottom(framePointer, alien_bottom_in_12x8, false); // Draw bot aliens
    } else { // And vice versa
        alien_count = 0;
        build_tops(framePointer, alien_top_out_12x8, false); // Draw top aliens
        build_middle(framePointer, alien_middle_out_12x8, false); // Draw mid aliens
        build_bottom(framePointer, alien_bottom_out_12x8, false); // Draw bot aliens
    }

    for (row = 0; row < ALIEN_HEIGHT; row++) { // For all the alien Y pixels
        for (i = 0; i < MIDDLE_TOTAL; i++) { // For every alien
            // Erase them for the middle and bottom aliens - top is skinnier
            if(bottomAlien[i].alive){
```

```
                util_draw_pixel(framePointer, row + bottomAlien[i].row,
                        WORD_WIDTH + bottomAlien[i].col, BLACK);
            }
            if(middleAlien[i].alive){
                util_draw_pixel(framePointer, row + middleAlien[i].row,
                        WORD_WIDTH + middleAlien[i].col, BLACK);
            }
        }
    }

    // Here we loop through every single dang alien and see if they hit the dang bunkers
    for (i = 0; i < MIDDLE_TOTAL; i++) { // Move every single alien LEFT
        if (i < TOP_TOTAL) {
            if(top[i].alive){
                bunkers_detect_collision(top[i].row,top[i].col,true);
                bunkers_detect_collision(top[i].row+ALIEN_HEIGHT/2,top[i].col,true);
                bunkers_detect_collision(top[i].row+ALIEN_HEIGHT,top[i].col,true);
            }
        } // Move the top aliens LEFT

        if(middleAlien[i].alive){
            bunkers_detect_collision(middleAlien[i].row,middleAlien[i].col,true);
            bunkers_detect_collision(middleAlien[i].row+ALIEN_HEIGHT/2,middleAlien[i].col,
true);
            bunkers_detect_collision(middleAlien[i].row+ALIEN_HEIGHT,middleAlien[i].col,t
rue);
        }

        if(bottomAlien[i].alive){
            bunkers_detect_collision(bottomAlien[i].row,bottomAlien[i].col,true);
            bunkers_detect_collision(bottomAlien[i].row+ALIEN_HEIGHT/2,bottomAlien[i].col,
true);
            bunkers_detect_collision(bottomAlien[i].row+ALIEN_HEIGHT,bottomAlien[i].col,t
rue);
        }
    }


}

// Does the needful to move the aliens right
void aliens_right(uint32_t * framePointer) {
    int32_t i, row; // Declare loop vars
    for (i = 0; i < MIDDLE_TOTAL; i++) { // Move every single alien RIGHT
        if (i < 11) {
            top[i].col += 1;
        } // Move top aliens RIGHT
        middleAlien[i].col += 1; // Move middle aliens RIGHT
        bottomAlien[i].col += 1; // Move bottom aliens RIGHT
    }

    if (alien_count == 0) { // If aliens are out, make them in
        alien_count = 1;
        build_tops(framePointer, alien_top_in_12x8, false); // Draw top aliens
        build_middle(framePointer, alien_middle_in_12x8, false); // Draw mid aliens
        build_bottom(framePointer, alien_bottom_in_12x8, false); // Draw bot aliens
    } else { // And vice versa
        alien_count = 0;
```

```
        build_tops(framePointer, alien_top_out_12x8, false); // Draw top aliens
        build_middle(framePointer, alien_middle_out_12x8, false); // Draw mid aliens
        build_bottom(framePointer, alien_bottom_out_12x8, false); // Draw bot aliens
    }

    for (row = 0; row < ALIEN_HEIGHT; row++) { // For all the alien Y pixels
        for (i = 0; i < MIDDLE_TOTAL; i++) { // For every alien
            // Erase that column of pixels for mid and bottom. Top not necessary
            if(bottomAlien[i].alive){
                util_draw_pixel(framePointer, row + bottomAlien[i].row,
                        bottomAlien[i].col - 1, BLACK); // Notice it's col-1 bottom
            }
            if(middleAlien[i].alive){
                util_draw_pixel(framePointer, row + middleAlien[i].row,
                        middleAlien[i].col, BLACK);
            }
        }
    }


    // Here we loop through every single dang alien and see if they hit the dang bunkers
    for (i = 0; i < MIDDLE_TOTAL; i++) { // Move every single alien LEFT
        if (i < TOP_TOTAL) {
            if(top[i].alive){
                bunkers_detect_collision(top[i].row,top[i].col+ALIEN_WIDTH,true);
            }
        } // Move the top aliens LEFT

        if(middleAlien[i].alive){
            bunkers_detect_collision(middleAlien[i].row,middleAlien[i].col+ALIEN_WIDTH,tr
ue);
        }

        if(bottomAlien[i].alive){
            bunkers_detect_collision(bottomAlien[i].row,bottomAlien[i].col+ALIEN_WIDTH,tr
ue);
        }
    }
}



// Does the needful when aliens hit the left rail
void hit_left_rail(uint32_t * framePointer) {
    // Erase ALL the aliens.
    build_tops(framePointer, alien_bottom_out_12x8, true);
    build_middle(framePointer, alien_bottom_out_12x8, true);
    build_bottom(framePointer, alien_bottom_out_12x8, true);

    // First we erase the entire top row of alien pixels for moving down.
    int32_t col, row, i; // declare loop vars
    for (row = 0; row < ALIEN_HEIGHT; row++) { // Go through alien pixels Y
        for (col = 0; col < WORD_WIDTH; col++) { // Go through alien pixels X
            if (((alien_top_out_12x8[row] | alien_top_in_12x8[row]) & (1
                    << (WORD_WIDTH - col - 1)))) {// if pixel exists here
                for (i = 0; i < TOP_TOTAL; i++) { // ERASE IT!
                    util_draw_pixel(framePointer, row + top[i].row,
                            col + top[i].col, BLACK);
```

```
                }
            }
        }
    }
    for (i = 0; i < MIDDLE_TOTAL; i++) { // For all the aliens, move them down
        if (i < TOP_TOTAL) {
            top[i].row += MOVE_DOWN_PIXELS;
        } // Move top aliens down
        middleAlien[i].row += MOVE_DOWN_PIXELS; // Move mid aliens down
        bottomAlien[i].row += MOVE_DOWN_PIXELS; // Move bot aliens down
    }
    for (row = 0; row < ALIEN_HEIGHT; row++) { // Now to erase pixels on left side
        for (i = 0; i < MIDDLE_TOTAL; i++) { // For all the middle aliens
            util_draw_pixel(framePointer, row + middleAlien[i].row,
                    middleAlien[i].col, BLACK);// Erase the pixels on the left
        }
    }

}

// Does the needful when aliens hit the right rail
void hit_right_rail(uint32_t * framePointer) {
    // Erase ALL the aliens.
    build_tops(framePointer, alien_bottom_out_12x8, true);
    build_middle(framePointer, alien_bottom_out_12x8, true);
    build_bottom(framePointer, alien_bottom_out_12x8, true);

    // First we erase the entire top row of alien pixels for moving down
    int32_t col, row, i; // Declare loop vars
    for (row = 0; row < ALIEN_HEIGHT; row++) { // Go through alien pixels Y
        for (col = 0; col < WORD_WIDTH; col++) { // Go through alien pixels X
            if (((alien_top_out_12x8[row] | alien_top_in_12x8[row]) & (1
                    << (WORD_WIDTH - col - 1)))) {// if pixel exists here
                for (i = 0; i < TOP_TOTAL; i++) { // Erase it!
                    util_draw_pixel(framePointer, row + top[i].row,
                            col + top[i].col, BLACK);
                }
            }
        }
    }
    for (i = 0; i < MIDDLE_TOTAL; i++) { // For all the aliens, move them down
        if (i < TOP_TOTAL) {
            top[i].row += MOVE_DOWN_PIXELS;

        }// Move top aliens down
        middleAlien[i].row += MOVE_DOWN_PIXELS; // Move mid aliens down
        bottomAlien[i].row += MOVE_DOWN_PIXELS; // Move bot aliens down

        aliens_detect_game_over();
    }
    for (row = 0; row < ALIEN_HEIGHT; row++) { // Now to erase pixels on the right side
        for (i = 0; i < TOP_TOTAL; i++) { // Erase the pixels on the right
            util_draw_pixel(framePointer, row + top[i].row,
                    WORD_WIDTH - 1 + top[i].col, BLACK);
        }
    }
}
```

