Adam Hastings
Yazan Halawa

Lab3 Code Submission

```
/*
 * Copyright (c) 2009 Xilinx, Inc.  All rights reserved.
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
 * COURTESY TO YOU.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
 * ONE POSSIBLE   IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
 * STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
 * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION.
 * XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO
 * THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO
 * ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE
 * FROM CLAIMS OF INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.
 *
 */

/*
 *vdmaTest.c: simple test application
 */

#include <stdio.h>
#include <stdlib.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "render.h"
#include "globals.h"
#define DEBUG
void print(char *str);

#define FRAME_BUFFER_0_ADDR 0xC1000000  // Starting location in DDR where we
will store the images that we display.
#define MAX_SILLY_TIMER 10000000;

int main()
{
      init_platform();                   // Necessary for all programs.
      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.
    unsigned int * framePointer0 = (unsigned int *) FRAME_BUFFER_0_ADDR;
    //unsigned int * framePointer1 = ((unsigned int *) FRAME_BUFFER_0_ADDR)
+ 640*480;
    // 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<480; row++) {
      for(col=0; col<640; col++) {
            framePointer0[row*640 + col] = BLACK;    // init screen to black
            if (row > 440  && row < 444) {
                  framePointer0[row*640 + col] = GREEN;
            }
      }

    }
    // 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;
      setTankPositionGlobal(20);
      render(framePointer0);
    while (1) {
      input = getchar();
        if (XST_FAILURE == XAxiVdma_StartParking(&videoDMAController,
frameIndex,  XAXIVDMA_READ)) {
            xil_printf("vdma parking failed\n\r");
        }

        if (input == '2') {
            xil_printf(" please select an alien between 00 and 54: ");
            input = (unsigned int)getchar();
            unsigned int input2 = (unsigned int) getchar();
            input = ((input-48)*10 + (input2-48));
```

```
            xil_printf("%d\n\r", input);
            killAlien(input);
            drawAliens(framePointer0);
        }

        if (input == '3') {
            xil_printf("alien from bottom row randomly selected to fire
bullet\n\r");
            alienFire(framePointer0);
        }

        if (input == '4') {
                xil_printf("move tank left\n\r");
                setTankPositionGlobal(getTankPositionGlobal() - 5);
                drawTank(false, 0, 0, framePointer0);
        }

        if (input == '5') {
            if (!getBulletStatus()){
                xil_printf("tank bullet fired\n\r");
                setBulletStatus(true);
                point_t bullet;
                bullet.x = getTankPositionGlobal() + 15;
                bullet.y = 410;
                setTankBulletPosition(bullet);
                drawTankBullet(false, framePointer0);
            }
            else{
                xil_printf("A Tank Bullet already is in motion\n\r");
            }
        }

        if (input == '6') {
                xil_printf("move tank right\n\r");
                setTankPositionGlobal(getTankPositionGlobal() + 5);
                drawTank(false, 0, 0, framePointer0);
             }

            if (input == '7') {
                xil_printf(" please select which bunker to erode\n\r");
                input = getchar();
                if (input =='0' || input == '1' || input == '2' || input
== '3' ) {
                    // Check current Erosion state
                    int erosionState = getErosionDegree();
                    if (erosionState <-1 || erosionState > 3){
                          xil_printf("invalid erosion state\n\r");
                    } else{
                         erodeBunker(erosionState,  framePointer0,  input,
0, 0);
                    }
                } else {
                    xil_printf(" please enter a valid bunker\n\r");
                }
            }

            if (input == '8') {
```

```
                drawAliens(framePointer0);
            }

        if (input == '9') {
                point_t tank_bullet;
                if (getBulletStatus()) {
                    tank_bullet.x = getTankBulletPosition().x;
                    tank_bullet.y = getTankBulletPosition().y-3;
                }
                setTankBulletPosition(tank_bullet);
                drawTankBullet(true, framePointer0);
                updateBullets(framePointer0);

        }
    }
    cleanup_platform();

    return 0;
}
```

```c
/**
 *
 *
 * globals.c
 */

// Demonstrates one way to handle globals safely in C.