```
// dete3cts if teh aliens win
void aliens_detect_game_over(){
#define WINLINE 210 - ALIEN_HEIGHT
    if (bottomAlien[ALIEN_COLUMNS].row >= WINLINE ){  // if the aliens are to low end the
game
        int i;
        for(i=ALIEN_COLUMNS;i<ALIEN_COLUMNS+ALIEN_COLUMNS;i++){
            if(bottomAlien[i].alive){
                interface_game_over(); // end game put up game over screen
                return;
            }
        }
        if (bottomAlien[0].row >= WINLINE ){  // if the aliens are to low end the game
            for(i=0;i<ALIEN_COLUMNS;i++){
                if(bottomAlien[i].alive){
                    interface_game_over(); // end game put up game over screen
                    return;
                }
            }
            if (middleAlien[ALIEN_COLUMNS].row >= WINLINE ){  // if the aliens are to low
end the game
                int i;
                for(i=ALIEN_COLUMNS;i<ALIEN_COLUMNS+ALIEN_COLUMNS;i++){
                    if(middleAlien[i].alive){
                        interface_game_over(); // end game put up game over screen
                        return;
                    }
                }
                if (middleAlien[0].row >= WINLINE ){  // if the aliens are to low end the
game
                    for(i=0;i<ALIEN_COLUMNS;i++){
                        if(middleAlien[i].alive){
                            interface_game_over(); // end game put up game over screen
                            return;
                        }
                    }
                    if (top[0].row >= WINLINE ){  // if the aliens are to low end the game
                        for(i=0;i<ALIEN_COLUMNS;i++){
                            if(top[i].alive){
                                interface_game_over(); // end game put up game over screen
                                return;
                            }
                        }
                    }
                }
            }
        }
    }
}

// moves the aliens and detects wall boundries and direction changes too!
void aliens_move(uint32_t * framePointer) {
    static int32_t flag;
    int32_t i, j;
    for (i = 0; i < ALIEN_COLUMNS; i++) { // Go through every alien column
        // And see if any alien in that column is alive and has hit left
        if (top[i].alive || middleAlien[i].alive || middleAlien[i
                                                                + ALIEN_COLUMNS].alive ||
```

```
bottomAlien[i].alive || bottomAlien[i

+ ALIEN_COLUMNS].alive) {
            if (top[i].col == LEFT_BOUNDRY) { // If an alien has hit side
                flag = 1; // Set the flag that we've hit the side
                hit_left_rail(framePointer); // Call hit_rail.
            }
        }
    }
    for (j = ALIEN_COLUMNS - 1; j >= 0; j--) { // Now to check to see
        if (top[j].alive || middleAlien[j].alive || middleAlien[j
                                                          + ALIEN_COLUMNS].alive ||
bottomAlien[j].alive || bottomAlien[j

+ ALIEN_COLUMNS].alive) {
            if (top[j].col == RIGHT_BOUNDRY) {// if an alien has hit right.
                flag = 0; // false
                hit_right_rail(framePointer); // we have hit the right rail
            }
        }
    }
    if (flag == 1) { // if we are moving right
        aliens_right(framePointer); // go right
    } else { // we are actually going left
        aliens_left(framePointer); // so go left
    }
}

// Kills a random alien
// Currently has a bug that if the last alien dies, infinite loop
void aliens_kill(uint32_t * framePointer) {
    int32_t r = rand() % 55; // Get a random number

    if (r < TOP_TOTAL) { // If we have killed a top
        if (!top[r].alive) { // Already dead!
            aliens_kill(framePointer); // Try again
        } else {
            top[r].alive = false; // kill the alien
            build_tops(framePointer, alien_top_in_12x8, false); // redraw aliens
        }
    } else if (r < (TOP_TOTAL + MIDDLE_TOTAL)) { // if we have killed a mid
        if (!middleAlien[r - TOP_TOTAL].alive) { // Already dead!
            aliens_kill(framePointer); // try again
        } else {
            middleAlien[r - TOP_TOTAL].alive = false; // kill alien
            build_middle(framePointer, alien_middle_in_12x8, false);// redraw aliens
        }
    } else { // we have killed a bot
        if (!bottomAlien[r - (TOP_TOTAL + MIDDLE_TOTAL)].alive) { // Already dead!
            aliens_kill(framePointer); // Try again
        } else {
            bottomAlien[r - (TOP_TOTAL + MIDDLE_TOTAL)].alive = false;// Kill alien
            build_bottom(framePointer, alien_bottom_in_12x8, false);// redraw aliens
        }
    }
}

// Returns true if aliens can shoot- that is, if there exists a top alive alien
```

```
bool can_aliens_shoot() {
    int32_t i; // Declare loop variable
    for (i = 0; i < TOP_TOTAL; i++) { // Look at all the top aliense
        if (top[i].alive) { // If there exists a single alive top alien
            return true; // We have an alive alien!
        }
    }
    return false; // All the top aliens are dead; we cannot shoot
}

// Fires a bullet from a random alien
void alien_missle(uint32_t * framePointer) {
#define TRY_TO_SHOOT_TIMES 3
    if (!can_aliens_shoot()) { // The aliens can't even shoot! Don't even try.
        return;
    }

    int32_t r = rand() % ALIEN_COLUMNS; // Get a random column
    int32_t bullet_address = BAD_ADDRESS; // Initialize the address

    int32_t trying = 0;     // Try several times to shoot
    do { // Keep trying to shoot
        bullet_address = fire_bottom(framePointer, r);
    } while (bullet_address == BAD_ADDRESS && (trying++ <TRY_TO_SHOOT_TIMES)); // until
we get a good address

    if(bullet_address == BAD_ADDRESS){  // We tried 3 times to shoot
        return;                         // But failed! :'(
    }

    // We have a bullet address! now to make it alive and draw it.
    int32_t i;
    for (i = 0; i < ALIEN_NUM_BULLETS; i++) {
        if (alien_bullet[i].alive) { // If we already have a living bullet
            continue; // Go on to the next one
        } else { // We have a dead bullet spot- let's alive a bullet here!
            alien_bullet[i].alive = true;
            // Randomly choose a bullet type
            alien_bullet[i].bullet_type
            = rand() % ALIEN_NUM_BULLET_TYPES ? cross0 : lightning0;
            // TODO: This math can be simplified
            alien_bullet[i].col = bullet_address % SCREEN_RES_X;// Set address
            alien_bullet[i].row = bullet_address / SCREEN_RES_X;// of bullet
            draw_bullet(framePointer, i, WHITE); // And draw it!
            return;
        }
    }
}

// Draws the selected bullet to the screen
void draw_bullet(uint32_t * framePointer, int32_t bullet, uint32_t color) {
#define PIXEL_LINE_1 1      // These
#define PIXEL_LINE_2 2      // defines
#define PIXEL_LINE_3 3      // only
#define PIXEL_LINE_4 4      // have
#define PIXEL_LEFT -1       // meaning
#define PIXEL_RIGHT 1       // in this function, so I put them here
    uint32_t row = alien_bullet[bullet].row; // Current row
```

```
uint32_t col = alien_bullet[bullet].col; // and column where to draw
switch (alien_bullet[bullet].bullet_type) {
case cross0: // Cross0 and cross 3 are identically drawn
case cross3: // The only difference is in the state machine where they go
    // 5 pixels down in a line
    util_draw_pixel(framePointer, row, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_2, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_4, col, color);

    // Crossbar on the cross - right in the middle
    util_draw_pixel(framePointer, row + PIXEL_LINE_2, col + PIXEL_RIGHT,
            color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_2, col + PIXEL_LEFT,
            color);
    break;
case cross1:
    // 5 pixels down in a line
    util_draw_pixel(framePointer, row, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_2, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_4, col, color);

    // Crossbar on the cross- on the lower one
    util_draw_pixel(framePointer, row + PIXEL_LINE_3, col + PIXEL_RIGHT,
            color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_3, col + PIXEL_LEFT,
            color);
    break;
case cross2:
    // 5 pixels down in a line
    util_draw_pixel(framePointer, row, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_2, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_4, col, color);

    // Crossbar on the cross- on the upper one
    util_draw_pixel(framePointer, row + PIXEL_LINE_1, col + PIXEL_RIGHT,
            color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_1, col + PIXEL_LEFT,
            color);
    break;
case lightning0:
    // 5 pixels down - starting left then right, then going back left
    util_draw_pixel(framePointer, row, col + PIXEL_LEFT, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_2, col + PIXEL_RIGHT,
            color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
    util_draw_pixel(framePointer, row + PIXEL_LINE_4, col + PIXEL_LEFT,
            color);
    break;
case lightning1:
    // 5 pixels down - starting right then left, then back right
    util_draw_pixel(framePointer, row, col + PIXEL_RIGHT, color);
```