#include "globals.h"

#define LEFT 0;
#define RIGHT 1;


// Global variables
static unsigned short tankPosition;
static point_t tankBulletPosition;
static point_t alienBlockPosition;
static bool aliveAliens[55] = { 1 };
static bool legsOut = true;
static bool tankBulletFired = false;
static bool aliensDirection = RIGHT;
static int erosionDegree = -1;
static int farthestRightAlienColumn = 10;
static int farthestLeftAlienColumn = 0;
static int bottomRowAliens[11] = {44,45,46,47,48,49,50,51,52,53,54};
static bullet alienBullets[4];


void addAlienBullet() {
	// Some ints to help with computation.
	int i;
	int alienWidth = TOTAL_HORIZONTAL_DISTANCE_BETWEEN_ALIENS;
	int alienHeight = TOTAL_VERTICAL_DISTANCE_BETWEEN_ALIENS;
	int alienRows = ALIEN_ROWS;
	int alienCols = ALIEN_COLUMNS;

	for (i=0; i<alienRows--; i++) {
		if (!alienBullets[i].is_in_flight) {
			enum bullet_type t = rand()%2;
			int alienNumber = COLUMN_DEAD;

			int index = (rand()%alienCols);
			alienNumber = bottomRowAliens[index];
			if (alienNumber != -1) {	// clever programming, but it
is written in a way that causes not every tick to shoot....2162
				alienBullets[i].point.x = alienBlockPosition.x +
(index * alienWidth) + 10;
				alienBullets[i].point.y = alienBlockPosition.y +
(((alienNumber/alienCols) + 1)*alienHeight) - 5;
				alienBullets[i].is_in_flight = true;
				alienBullets[i].type = t;
				alienBullets[i].bullet_stage = BULLET_STAGE_0;
```

```
                        break;
                    }

            }
        }
}

// Returns the bullet struct for a given bullet.
bullet getAlienBullet(int index){
      return alienBullets[index];
}

// Moves an alien bullet down the screen by two pixels
void shiftAlienBullet(int index) {
      alienBullets[index].point.y += 2;
}

// The alien bullets cycle through stages
// This code increments the stage of an alien bullet.
void incrementBulletStage(int index) {
      // Increment the bullet's stage
      alienBullets[index].bullet_stage++;

      // Some macros used to help with readability
      int stage0 = BULLET_STAGE_0;
      int stage3 = BULLET_STAGE_3;
      int stage4 = BULLET_STAGE_4;

      // The T_type bullet resets after 3 iterations
      if (alienBullets[index].type == T_type &&
alienBullets[index].bullet_stage >= stage3)
            alienBullets[index].bullet_stage = stage0;

      // The S_type bullet resets after 4 iterations.
      if (alienBullets[index].type == S_type &&
alienBullets[index].bullet_stage >= stage4)
            alienBullets[index].bullet_stage = stage0;
}

// Disable the is_in_flight bool of an alien bullet.
// This allows the aliens to shoot another bullet.
void disableBullet(int i) {
      alienBullets[i].is_in_flight = false;
}

// Gets the farthest right column with at least one alive alien.
int getFarthestRightAlienColumn() {
      return farthestRightAlienColumn;
}

// Gets the farthest left column with at least one alive alien.
int getFarthestLeftAlienColumn() {
      return farthestLeftAlienColumn;
}

// Called when the aliens reach the edge of the screen.
// This switches the direction the aliens will move.
```

```
void switchAliensDirection() {
	aliensDirection = !aliensDirection;
}

// Returns whether the aliens should be moving left or right.
bool getAliensDirection() {
	return aliensDirection;
}

// Sets the x-coordinate of the tank
void setTankPositionGlobal(unsigned short val) {
  tankPosition = val;
}

// Gets the bool of tankBulletFired.
// If there currently is a tank bullet in the air,
// then this will be set to true and disable further
// tank bullets.
bool getBulletStatus(){
	return tankBulletFired;
}

// Sets the bool of tankBulletFired.