```
        util_draw_pixel(framePointer, row + PIXEL_LINE_1, col, color);
        util_draw_pixel(framePointer, row + PIXEL_LINE_2, col + PIXEL_LEFT,
                color);
        util_draw_pixel(framePointer, row + PIXEL_LINE_3, col, color);
        util_draw_pixel(framePointer, row + PIXEL_LINE_4, col + PIXEL_RIGHT,
                color);
        break;
    }

}

// This sees if our bottom alien at index r is alive to shoot
int32_t fire_bottom(uint32_t * framePointer, int32_t r) {
    if (!bottomAlien[r + ALIEN_COLUMNS].alive) { // If the very bottom alien is dead
        if (!bottomAlien[r].alive) {// AND the second row alien is also dead
            return fire_middle(framePointer, r); // Try to make a higher alien shoot it
        } else { // the bottom alien is dead, but the second-row one is alive
            // This is the starting coordinate of the bullet.
            return (bottomAlien[r].row + BULLET_COL_OFFSET + 1) * SCREEN_RES_X
                    + (BULLET_COL_OFFSET + bottomAlien[r].col);
        }
    } else { // The very bottom alien is alive and needs to shoot
        // Time to return the starting position of the bullet!
        return (bottomAlien[r + ALIEN_COLUMNS].row + BULLET_COL_OFFSET + 1)
                * SCREEN_RES_X + (BULLET_COL_OFFSET + bottomAlien[r
                                                                 + ALIEN_COLUMNS].col);
    }
}

// This sees if either middle alien at index r is alive to shoot
int32_t fire_middle(uint32_t * framePointer, int32_t r) {
    if (!middleAlien[r + ALIEN_COLUMNS].alive) { // If the very bottom (middle) alien is
dead
        if (!middleAlien[r].alive) {// AND the second row (middle) alien is dead
            return fire_top(framePointer, r); // Top row alien has to fire
        } else { // the bottom alien is dead, but the second-row one is alive
            // This is the starting coordinate of the bullet
            return (middleAlien[r].row + BULLET_COL_OFFSET) * SCREEN_RES_X
                    + (BULLET_COL_OFFSET + middleAlien[r].col);
        }
    } else { // The bottom alien is alive and needs to fire
        // This is the starting coordinate of the bullet
        return (middleAlien[r + ALIEN_COLUMNS].row + BULLET_COL_OFFSET)
                * SCREEN_RES_X + (BULLET_COL_OFFSET + middleAlien[r
                                                                 + ALIEN_COLUMNS].col);
    }
}

// This sees to see if our top alien at index r is alive to shoot
int32_t fire_top(uint32_t * framePointer, int32_t r) {
    if (!top[r].alive) { // Our top alien is dead.
        return BAD_ADDRESS; // We failed to fire a missle! return -1
    } else { // Our alien is alive!
        return (top[r].row + BULLET_COL_OFFSET) * SCREEN_RES_X
                + (BULLET_COL_OFFSET + top[r].col); // Return good address
    }
}
```

```
// Updates alien bullets. erases previous one, increments type, and redraws.
void aliens_update_bullets(uint32_t * framePointer) {
    int32_t i; // Declare loop var
    for (i = 0; i < ALIEN_NUM_BULLETS; i++) { // Cycle through all bullets
        if (alien_bullet[i].row > SCREEN_HEIGHT) { // If bullet off screen
            alien_bullet[i].alive = false; // kill it
        } else if (alien_bullet[i].alive) { // If bullet is alive
            draw_bullet(framePointer, i, BLACK); // erase to prep redraw
            if(tank_detect_collision(alien_bullet[i].row+BULLET_HEIGHT,
alien_bullet[i].col)){
                alien_bullet[i].alive = false;
                continue;
            }
            if(bunkers_detect_collision(alien_bullet[i].row +
BULLET_HEIGHT,alien_bullet[i].col, false)){
                alien_bullet[i].alive = false;
                continue;
            }
            if(alien_bullet[i].row == 220){
                alien_bullet[i].alive = false;
                continue;
            }

            switch (alien_bullet[i].bullet_type) { // Increment bullet type
            case cross0: // mid, going down
                alien_bullet[i].bullet_type = cross1; // bar go down
                break;
            case cross1: // down
                alien_bullet[i].bullet_type = cross3; // bar go mid
                break;
            case cross2: // up
                alien_bullet[i].bullet_type = cross0; // bar go down
                break;
            case cross3: // mid, going up
                alien_bullet[i].bullet_type = cross2; // bar go up
                break;
            case lightning0:// left lightning
                alien_bullet[i].bullet_type = lightning1; // go right
                break;
            case lightning1:// right lightning
                alien_bullet[i].bullet_type = lightning0; // go left
                break;
            }
            alien_bullet[i].row++; // Move bullet down
            draw_bullet(framePointer, i, WHITE); // redraw bullet
        }
    }
}

void aliens_delete_bottom(uint32_t location){
    int32_t row, col; // Declare vars
    for (row = 0; row < ALIEN_HEIGHT; row++) { // looping through y pixels
        for (col = 0; col < WORD_WIDTH; col++) { // looping through x pixels
            if (alien_bottom_out_12x8[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + bottomAlien[location].row,  col +
bottomAlien[location].col,
                        BLACK);
```

```
            }
            if (alien_bottom_in_12x8[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + bottomAlien[location].row,  col +
bottomAlien[location].col,
                        BLACK);
            }
            if (deadAlien[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + bottomAlien[location].row,  col +
bottomAlien[location].col,
                        WHITE);
            }
        }
    }
}
void aliens_delete_top(uint32_t location){
    int32_t row, col; // Declare vars
    for (row = 0; row < ALIEN_HEIGHT; row++) { // looping through y pixels
        for (col = 0; col < WORD_WIDTH; col++) { // looping through x pixels
            if (alien_top_out_12x8[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + top[location].row,  col + top[location].col,
                        BLACK);
            }
            if (alien_top_in_12x8[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + top[location].row,  col + top[location].col,
                        BLACK);
            }
            if (deadAlien[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + top[location].row,  col + top[location].col,
                        WHITE);
            }
        }
    }
}
void aliens_delete_middle(uint32_t location){
    int32_t row, col; // Declare vars
    for (row = 0; row < ALIEN_HEIGHT; row++) { // looping through y pixels
        for (col = 0; col < WORD_WIDTH; col++) { // looping through x pixels
            if (alien_middle_out_12x8[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + middleAlien[location].row,  col +
middleAlien[location].col,
                        BLACK);
            }
            if (alien_middle_in_12x8[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + middleAlien[location].row,  col +
middleAlien[location].col,
                        BLACK);
            }
            if (deadAlien[row] & (1 << (WORD_WIDTH - col - 1))) {
                // If our alien is alive and has a pixel here, draw it
                util_draw_pixel(frame, row + middleAlien[location].row,  col +
middleAlien[location].col,
```

```
                        WHITE);
            }
        }
    }
}

// Tank calls this to see if its bullet collides with an alien
bool aliens_detect_collision(uint32_t row,uint32_t col){
    if(row == (top[0].row + ALIEN_HEIGHT)){ //
        int i;
        for(i=0;i<ALIEN_COLUMNS;i++){
            if(top[i].alive && col > top[i].col && col < top[i].col + ALIEN_WIDTH){
                // The bullet has hit the bottom of our alien!
                interface_increment_score(TOP_POINTS);
                top[i].alive = false;   // Kill the alien
                top[i].exploding = true;
                aliens_delete_top(i); // kill alien
                if(--how_many_aliens_left == 0){
                    interface_success();
                }
                return true;            // We hit something!
            }
        }
    }
    if(row == (middleAlien[0].row + ALIEN_HEIGHT)){
        int i;
        for(i=0;i<ALIEN_COLUMNS;i++){
            if(middleAlien[i].alive &&
                    col > middleAlien[i].col&&col<middleAlien[i].col+ALIEN_WIDTH){
                // The bullet has hit the bottom of our alien!
                interface_increment_score(MIDDLE_POINTS);
                middleAlien[i].alive = false;   // Kill the alien
                middleAlien[i].exploding = true;
                aliens_delete_middle(i); // kill alien
                if(--how_many_aliens_left == 0){
                    interface_success();
                }
                return true;            // We hit something!
            }
        }
    }

    if(row == (middleAlien[ALIEN_COLUMNS].row + ALIEN_HEIGHT)){
        int i;
        for(i=ALIEN_COLUMNS;i<ALIEN_COLUMNS+ALIEN_COLUMNS;i++){
            if(middleAlien[i].alive &&
                    col>middleAlien[i].col && col < middleAlien[i].col + ALIEN_WIDTH){
                // The bullet has hit the bottom of our alien!
                interface_increment_score(MIDDLE_POINTS);
                aliens_delete_middle(i); // kill alien
                middleAlien[i].alive = false;   // Kill the alien
                middleAlien[i].exploding = true;
                if(--how_many_aliens_left == 0){
                    interface_success();
                }
                return true;            // We hit something!
            }
        }
```