// If there currently is a tank bullet in the air,
// then this will be set to true and disable further
// tank bullets.
void setBulletStatus(bool newStatus){
	tankBulletFired = newStatus;
}

// Gets the degree of erosion of the bunker
int getErosionDegree(){
	return erosionDegree;
}

// Sets the degree of erosion of the bunker
void setErosionDegree(int newDegree){
	erosionDegree = newDegree;
}

// Gets the x-coordinate of the tank
unsigned short getTankPositionGlobal() {
  return tankPosition;
}

// Initialization code.
void initAliveAliens() {
	// Some ints to help us with calculations.
	int i;
	int rowSize = ALIEN_ROWS;
	int blockSize = NUMBER_OF_ALIENS;

	// The array of aliveAliens is initialized to be all true,
	// because all the aliens start off as being alive.
	for (i=0; i<blockSize; i++) {
		aliveAliens[i] = true;
	}
```

```
	// Initialize the alienBlockPosition
	alienBlockPosition.x = ALIEN_BLOCK_POSITION_INIT_X;
	alienBlockPosition.y = ALIEN_BLOCK_POSITION_INIT_Y;

	// Initialize the alienBullets to be available for firing.
	for (i=0; i<rowSize--; i++) {
		alienBullets[i].is_in_flight = false;
	}
}

// Gets the point of the alienBlockPosition
point_t getAlienBlockPosition() {
	return alienBlockPosition;
}

// Checks to see if the alien at index i is alive
bool getAliveAlien(unsigned int i){
	return aliveAliens[i];
}

void killAlien(unsigned int i){
	// The alien at index i is now dead.
	// The array of bools is updated to reflect this.
	aliveAliens[i] = false;

	// Some ints to help us with calculation
	int col;
	int row;
	int rowSize = ALIEN_ROWS;
	int colSize = ALIEN_COLUMNS;

	// Because an alien has been killed, we must check to see if there are
	// any aliens left in the column. If not, we need to shift the value of
	// farthesLeftAlien to the next available column.
	for (col=farthestLeftAlienColumn; col<colSize; col++) {
		for (row=0; row<rowSize; row++) {
			if (aliveAliens[(row*colSize) + col]) {
				if (col == farthestLeftAlienColumn) {
					// There exists an alien in the column of alien
i.
					// Set the iterators to end the loops.
					row=rowSize; col=colSize;
				} else {
					// There does not exist an alien in the column
of alien i.
					// Update the value of farthestLeftAlienColumn
to reflect this change.
					farthestLeftAlienColumn = col;
					// Set the iterators to end the loops.
					row=rowSize; col=colSize;
				}
			}
		}
	}

	// Because an alien has been killed, we must check to see if there are
```

```
	// any aliens left in the column. If not, we need to shift the value of
	// farthesRightAlien to the next available column
	for (col=farthestRightAlienColumn; col>0; col--) {
		for (row=0; row<rowSize; row++) {
			if (aliveAliens[(row * colSize) + col]) {
				if (col == farthestRightAlienColumn) {
					// There exists an alien in the column of alien
i.
					// Set the iterators to end the loops.
					row=rowSize; col=0;
				} else {
					// There does not exist an alien in the column
of alien i.
					// Update the value of farthestLeftAlienColumn
to reflect this change.
					farthestRightAlienColumn = col;
					// Set the iterators to end the loops.
					row=rowSize; col=0;
				}
			}
		}
	}

	col = i%ALIEN_COLUMNS;
	bool columnDead = true;

	// Now we must update the array of aliens that can shoot.
	// If an alien in the bottomRowAliens dies, then it is replaced by
	// the next available alien above it. If there are no aliens above it,
	// then the array is given a -1. The -1 serves as a flag that there is
no available
	// alien in the given column
	for (row = rowSize--; row>=0; row--) {
		if (aliveAliens[row*colSize + col]) {		// The given
alien above alien i is alive
			if (i == bottomRowAliens[col])			// The
alien that was killed was in bottomRowAliens.
				bottomRowAliens[col] = i-colSize;	// Maybe this
should be row*colSize + col?
				columnDead = false;					//
The exists an alien in the column.
				break;
		}
	}

	// If the column is dead, then the array needs to be given a -1.
	// -1 is defined as COLUMN_DEAD.
	if (columnDead) {
		bottomRowAliens[col] = COLUMN_DEAD;
	}
}

// Invert the bool that defines which way the alien legs are facing.
void switchLegsOut() {
	legsOut = !legsOut;
}
```

```
// Get the bool that determines which way the alien legs are facing.
bool getLegsOut() {
      return legsOut;
}

// Use this to update the alienBlockPosition.
// Its parameters are the new x,y coordinates of alienBlockPosition
void setAlienBlockPosition(unsigned int x, unsigned int y) {
      alienBlockPosition.x = x;
      alienBlockPosition.y = y;
}

// Use this to set the tankBulletPosition.
// Its parameter is the point with the new x,y coordinates of
tankBulletPosition
void setTankBulletPosition(point_t val) {
  tankBulletPosition.x = val.x;
  tankBulletPosition.y = val.y;
}

// Get the point of the tankBulletPosition.
point_t getTankBulletPosition() {
  return tankBulletPosition;
}
```

```
/*
 * render.c
 *
 *  Created on: Sep 26, 2015
 *      Author: superman
 */
#include "render.h"
#include "globals.h"
#include "bitmaps.h"

#define INITIAL_TANK_POSITION 20;

static unsigned int old_position = INITIAL_TANK_POSITION;
static const int pixelWidth = PIXEL_WIDTH;
static const int pixelHeight = PIXEL_HEIGHT;
static const int scalingFactor = SCALING_FACTOR;
static const int alienWidth = ALIEN_PIXEL_WIDTH;
static const int alienHeight = ALIEN_PIXEL_HEIGHT;
static const int totalAlienWidth = TOTAL_HORIZONTAL_DISTANCE_BETWEEN_ALIENS;
static const int totalAlienHeight = TOTAL_VERTICAL_DISTANCE_BETWEEN_ALIENS;
static bool aliensStillMovingHorizontally = true;


// Draws the tank on the screen.
void drawTank(bool staticTank, int startRow, int startCol, unsigned int *
framePointer0) {
	int row = 0;
	int col = 0;
	int tankHeight = TANK_HEIGHT;
	int tankWidth = TANK_WIDTH;
	int tankStart = TANK_ROW_START;
	int tankStop = TANK_ROW_STOP;

	if (staticTank){
		for (row = startRow; row < startRow + tankHeight; row++) {
			for (col = startCol; col < startCol + tankWidth; col++) {
				if ((tank_15x8[(row - startRow)/scalingFactor] &
(1<<((col-startCol)/scalingFactor))) == 0) {
					framePointer0[row*pixelWidth + col] = BLACK;
				} else {
					framePointer0[row*pixelWidth + col] = GREEN;
				}
			}
		}
	}
	else{
		int current_position = getTankPositionGlobal();

		// First we need to erase the old tank
		for (row = tankStart; row < tankStop; row++) {
			for (col = old_position; col < old_position + tankWidth;
col++) {
```

```
                        framePointer0[row*pixelWidth + col] = BLACK;
                  }
            }
            // Now draw the tank in its new position
            for (row = tankStart; row < tankStop; row++) {
                  for (col = current_position; col < current_position +
tankWidth; col++) {
                        if ((tank_15x8[(row - tankStart)/scalingFactor] &
(1<<((col-current_position)/scalingFactor))) == 0) {
                              framePointer0[row*pixelWidth + col] = BLACK;
                        } else {
                              framePointer0[row*pixelWidth + col] = GREEN;
                        }
                  }
            }
            // Update the position of the tank in memory
            old_position = current_position;
      }
}

// Erodes part of a given bunker in a given location.
void destroyPartOfBunker(const int array[], unsigned int * framePointer0, int
BunkerNumber, int segmentNumRow, int segmentCol){
      int row = 0;
      int col = 0;
      int incrementRow = segmentNumRow*12;
      int incrementCol = BunkerNumber*160 + segmentCol*12;
      // Now draw the tank in its new position
      for (row = 359 + incrementRow; row < 359 + incrementRow + 12; row++) {
           for (col = 56 + incrementCol; col < 56 + incrementCol + 12; col++)
{
                  if ((array[(row - 359 - incrementRow)/scalingFactor] &
(1<<((col-56 - incrementCol)/scalingFactor))) == 0) {
                        framePointer0[(row-12)*pixelWidth + col] = BLACK;
                  } else {
                        framePointer0[(row-12)*pixelWidth + col] = GREEN;
                  }
            }
      }
}

// Called when eroding a bunker by a given degree
void erodeBunker(int erosionState, unsigned int * framePointer0, int
BunkerNumber, int segmentNumRow, int segmentCol){
      if (erosionState == -1){
            setErosionDegree(3);
            destroyPartOfBunker(bunkerDamage3_6x6, framePointer0,
BunkerNumber, segmentNumRow, segmentCol);
      }
      else if (erosionState == 3){
            setErosionDegree(2);
            destroyPartOfBunker(bunkerDamage2_6x6, framePointer0,
BunkerNumber, segmentNumRow, segmentCol);
      }
      else if (erosionState == 2){
            setErosionDegree(1);
```

```
            destroyPartOfBunker(bunkerDamage1_6x6, framePointer0,
BunkerNumber, segmentNumRow, segmentCol);
      }

      else if (erosionState == 1){
            setErosionDegree(0);
            destroyPartOfBunker(bunkerDamage0_6x6, framePointer0,
BunkerNumber, segmentNumRow, segmentCol);
      }

}

// Draw a bunker on the screen.