```
    }
    if(row == (bottomAlien[0].row + ALIEN_HEIGHT)){
        int i;
        for(i=0;i<ALIEN_COLUMNS;i++){
            if(bottomAlien[i].alive &&
                    col > bottomAlien[i].col && col < bottomAlien[i].col + ALIEN_WIDTH){
                // The bullet has hit the bottom of our alien!
                interface_increment_score(BOTTOM_POINTS);
                aliens_delete_bottom(i); // kill alien
                bottomAlien[i].alive = false;   // Kill the alien
                bottomAlien[i].exploding = true;
                if(--how_many_aliens_left == 0){
                    interface_success();
                }
                return true;            // We hit something!
            }
        }
    }
    if(row == (bottomAlien[ALIEN_COLUMNS].row + ALIEN_HEIGHT)){
        int i;
        for(i=ALIEN_COLUMNS;i<ALIEN_COLUMNS+ALIEN_COLUMNS;i++){
            if(bottomAlien[i].alive &&
                    col > bottomAlien[i].col && col < bottomAlien[i].col + ALIEN_WIDTH){
                // The bullet has hit the bottom of our alien!
                interface_increment_score(BOTTOM_POINTS);
                aliens_delete_bottom(i); // kill alien
                bottomAlien[i].alive = false;   // Kill the alien
                bottomAlien[i].exploding = true;
                if(--how_many_aliens_left == 0){
                    interface_success();
                }
                return true;            // We hit something!
            }
        }
    }

    // If we get here, the bullet is not at the row of any alien
    return false;   // No collision detected.
}
```

```c
/*
 * bunkers_new.h
 *  Taylor Cowley and Andrew Okazaki
 */

#ifndef BUNKERS_H_
#define BUNKERS_H_

#include <stdint.h>
#include <stdbool.h>
// inits the bunkers
void bunkers_init(uint32_t * framePointer);

// Draws the bunkers
void bunkers_build(uint32_t * framePointer);

// For debugging
void bunkers_debug_print();

// Have I been hit?
bool bunkers_detect_collision(uint32_t row, uint32_t col, bool forceDestroy);

#endif /* BUNKERS_NEW_H_ */
```

```c
/*
 * bunkers.c
 * Taylor Cowley and Andrew Okazaki
 */
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "util.h"
#include "bunkers.h"


#define NUM_BUNKERS 4           // We have 4 bunkers
#define NUM_SQUARES 10          // Each bunker has 10 sections
#define NUM_SQUARES_IN_LINE 4   // In a line there are 4 sections
#define BUNKER_ROW 60           // Row the bunkers live on
#define LOC_BUNKER_ONE 60       // Wher ethe first bunker is
#define SQUARE_INCREMENT 6      // Each section is this square
#define LEFT_STRUT_ROW 12       // The extra sections live here
#define LEFT_STRUT_COL 0        // and here
#define RIGHT_STRUT_ROW 12      // and here
#define RIGHT_STRUT_COL 18      // and here
#define BUNKER_ROWS     18      // How many rows each bunker has
#define BUNKER_COLS     24      // How many columns each bunker has
#define GREEN 0x0000FF00        // Hex value for green
#define BUNKER_ROW_LOC 175      // Where our bunker livess?
#define BUNKER_DAMAGE_1 1       // how
#define BUNKER_DAMAGE_2 2       // much
#define BUNKER_DAMAGE_3 3       // damage
#define BUNKER_DAMAGE_4 4       // we have
#define WHITE 0xFFFFFFF     // These
#define BLACK 0x0000000     // are colors
#define ZERO_DAMAGE 0           // No damage!
#define BUFFER 1                // One pixel buffer needed sometimes

// -----------------------------------------------
//  hardcoded static const stuff

// Necessary for storing bunker damage data
#define packword6(b5,b4,b3,b2,b1,b0) \
        ((b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0
 ) )

// Necessary for storing the bunker data
#define
packword24(b23,b22,b21,b20,b19,b18,b17,b16,b15,b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2
,b1,b0) \
        ((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 ) )
```

```
// Shape of the entire bunker.
static const int32_t bunker_24x18[BUNKER_ROWS] = {
        packword24(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0),
        packword24(0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
        packword24(0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
        packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1)};

// First time a bunker is hit, the first damage that happens
static const int32_t bunkerDamage0_6x6[SQUARE_INCREMENT] = {
        packword6(0,1,1,0,0,0), packword6(0,0,0,0,0,1), packword6(1,1,0,1,0,0),
        packword6(1,0,0,0,0,0), packword6(0,0,1,1,0,0), packword6(0,0,0,0,1,0)};

// Second time a bunker is hit, this is its damage
static const int32_t bunkerDamage1_6x6[SQUARE_INCREMENT] = {
        packword6(1,1,1,0,1,0), packword6(1,0,1,0,0,1), packword6(1,1,0,1,1,1),
        packword6(1,0,0,0,0,0), packword6(0,1,1,1,0,1), packword6(0,1,1,0,1,0)};

// Third time a bunker is hit, this is its damage
static const int32_t bunkerDamage2_6x6[SQUARE_INCREMENT] = {
        packword6(1,1,1,1,1,1), packword6(1,0,1,1,0,1), packword6(1,1,0,1,1,1),
        packword6(1,1,0,1,1,0), packword6(0,1,1,1,0,1), packword6(1,1,1,1,1,1)};

// Fourth time a bunker is hit, this is its damage
static const int32_t bunkerDamage3_6x6[SQUARE_INCREMENT] = {
        packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1),
        packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1)};


//  End hardcoded static const stuff
// ------------------------------------------------------


// -------------------------------------
// Internal function declaration
void squares_init();
void bunker_degrade(uint32_t i, uint32_t j);
// end internal function declaration
// -------------------------------------

struct bunker{                  // Our bunker
    uint32_t row;               // has a row
    uint32_t col;               // and a column
    struct squares{             // and 10 sections
        uint32_t row;               // Which have their rows
```

```
        uint32_t col;               // and columns
        uint32_t damage;            // and damage
    }squares[NUM_SQUARES];
}bunker[NUM_BUNKERS];

uint32_t * frame;               // Variable to store the screen frame


// For debugging. Prints out a pixel for each section of bunker
void bunkers_debug_print(){
    int i,j;
    for(i=0;i<NUM_BUNKERS;i++){
        //xil_printf("Bunker %d: %d row, %d col\n\r", i, bunker[i].row, bunker[i].col);
        for(j=0;j<NUM_SQUARES;j++){
            //xil_printf("Bunker %d, square %d: %d row %d col\n\r", i, j,
bunker[i].squares[j].row,bunker[i].squares[j].col);
            util_draw_pixel(frame,SQUARE_INCREMENT+bunker[i].squares[j].row,
SQUARE_INCREMENT+bunker[i].squares[j].col, 0x00000FF);
            util_draw_pixel(frame,bunker[i].squares[j].row,bunker[i].squares[j].col,
0xFFF0000);
        }
    }
}

// Initializes the bunkers
void bunkers_init(uint32_t * framePointer){
    int32_t i, loc = LOC_BUNKER_ONE;    //
    for(i = 0; i < NUM_BUNKERS ; i++){
        bunker[i].row = BUNKER_ROW_LOC; // Divided by 2 because screen is half
        bunker[i].col = loc;        // which column it is at
        loc += LOC_BUNKER_ONE;      // Add by the offset
    }
    bunkers_build(framePointer);    // Draw the bunkers on the screen

    squares_init(); // init the bunker squares
}

// Initializes the bunker sections
void squares_init(){
    uint32_t i, j, row_count, col_count;            // Var init
    row_count = 0;
    col_count = 0;
    for(i = 0; i < NUM_BUNKERS; i++){               // Go through all bunkers
        for(j = 0; j < NUM_SQUARES-2; j++){         // And all squares
            if(j == NUM_SQUARES_IN_LINE){
                row_count += SQUARE_INCREMENT;
                col_count = 0;
            }
            //// And give them addresses and damage
            bunker[i].squares[j].row = bunker[i].row + row_count;
            bunker[i].squares[j].col = bunker[i].col + col_count;
            bunker[i].squares[j].damage = ZERO_DAMAGE;
            col_count += SQUARE_INCREMENT;
        }
        // Now to initialize the last two sections
        bunker[i].squares[j].row = bunker[i].row + LEFT_STRUT_ROW;
        bunker[i].squares[j].col = bunker[i].col + LEFT_STRUT_COL;
        bunker[i].squares[j].damage = ZERO_DAMAGE;
```

```
        j++;
        bunker[i].squares[j].row = bunker[i].row + RIGHT_STRUT_ROW;
        bunker[i].squares[j].col = bunker[i].col + RIGHT_STRUT_COL;
        bunker[i].squares[j].damage = ZERO_DAMAGE;
        row_count = 0;
        col_count = 0;
    }
}

// Draws the bunkers
void bunkers_build(uint32_t * framePointer){
    frame = framePointer;
    int32_t row, col, b;                            // Declare loop vars
    for(row=0;row<BUNKER_ROWS;row++){                   // Go through rows
        for(col=0;col<BUNKER_COLS;col++){               // Go through cols
            if ((bunker_24x18[row] & (1<<(BUNKER_COLS-col-1)))) {// if pixel
                for(b = 0; b <NUM_BUNKERS; b++){// draw that pixel every time
                    util_draw_pixel(framePointer,row+bunker[b].row,col+bunker[b].col,GREEN
);
                }
            }
        }
    }
}



// Is our bunker hit by something?
bool bunkers_detect_collision(uint32_t row, uint32_t col, bool forceDestroy){
    uint32_t i, j;
    for(i = 0; i < NUM_BUNKERS; i++){
        for(j=0; j < NUM_SQUARES; j++){
            if(bunker[i].squares[j].damage < 4 && bunker[i].squares[j].row +
                    SQUARE_INCREMENT >= row&&  bunker[i].squares[j].row <= row){
                // If we have been hit
                if((col <=  bunker[i].squares[j].col + SQUARE_INCREMENT+BUFFER)
                        && (col >= bunker[i].squares[j].col-BUFFER)){
                    // and we have been hit
                    if(forceDestroy){       // an alien crashed into us
                        bunker_degrade(i,j);// completely
                        bunker_degrade(i,j);// destroy
                        bunker_degrade(i,j);// totally
                        bunker_degrade(i,j);//
                    }else {                 // Just a bullet
                        bunker_degrade(i,j);// only one destroy
                    }
                    return true;            // We have been hit!
                }
            }
        }
    }
    return false;                           // Noone got hit, sorry
}

void bunker_degrade(uint32_t i, uint32_t j){
    bunker[i].squares[j].damage++;
    int32_t r,c;
    for(r=0;r<SQUARE_INCREMENT;r++){        // Go through rows
```

```
        for(c=0;c<SQUARE_INCREMENT;c++){        // and columns
            if (bunker[i].squares[j].damage == BUNKER_DAMAGE_1 && (bunkerDamage0_6x6[r] &
(1<<(SQUARE_INCREMENT-c-1)))){
                // If we need to erase a pixel here, do so.
                util_draw_pixel(frame,r+bunker[i].squares[j].row,c+bunker[i].squares[j].c
ol, BLACK);
            }else if(bunker[i].squares[j].damage == BUNKER_DAMAGE_2 &&
(bunkerDamage1_6x6[r] & (1<<(SQUARE_INCREMENT-c-1)))){
                // If we need to erase a pixel here, do so.
                util_draw_pixel(frame,r+bunker[i].squares[j].row,c+bunker[i].squares[j].c
ol, BLACK);

            }else if(bunker[i].squares[j].damage == BUNKER_DAMAGE_3 // 2 damage level
                    && (bunkerDamage2_6x6[r] & (1<<(SQUARE_INCREMENT-c-1)))){
                // If we need to erase a pixel here, do so.
                util_draw_pixel(frame,r+bunker[i].squares[j].row,c+bunker[i].squares[j].c
ol, BLACK);

            }else if(bunker[i].squares[j].damage == BUNKER_DAMAGE_4 // 3 damage level
                    && (bunkerDamage3_6x6[r] & (1<<(SQUARE_INCREMENT-c-1)))){
                // If we need to erase a pixel here, do so.
                util_draw_pixel(frame,r+bunker[i].squares[j].row,c+bunker[i].squares[j].c
ol, BLACK);
            }
        }
    }
}
```

```c
/*
 * interface.h
 * Taylor Cowley and Andrew Okazaki
 */

#ifndef INTERFACE_H_
#define INTERFACE_H_

#include <stdbool.h>

// adds a value to the score
void interface_increment_score(uint32_t incrementor);

//Initialize entire board
void interface_init_board(uint32_t * framePointer);

// The tank has been hit
void interface_kill_tank();

// Our game over screen :)
void interface_game_over();

// Our success screen
void interface_success();

// Draws the mother ship points that you scored.
void interface_alien_ship_points(uint32_t mother_ship_points, uint32_t col_loc, bool
erase);