void drawBunker(unsigned int * framePointer0, int BunkerNumber){
      int row = 0;
      int col = 0;
      int incrementY = BunkerNumber*160;
      // Now draw the tank in its new position
      for (row = 359; row < 395; row++) {
            for (col = 56 + incrementY; col < 104 + incrementY; col++) {
                  if ((bunker_24x18[(row - 359)/scalingFactor] & (1<<((col-56
+ incrementY)/scalingFactor))) == 0) {
                        framePointer0[row*pixelWidth + col] = BLACK;
                  } else {
                        framePointer0[row*pixelWidth + col] = GREEN;
                  }
            }
      }
}

// Draw all four bunkers on the screen.
void drawBunkers(unsigned int * framePointer0) {
      int i = 0;
      for (i = 0; i < 4; i++){
            drawBunker(framePointer0, i);
      }

}

// Draw a top row alien
void drawTopAlien(unsigned int * framePointer0, unsigned int start_x,
unsigned int start_y) {
      int row = 0;
      int col = 0;
      for (row = start_y; row < start_y+16; row++) {
            for (col = start_x; col < start_x+alienWidth; col++) {
                  if ((alien_top_out_12x8[(row - start_y)/scalingFactor] &
(1<<((col-start_x)/scalingFactor))) == 0){
                        framePointer0[row*pixelWidth + col] = BLACK;
                  } else {
                        framePointer0[row*pixelWidth + col] = WHITE;
                  }
            }
      }
}

// Draw a middle row alien
```

```
void drawMiddleAlien(unsigned int * framePointer0, unsigned int start_x,
unsigned int start_y) {
	int row = 0;
	int col = 0;
	for (row = start_y; row < start_y+16; row++) {
		for (col = start_x; col < start_x+alienWidth; col++) {
			if ((alien_middle_out_12x8[(row - start_y)/scalingFactor] &
(1<<((col-start_x)/scalingFactor))) == 0){
				framePointer0[row*pixelWidth + col] = BLACK;
			} else {
				framePointer0[row*pixelWidth + col] = WHITE;
			}
		}
	}
}

// Draw a bottom row alien
void drawBottomAlien(unsigned int * framePointer0, unsigned int start_x,
unsigned int start_y) {
	int row = 0;
	int col = 0;
	for (row = start_y; row < start_y+alienHeight; row++) {
		for (col = start_x; col < start_x+alienWidth; col++) {
			if ((alien_bottom_out_12x8[(row - start_y)/scalingFactor] &
(1<<((col-start_x)/scalingFactor))) == 0){
				framePointer0[row*pixelWidth + col] = BLACK;
			} else {
				framePointer0[row*pixelWidth + col] = WHITE;
			}
		}
	}
}

// Draw all the top row aliens
void drawTopAliens(unsigned int * framePointer0){
	int alienNumber = 0;
	for (alienNumber=0; alienNumber<11; alienNumber++) {
		if (!getAliveAlien(alienNumber)) {
			continue;
		}
		int alien_start_x = getAlienBlockPosition().x +
(alienNumber*totalAlienWidth);
		int alien_start_y = getAlienBlockPosition().y;
		drawTopAlien(framePointer0, alien_start_x, alien_start_y);
	}
}

// Draw all the middle row aliens
void drawMiddleAliens(unsigned int * framePointer0) {
	int alienNumber = 0;
	for (alienNumber=11; alienNumber<22; alienNumber++) {
		if (!getAliveAlien(alienNumber)) {
			continue;
		}
		int alien_start_x = getAlienBlockPosition().x + ((alienNumber-
11)*totalAlienWidth);
		int alien_start_y = getAlienBlockPosition().y + totalAlienHeight;
```

```
            drawMiddleAlien(framePointer0, alien_start_x, alien_start_y);
      }

      for (alienNumber=22; alienNumber<33; alienNumber++) {
            if (!getAliveAlien(alienNumber)) {
                  continue;
            }