#endif /* INTERFACE_H_ */
```

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

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <stdbool.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "util.h"
#include "interface.h"
#define WORDS_HEIGHT 5      // height of score and lives
#define TANK_HEIGHT 8       // our tank is 8 high
#define GAME_X 320          // How wide our game screen is
#define LINE_Y 225          // Where the line at the bottom goes

#define EXTRA_TANK_0 250    // X coordinate of extra tanks
#define EXTRA_TANK_1 270    // X coordinate of extra tanks
#define EXTRA_TANK_2 290    // X coordinate of extra tanks
#define EXTRA_TANK_Y_OFFSET 5   // How far down the extra tanks are

#define LIVES_WIDTH 24      // How wide our lives display is
#define SCORE_WIDTH 28      // How wide our score is
#define TANK_WIDTH 15       // How wide our tank is
#define NUMBER_WIDTH 4      // How wide each number is
#define GREEN 0x0000FF00    // Hex for green
#define WHITE 0xFFFFFFF     // These
#define BLACK 0x0000000     // are colors
#define RED 0xFFF0000       // Shocking pink is the best one
#define SHOCKING_PINK 0xFF6FFF
#define MOTHER_SHIP_POINT_COLOR SHOCKING_PINK

#define WORDS_ROW_OFFSET 7      // which row to place words lives and row
#define LIVES_COL_OFFSET 220    // which col to place lives
#define SCORE_COL_OFFSET 15     // which col to place score
#define GAME_COL_OFFSET 110     // Game Over position
#define GAME_ROW_OFFSET 120     // Game Over position
#define OVER_COL_OFFSET 150     // Game Over position
#define OVER_ROW_OFFSET 120     // Game Over position
#define SHIP_ROW 22             // row of the ship

#define DIGIT_ONE   55  // scores first digit
#define DIGIT_TWO   50  // scores second digit
#define DIGIT_THREE 45  // scores third digit
#define DIGIT_FOUR  40  // scores fourth digit
#define DIGIT_FIVE  35  // scores fifth digit
#define DIGIT_SIX   30  // scores sixth digit



// 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 ) )

static const uint32_t tank_15x8[TANK_HEIGHT] = {
packword15(0,0,0,0,0,0,0,1,0,0,0,0,0,0,0),
packword15(0,0,0,0,0,0,0,1,1,1,0,0,0,0,0),
packword15(0,0,0,0,0,0,0,1,1,1,0,0,0,0,0),
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)
};

#define packword4(b3,b2,b1,b0) ((b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ))
static const uint32_t ZERO_4x5[] ={ // sprite 0
    packword4(1,1,1,1), packword4(1,0,0,1), packword4(1,0,0,1),
    packword4(1,0,0,1), packword4(1,1,1,1)};
static const uint32_t ONE_4x5[] = { // sprite 1
    packword4(0,1,1,0), packword4(0,0,1,0), packword4(0,0,1,0),
    packword4(0,0,1,0), packword4(0,1,1,1)};
static const uint32_t TWO_4x5[] = { // sprite 2
    packword4(1,1,1,1), packword4(0,0,0,1), packword4(1,1,1,1),
    packword4(1,0,0,0), packword4(1,1,1,1)};
static const uint32_t THREE_4x5[] = { // sprite 3
    packword4(1,1,1,1), packword4(0,0,0,1), packword4(1,1,1,1),
    packword4(0,0,0,1), packword4(1,1,1,1)};
static const uint32_t FOUR_4x5[] = { // sprite 4
    packword4(1,0,0,1), packword4(1,0,0,1), packword4(1,1,1,1),
    packword4(0,0,0,1), packword4(0,0,0,1)};
static const uint32_t FIVE_4x5[] = { // sprite 5
    packword4(1,1,1,1), packword4(1,0,0,0), packword4(1,1,1,1),
    packword4(0,0,0,1), packword4(1,1,1,1)};
static const uint32_t SIX_4x5[] = { // sprite 6
    packword4(1,1,1,1), packword4(1,0,0,0), packword4(1,1,1,1),
    packword4(1,0,0,1), packword4(1,1,1,1)};
static const uint32_t SEVEN_4x5[] = { // sprite 7
    packword4(1,1,1,1), packword4(0,0,0,1), packword4(0,0,0,1),
    packword4(0,0,0,1), packword4(0,0,0,1)};
static const uint32_t EIGHT_4x5[] = { // sprite 8
    packword4(1,1,1,1), packword4(1,0,0,1), packword4(1,1,1,1),
    packword4(1,0,0,1), packword4(1,1,1,1)};
static const uint32_t NINE_4x5[] = { // sprite 9
    packword4(1,1,1,1), packword4(1,0,0,1), packword4(1,1,1,1),
    packword4(0,0,0,1), packword4(0,0,0,1)};


#define
packword28(b27,b26,b25,b24,b23,b22,b21,b20,b19,b18,b17,b16,b15,b14,b13,b12,b11,b10,b9,b8,b
7,b6,b5,b4,b3,b2,b1,b0) \
((b27 << 27) | (b26 << 26) | (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 uint32_t SCORE_28x5[SCORE_WIDTH] = { // sprite "SCORE"
    packword28(0,1,1,1,1,0,1,1,1,1,0,1,1,1,1,0,1,1,1,1,0,0,1,1,1,1,0,0),
    packword28(1,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0),
    packword28(0,1,1,1,0,0,1,0,0,0,0,1,0,0,1,0,1,1,1,1,1,0,0,1,1,1,0,0,0),
    packword28(0,0,0,0,1,0,1,0,0,0,0,1,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,0),
    packword28(1,1,1,1,0,0,1,1,1,1,0,1,1,1,1,0,1,0,0,0,1,0,1,1,1,1,0,0)};
static const uint32_t GAME_28x5[SCORE_WIDTH] = { // sprite "GAME"
    packword28(0,1,1,1,1,0,0,0,1,0,0,0,1,0,0,0,1,0,1,1,1,1,0,0,0,0,0,0),
    packword28(1,0,0,0,0,0,0,0,1,0,1,0,0,1,1,0,1,1,0,1,0,0,0,0,0,0,0,0),
    packword28(1,0,1,1,1,0,1,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,0,0,0,0,0,0),
    packword28(1,0,0,0,1,0,1,1,1,1,1,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0),
    packword28(1,1,1,1,0,0,1,0,0,0,1,0,1,0,0,0,1,0,1,1,1,1,0,0,0,0,0,0)};
static const uint32_t OVER_28x5[SCORE_WIDTH] = { // sprite "OVER"
    packword28(0,1,1,0,0,1,0,0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,0,1,0,1,0,0,0,1,0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0),
    packword28(1,0,0,1,0,1,0,0,0,1,0,1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,0,1,0,0,1,0,1,0,0,1,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0),
    packword28(0,1,1,0,0,0,0,1,0,0,0,1,1,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0)};
static const uint32_t WIN_28x5[SCORE_WIDTH] = { // sprite "WIN"
    packword28(0,0,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,0,0,0,1,0,1,0,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,1,0,0,1,0,1,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packword28(1,0,1,0,1,0,0,1,0,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packword28(0,1,1,1,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0)};

#define
packword24(b23,b22,b21,b20,b19,b18,b17,b16,b15,b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2
,b1,b0) \
((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 uint32_t LIVES_24x5[LIVES_WIDTH] = { // sprite "LIVES"
    packword24(1,0,0,0,0,1,0,1,0,0,0,1,0,1,1,1,1,0,0,1,1,1,1,1),
    packword24(1,0,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,0,1,0,0,0,0,0),
    packword24(1,0,0,0,0,1,0,1,0,0,0,1,0,1,1,1,1,0,0,0,1,1,1,1,0),
    packword24(1,0,0,0,0,1,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0,1),
    packword24(1,1,1,1,0,1,0,0,0,1,0,0,0,1,1,1,1,0,1,1,1,1,1,0)};


//------------------------------------------------------------------------------
// Internal functions not defined in our .h
void interface_draw_line();     // Draws the line at the bottom of the screen
void interface_draw_tanks();    // Draws the "extra life" tanks
void interface_draw_lives();    //Draws the lives to the screen;
void interface_draw_score();    // Draws the Score to the screen
void interface_init_numbers();  //Draw the score in numbers
void interface_update_digit(const uint32_t number[], uint32_t digit);   // writes digit
void interface_digit(uint32_t value, uint32_t digit);           // Also.
void interface_draw_game_over();// Draws game over to the screen
void interface_update_ship_digit(const uint32_t number[], uint32_t digit, bool erase);
// End defining internal functions
//------------------------------------------------------------------------------

uint32_t * frame;               // How to write to the screen
```

```
int32_t lives = 3;              // How many lives do we have?
uint32_t score = 0;             // keep track of game score

//initialize the score board to all zeros
void interface_init_numbers(){                                      //set the frame
    int row, col;                                                   //declare vars
    for(row=0;row<WORDS_HEIGHT;row++){                              //through width
         for(col=0;col<NUMBER_WIDTH;col++){                         //and height
             if((ZERO_4x5[row] & (1<<(NUMBER_WIDTH-col-1)))){       //and draw score
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_ONE, GREEN);   //draw first digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_TWO, GREEN);   //draw second digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_THREE, GREEN); //draw third digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_FOUR, GREEN);  //draw fourth digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_FIVE, GREEN);  //draw fifth digit
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + DIGIT_SIX, GREEN);   //draw sixth digit
             }
         }
     }
}

//increment the score by value
void interface_increment_score(uint32_t value){
    uint32_t i, temp_score; // initialize variables
    uint32_t mod = 10;              // set the modulus value
    uint32_t divide = 1;            // set the value to divide by
    uint32_t digit_loc = 55;        // set the column location of first digit
    score += value;                 // increment the game score by value
    temp_score = score;             // set a temporary score to edit

    for(i = 0; i < 6; i++){                 // loop through all six digits
        uint32_t number = temp_score % mod; // modulus the score
        number = number / divide;           // convert to a single digit value
        temp_score = temp_score - number;   // update the temporary score
        interface_digit(number,digit_loc);  // print to screen

        digit_loc -= 5; // update to the next digit column location
        divide *= 10;   // increment the number we divide by
        mod *= 10;      // increment the modulus number
    }
}

// convert a integer to a sprite to enable us to draw to screen
// value is the integer to print to screen
// digit is the column location of the digit to print to
void interface_digit(uint32_t value, uint32_t digit){
    switch(value){                                  // value the integer
        case 0:                                     // if value = 0
            interface_update_digit(ZERO_4x5,digit); // print 0 to location
            break;
        case 1:                                     // value = 1
            interface_update_digit(ONE_4x5,digit);  // print 1 to location
            break;
```

```
        case 2:                                     // value = 2
            interface_update_digit(TWO_4x5,digit);  // print 2 to location
            break;
        case 3:                                     // value = 3
            interface_update_digit(THREE_4x5,digit);// print 3 to location
            break;
        case 4:                                     // value = 4
            interface_update_digit(FOUR_4x5,digit); // print 4 to location
            break;
        case 5:                                     // value = 5
            interface_update_digit(FIVE_4x5,digit); // print 5 to location
            break;
        case 6:                                     // value = 6
            interface_update_digit(SIX_4x5,digit);  // print 6 to location
            break;
        case 7:                                     // value = 7
            interface_update_digit(SEVEN_4x5,digit);// print 7 to location
            break;
        case 8:                                     // value = 8
            interface_update_digit(EIGHT_4x5,digit);// print 8 to location
            break;
        case 9:                                     // value = 9
            interface_update_digit(NINE_4x5,digit); // print 9 to location
            break;
    }
}

//Draw the digit to the score