            int alien_start_x = getAlienBlockPosition().x + ((alienNumber-
22)*totalAlienWidth);
            int alien_start_y = getAlienBlockPosition().y + 48;
            drawMiddleAlien(framePointer0, alien_start_x, alien_start_y);
      }
}

// Draw all the bottom row aliens
void drawBottomAliens(unsigned int * framePointer0) {
      int alienNumber = 0;
      for (alienNumber=33; alienNumber<44; alienNumber++) {
            if (!getAliveAlien(alienNumber)) {
                  continue;
            }
            int alien_start_x = getAlienBlockPosition().x + ((alienNumber-
33)*totalAlienWidth);
            int alien_start_y = getAlienBlockPosition().y + 72;
            drawBottomAlien(framePointer0, alien_start_x, alien_start_y);
      }
      for (alienNumber=44; alienNumber<55; alienNumber++) {
            if (!getAliveAlien(alienNumber)) {
                  continue;
            }
            int alien_start_x = getAlienBlockPosition().x + ((alienNumber-
44)*totalAlienWidth);
            int alien_start_y = getAlienBlockPosition().y + 96;
            drawBottomAlien(framePointer0, alien_start_x, alien_start_y);
      }
}


// Draw a top row alien with legs in
void drawTopAlien_in(unsigned int * framePointer0, unsigned int start_x,
unsigned int start_y) {
      int row = 0;
      int col = 0;
      for (row = start_y; row < start_y+16; row++) {
            for (col = start_x; col < start_x+alienWidth; col++) {
                  if ((alien_top_in_12x8[(row - start_y)/scalingFactor] &
(1<<((col-start_x)/scalingFactor))) == 0){
                        framePointer0[row*pixelWidth + col] = BLACK;
                  } else {
                        framePointer0[row*pixelWidth + col] = WHITE;
                  }
            }
      }
}

// Draw a middle row alien with legs in
```

```
void drawMiddleAlien_in(unsigned int * framePointer0, unsigned int start_x,
unsigned int start_y) {
	int row = 0;
	int col = 0;
	for (row = start_y; row < start_y+alienHeight; row++) {
		for (col = start_x; col < start_x+alienWidth; col++) {
			if ((alien_middle_in_12x8[(row - start_y)/scalingFactor] &
(1<<((col-start_x)/scalingFactor))) == 0){
				framePointer0[row*pixelWidth + col] = BLACK;
			} else {
				framePointer0[row*pixelWidth + col] = WHITE;
			}
		}
	}
}

// Draw a bottom row alien with legs in
void drawBottomAlien_in(unsigned int * framePointer0, unsigned int start_x,
unsigned int start_y) {
	int row = 0;
	int col = 0;
	for (row = start_y; row < start_y+alienHeight; row++) {
		for (col = start_x; col < start_x+alienWidth; col++) {
			if ((alien_bottom_in_12x8[(row - start_y)/scalingFactor] &
(1<<((col-start_x)/scalingFactor))) == 0){
				framePointer0[row*pixelWidth + col] = BLACK;
			} else {
				framePointer0[row*pixelWidth + col] = WHITE;
			}
		}
	}
}

// Draw all top row aliens with legs in
void drawTopAliens_in(unsigned int * framePointer0){
	int alienNumber = 0;
	for (alienNumber=0; alienNumber<11; alienNumber++) {
		if (!getAliveAlien(alienNumber)) {
			continue;
		}
		int alien_start_x = getAlienBlockPosition().x +
(alienNumber*totalAlienWidth);
		int alien_start_y = getAlienBlockPosition().y;
		drawTopAlien_in(framePointer0, alien_start_x, alien_start_y);
	}
}

// Draw all middle row aliens with legs in
void drawMiddleAliens_in(unsigned int * framePointer0) {
	int alienNumber = 0;
	for (alienNumber=11; alienNumber<22; alienNumber++) {
		if (!getAliveAlien(alienNumber)) {
			continue;
		}
		int alien_start_x = getAlienBlockPosition().x + ((alienNumber-