//number[] is the sprite of 1,2,3 ect.
//digit is the column offset of the screen to print to
void interface_update_digit(const uint32_t number[], uint32_t digit){
    int row, col;                                           //init row and col
    for(row=0;row<WORDS_HEIGHT;row++){                      // Go through width
         for(col=0;col<NUMBER_WIDTH;col++){                 // and height
             if((number[row] & (1<<(NUMBER_WIDTH-col-1)))){ // if  sprite
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + digit, GREEN);   // print to pixel green
             }else{                         // if value = 0
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         + digit, BLACK);   // print to pixel black
             }
         }
     }
}

//initialize the entire screen
void interface_init_board(uint32_t * framePointer){
    frame = framePointer;           // Set the pointer to the screen
    interface_draw_score();         // Draw a score (0)
    interface_draw_lives();         // Draw "lives"
    interface_draw_line();          // Draw the line at the bottom
    interface_draw_tanks();         // Draw our extra lives
    interface_init_numbers();       // Make numbers good
}


//This draws the word score to the screen.
void interface_draw_score(){
```

```
    int row, col;
    for(row=0;row<WORDS_HEIGHT;row++){                    // Go through width
         for(col=0;col<SCORE_WIDTH;col++){                // and height
             if((SCORE_28x5[row] & (1<<(SCORE_WIDTH-col-1)))){// and draw score
                 util_draw_pixel(frame,row+WORDS_ROW_OFFSET,col+SCORE_COL_OFFSET
                         , WHITE);      // draw white
             }
         }
     }
}

//This draws the word lives to the screen.
void interface_draw_lives(){
    int row, col;
    for(row=0;row<WORDS_HEIGHT;row++){                    // Go through width
         for(col=0;col<LIVES_WIDTH;col++){                // and height
             if((LIVES_24x5[row] & (1<<(LIVES_WIDTH-col-1)))){// and draw Lives
                 util_draw_pixel(frame, row + WORDS_ROW_OFFSET, col +
                         LIVES_COL_OFFSET, WHITE);      // draw white
             }
         }
     }
}
// This draws the green line at the bottom of the screen
void interface_draw_line(){
    int row, col;                                 // Initialize
    row = LINE_Y;                                 // variables
    for(col=0;col<GAME_X;col++){                  // Go along the screen and draw
        util_draw_pixel(frame, row, col, GREEN);//draw green
    }
}

// This draws the extra tanks to the screen
void interface_draw_tanks(){
     int row, col;                                        // Init loop vars
     for(row=0;row<TANK_HEIGHT;row++){                    // Go through width
         for(col=0;col<TANK_WIDTH;col++){                 // and height
             if((tank_15x8[row] & (1<<(TANK_WIDTH-col-1)))) {// and draw 3 tanks
                 util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                         col+EXTRA_TANK_0, GREEN);
                 util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                         col+EXTRA_TANK_1, GREEN);
                 util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                         col+EXTRA_TANK_2, GREEN);
             }
         }
     }
}
// This draws the game over screen
void interface_draw_game_over(){
    int row, col;
    for(row=0;row<WORDS_HEIGHT;row++){                    // Go through width
         for(col=0;col<SCORE_WIDTH;col++){                // and height
             if((GAME_28x5[row] & (1<<(SCORE_WIDTH-col-1)))){// and draw score
                 util_draw_pixel(frame, row + GAME_ROW_OFFSET,
                         col + GAME_COL_OFFSET, RED);       // draw white
             }
         }
```

```
     }
    for(row=0;row<WORDS_HEIGHT;row++){                      // Go through width
         for(col=0;col<SCORE_WIDTH;col++){                  // and height
             if((OVER_28x5[row] & (1<<(SCORE_WIDTH-col-1)))){// and draw score
                 util_draw_pixel(frame, row + OVER_ROW_OFFSET,
                         col + OVER_COL_OFFSET, RED);       // draw white
             }
         }
     }
}

// This kills a tank
void interface_kill_tank(){
    lives--;                            // Take a live
    if(lives < 0){                      // maybe game over
        interface_draw_game_over();     // Game over
        interface_game_over();
    }


    int row, col;
    switch(lives){                                          // lives left
    case 2:                                                 // lives = 2
        for(row=0;row<TANK_HEIGHT;row++){                   // Go through width
             for(col=0;col<TANK_WIDTH;col++){               // and height
                 if((tank_15x8[row] & (1<<(TANK_WIDTH-col-1)))) {// draw 3 tanks
                     util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                             col+EXTRA_TANK_2, BLACK);
                 }
             }
         }
        break;
    case 1:                                                 // lives = 1
        for(row=0;row<TANK_HEIGHT;row++){                   // Go through width
             for(col=0;col<TANK_WIDTH;col++){               // and height
                 if((tank_15x8[row] & (1<<(TANK_WIDTH-col-1)))) {// draw 3 tanks
                     util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                             col+EXTRA_TANK_1, BLACK);
                 }
             }
         }
        break;
    case 0:                                                 //zero lives left
        for(row=0;row<TANK_HEIGHT;row++){                   // Go through width
             for(col=0;col<TANK_WIDTH;col++){               // and height
                 if((tank_15x8[row] & (1<<(TANK_WIDTH-col-1)))) {// draw 3 tanks
                     util_draw_pixel(frame, row+EXTRA_TANK_Y_OFFSET,
                             col+EXTRA_TANK_0, BLACK);
                 }
             }
         }
        break;
    }
}

// We have game over!
void interface_game_over(){
    interface_draw_game_over();     // draw "game over"
```

```
    //xil_printf("game over\n\r");  // print it.
    exit(1);                        // and kill program
}

// Draw the win screen
void interface_success(){
    int row, col;
    for(row=0;row<WORDS_HEIGHT;row++){                  // Go through width
         for(col=0;col<SCORE_WIDTH;col++){              // and height
             if((WIN_28x5[row] & (1<<(SCORE_WIDTH-col-1)))){// and draw score
                 util_draw_pixel(frame, row + GAME_ROW_OFFSET,
                         col + GAME_COL_OFFSET, RED);       // draw white
             }
         }
     }
    //xil_printf("you win!\n\r");
    exit(1);        // Kill the program
}

// convert a integer to a sprite to enable us to draw to screen
// value is the integer to print to screen
// digit is the column location of the digit to print to
void interface_ship_digit(const uint32_t value, uint32_t digit, bool erase){
    switch(value){                                      // value the integer
        case 0:                                         // if value = 0
            interface_update_ship_digit(ZERO_4x5,digit, erase); // print 0 to location
            break;
        case 1:                                         // value = 1
            interface_update_ship_digit(ONE_4x5,digit,  erase); // print 1 to location
            break;
        case 2:                                         // value = 2
            interface_update_ship_digit(TWO_4x5,digit,  erase); // print 2 to location
            break;
        case 3:                                         // value = 3
            interface_update_ship_digit(THREE_4x5,digit,  erase);// print 3 to location
            break;
        case 4:                                         // value = 4
            interface_update_ship_digit(FOUR_4x5,digit,  erase);    // print 4 to location
            break;
        case 5:                                         // value = 5
            interface_update_ship_digit(FIVE_4x5,digit,  erase);    // print 5 to location
            break;
        case 6:                                         // value = 6
            interface_update_ship_digit(SIX_4x5,digit,  erase); // print 6 to location
            break;
        case 7:                                         // value = 7
            interface_update_ship_digit(SEVEN_4x5,digit,  erase);// print 7 to location
            break;
        case 8:                                         // value = 8
            interface_update_ship_digit(EIGHT_4x5,digit,  erase);// print 8 to location
            break;
        case 9:                                         // value = 9
            interface_update_ship_digit(NINE_4x5,digit,  erase);    // print 9 to location
            break;
    }
}

//Draw the digit to the score
```

```
//number[] is the sprite of 1,2,3 ect.
//digit is the column offset of the screen to print to
void interface_update_ship_digit(const uint32_t number[], uint32_t digit, bool erase){
    uint32_t color = erase ? BLACK : MOTHER_SHIP_POINT_COLOR;
    int row, col;   //initialize row and column
    for(row=0;row<WORDS_HEIGHT;row+
+){                                                              // Go through width
        for(col=0;col<NUMBER_WIDTH;col+
+){                                                          // and height
            if((number[row] &
(1<<(NUMBER_WIDTH-col-1)))){                                                // if value
in sprite = 1
                util_draw_pixel(frame, row + SHIP_ROW, col + SCORE_COL_OFFSET + digit,
color); // print to pixel green

 }else{
/ if value = 0
                util_draw_pixel(frame, row + SHIP_ROW, col + SCORE_COL_OFFSET + digit,
BLACK); // print to pixel black
            }
        }
     }
}

// print the alien points of ship
void interface_alien_ship_points(uint32_t mother_ship_points, uint32_t col_loc, bool
erase){
//  xil_printf("printing points %d\n\r", mother_ship_points);

    uint32_t i, temp_score; // initialize variables
    uint32_t mod = 10;              // set the modulus value
    uint32_t divide = 1;            // set the value to divide by
    temp_score = mother_ship_points;            // set a temporary score to edit
    for(i = 0; i < 3; i++){                 // loop through all six digits
        uint32_t number = temp_score % mod; // modulus the score
        number = number / divide;           // divide the number to convert to a single
digit value
        temp_score = temp_score - number;   // update the temporary score
        interface_ship_digit(number,col_loc,erase);     // print to screen

        col_loc -= 5;   // update to the next digit column location
        divide *= 10;   // increment the number we divide by
        mod *= 10;      // increment the modulus number
    }
}
```

```c
/*
 * mother_ship.h
 *
 *  Taylor Cowley and Andrew Okazaki
 */

#ifndef MOTHER_SHIP_H_
#define MOTHER_SHIP_H_

#include <stdbool.h>
#include <stdint.h>

// Initializes the mother ship
void mother_ship_init();

// Spawns a mother ship
void mother_ship_spawn();

// Moves the mother ship right
void mother_ship_move();

// Detects a bullet collision on the mother ship
bool mother_ship_detect_collision(uint32_t row, uint32_t col);

// Draws the mother ship
void mother_ship_draw(uint32_t color);

// Shows the points for killing the mother ship
void mother_ship_points_blink();

#endif /* MOTHER_SHIP_H_ */
```

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

#include "mother_ship.h"
#include "interface.h" // enables update score
#include "util.h"

// Hard-coded definition for what the mother ship looks like
#define packword16(b15,b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0)  \
        ((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 uint32_t MOTHER_SHIP_16x7[] ={
        packword16(0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0),
        packword16(0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0),
        packword16(0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
        packword16(0,1,1,0,1,1,0,1,1,0,1,1,0,1,1,0),
        packword16(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
        packword16(0,0,1,1,1,0,0,1,1,0,0,1,1,1,0,0),
        packword16(0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0)};

#define MOTHER_SHIP_ROW 22      // Where the mother ship spawns at
#define MOTHER_SHIP_HEIGHT 7    // Mother ship is this tall
#define MOTHER_SHIP_WIDTH 16    // Mother ship is this wide
#define MOTHER_SHIP_MOVE_SPEED 2// Mother ship moves this many pixels each
#define SCREEN_WIDTH 320        // Screen is 320 pixels wide
#define SHOCKING_PINK 0xFF6FFF  // We want a cool color mother ship
#define MOTHER_SHIP_COLOR SHOCKING_PINK
#define BLACK 0x0               // Black color for erasing
#define BLINKING_TIMES 10       // How long we want the score to stay

struct{                         // Defines our mother ship
    uint32_t row;               // Lives at a certain row
    uint32_t col;               // Lives at a certain column
    bool alive;                 // Is she alive?
}mother_ship;

uint32_t mother_ship_points=123;// Initial value of points for killing her
uint32_t * frame;               // The variable to write pixels to the screen
bool blinking;                  // Whether the score of dead mother ship is.

// Initializes the mother ship
void mother_ship_init(uint32_t * framePointer){
    blinking = false;                           // Death score isn't there
    frame = framePointer;                       // Store the screen frame
    mother_ship.row = MOTHER_SHIP_ROW;          // She lives at this row
    mother_ship.alive = false;                  // She is not yet alive
    mother_ship.col = 0;                        // She spawns at left of screen
}

// Shows the points after a successful mother ship kill
void mother_ship_points_blink(){
    if(!blinking)
        return;                                 // If not blinking, don't go
```

```
    static uint32_t times_blink = 0;            // We blink for a time
    times_blink++;                              // Which counts up

    if(times_blink > BLINKING_TIMES){           // If we have displayed enough
        times_blink = 0;                        // Reset timer and erase it.
        interface_alien_ship_points(mother_ship_points, mother_ship.col, true);
        blinking = false;                       // And we aren't running no more
    }

}

// Spawns a mother ship
void mother_ship_spawn(){
    if(mother_ship.alive)                       // Can't spawn when alive!
        return;
    // Erases any previously-drawn points
    interface_alien_ship_points(mother_ship_points, mother_ship.col, true);
    mother_ship.col = 0;                        // Spawns at left
    mother_ship.alive = true;                   // She is now alive
    mother_ship_draw(MOTHER_SHIP_COLOR);        // Draw her.
    blinking = false;                           // No score blinking anymore
}

// Moves the mother ship right
void mother_ship_move(){
    if(!mother_ship.alive)
        return;                                 // Can't move when dead!
    mother_ship_draw(BLACK);                    // Erase old version
    mother_ship.col += MOTHER_SHIP_MOVE_SPEED;  // Move her
    if(mother_ship.col > SCREEN_WIDTH-MOTHER_SHIP_WIDTH){   // She left.
        mother_ship.alive = false;              // So is now dead
        mother_ship.col = SCREEN_WIDTH;         // And off the screen
        return;                                 // Exit
    }
    mother_ship_draw(MOTHER_SHIP_COLOR);        // Draw her!
}

// Detects a bullet collision on the mother ship
bool mother_ship_detect_collision(uint32_t row, uint32_t col){
    // If it is at the right row and in-between her columns
    if(row == mother_ship.row+MOTHER_SHIP_HEIGHT
            && col>mother_ship.col && col < mother_ship.col+MOTHER_SHIP_WIDTH){
        mother_ship_points = rand()%500 + 316;          // Make random point
        interface_increment_score(mother_ship_points);  // Player gets points
        mother_ship.alive = false;                      // She dies
        mother_ship_draw(BLACK);                        // and gets erased
        // Her points get drawn
        interface_alien_ship_points(mother_ship_points, mother_ship.col, false);
        blinking = true;                                // drawing her points
        return true;                                    // We hit something!
    }
    return false;                                       // nope, not hit.
}

// Draws the mother ship
void mother_ship_draw(uint32_t color){
    int r, c;
    for(r=0;r<MOTHER_SHIP_HEIGHT;r++){              // Go through width
```

```
            for(c=0;c<MOTHER_SHIP_WIDTH;c++){           // and height
                if((MOTHER_SHIP_16x7[r] & (1<<(MOTHER_SHIP_WIDTH-c-1)))){//draw ship
                  util_draw_pixel(frame,r+mother_ship.row,c+mother_ship.col,color);
                }
            }
        }
    }
```

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

#ifndef TANK_H_
#define TANK_H_
#include <stdint.h>
#include <stdbool.h>

void tank_init();
// moves our tank left by a certain number of pixels
void tank_move_left(uint32_t * framePointer);
// moves our tank right by a certain number of pixels
void tank_move_right(uint32_t * framePointer);

// This simply draws the tank on the screen, where it is at now.
void tank_draw(uint32_t * framePointer, bool erase);

// Alives a shell and draws it to the screen
void tank_fire(uint32_t * framePointer);

// Moves the shell up on the screen
void tank_update_bullet(uint32_t * framePointer);

// Our tank dies.
void tank_die();

// Our tank tells whether something hit it, and dies if it is hit.
bool tank_detect_collision(uint32_t row, uint32_t col);

#endif /* TANK_H_ */
```

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

#include <stdint.h>
#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.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_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 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(0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packWord26(0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,0),
    packWord26(0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,0),
    packWord26(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(0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,1,1,0,0,1,1,0,0),
    packWord26(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(0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,0),
    packWord26(0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,0),
    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),
    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),
    packWord26(0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
    packWord26(0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
    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),};
static const int tankDeath2[TANK_DEATH_HEIGHT] = {
    packWord26(1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packWord26(0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
    packWord26(0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1),
    packWord26(0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0),
    packWord26(0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0),
    packWord26(1,1,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0),
    packWord26(1,1,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0),
    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 = 210;     // Tank starts at this row
    tank.col = 160;     // 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
        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<20){          // 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();
        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 < 400 ; 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.
}
```

```c
/*
 * utilities.h
 *  Taylor Cowley and Andrew Okazaki
 *  This is a collection of functions used by several things.
 */

#ifndef UTIL_H_
#define UTIL_H_
#include <stdint.h>

#define UTIL_SCREEN_WIDTH   320 // Our game screen is 320
#define UTIL_SCREEN_HEIGHT  240 // by 240

// Draws a pixel on the screen.
void util_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t color);

#endif /* UTILITIES_H_ */
```

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

#include "util.h"
#define ROW_MULTIPLIER 1280 // 640 * 2 for screen doubling
#define ROW 640             // one row offset
#define COL_MULTIPLIER 2    // Offset of the row

/*
 * Draws a pixel on the screen. To compensate for our double-resolution screen,
 * it must draw 4 real pixels for every in-came pixel.
 */
void util_draw_pixel(uint32_t *frame, uint32_t row, uint32_t c,uint32_t color){
    // We draw 4 pixels for every 1 small-screen pixel
    frame[row * ROW_MULTIPLIER + c * COL_MULTIPLIER]            = color;
    frame[row * ROW_MULTIPLIER + c * COL_MULTIPLIER + 1]        = color;
    frame[row * ROW_MULTIPLIER + ROW + c * COL_MULTIPLIER]      = color;
    frame[row * ROW_MULTIPLIER + ROW + c * COL_MULTIPLIER + 1]  = color;
}
```