11)*totalAlienWidth);
		int alien_start_y = getAlienBlockPosition().y + totalAlienHeight;
```

```
            drawMiddleAlien_in(framePointer0, alien_start_x, alien_start_y);
      }

      for (alienNumber=22; alienNumber<33; alienNumber++) {
            if (!getAliveAlien(alienNumber)) {
                  continue;
            }
            int alien_start_x = getAlienBlockPosition().x + ((alienNumber-
22)*totalAlienWidth);
            int alien_start_y = getAlienBlockPosition().y + 48;
            drawMiddleAlien_in(framePointer0, alien_start_x, alien_start_y);
      }
}

// Draw all bottom row aliens with legs in
void drawBottomAliens_in(unsigned int * framePointer0) {
      int alienNumber = 0;
      for (alienNumber=33; alienNumber<44; alienNumber++) {
            if (!getAliveAlien(alienNumber)) {
                  continue;
            }
            int alien_start_x = getAlienBlockPosition().x + ((alienNumber-
33)*totalAlienWidth);
            int alien_start_y = getAlienBlockPosition().y + 72;
            drawBottomAlien_in(framePointer0, alien_start_x, alien_start_y);
      }
      for (alienNumber=44; alienNumber<55; alienNumber++) {
            if (!getAliveAlien(alienNumber)) {
                  continue;
            }
            int alien_start_x = getAlienBlockPosition().x + ((alienNumber-
44)*totalAlienWidth);
            int alien_start_y = getAlienBlockPosition().y + 96;
            drawBottomAlien_in(framePointer0, alien_start_x, alien_start_y);
      }
}

// Clear all aliens from the screen.
void resetAliens(unsigned int * framePointer0) {
      int alien_start_x = 0;
      int alien_start_y = 0;
      for (alien_start_x = getAlienBlockPosition().x; alien_start_x <
getAlienBlockPosition().x + (11*totalAlienWidth) + alienWidth;
alien_start_x++) {
            for (alien_start_y = getAlienBlockPosition().y; alien_start_y <
getAlienBlockPosition().y + 96 + alienHeight; alien_start_y++) {
                  framePointer0[alien_start_y*pixelWidth + alien_start_x] =
BLACK;
            }
      }
}

// Draw all living aliens on the screen.
void drawAliens(unsigned int * framePointer0) {
      resetAliens(framePointer0);
      int farthestLeftAlienPosition = getAlienBlockPosition().x +
((getFarthestLeftAlienColumn()) * totalAlienWidth);
```

```
	int farthestRightAlienPosition = getAlienBlockPosition().x +
((getFarthestRightAlienColumn() + 1) * totalAlienWidth);
	if ((farthestLeftAlienPosition < 20 || farthestRightAlienPosition > 620)
&& aliensStillMovingHorizontally) {
		setAlienBlockPosition(getAlienBlockPosition().x,
getAlienBlockPosition().y + 5);
		switchAliensDirection();
		aliensStillMovingHorizontally = false;
	} else {
		if (getAliensDirection()) {
			setAlienBlockPosition(getAlienBlockPosition().x + 5,
getAlienBlockPosition().y);
		} else {
			setAlienBlockPosition(getAlienBlockPosition().x - 5,
getAlienBlockPosition().y);
		}
		aliensStillMovingHorizontally = true;
	}
	if (getLegsOut()) {
		drawTopAliens(framePointer0);
		drawMiddleAliens(framePointer0);
		drawBottomAliens(framePointer0);
	} else {
		drawTopAliens_in(framePointer0);
		drawMiddleAliens_in(framePointer0);
		drawBottomAliens_in(framePointer0);
	}
	switchLegsOut();
}

// Draw a T_type bullet at stage 0
void drawTBullet0(bullet b, bool reset, unsigned int * framePointer0) {
	int rowStart = b.point.y;
	int colStart = b.point.x;
	int row, col;
	for (row = rowStart; row < rowStart + 10; row++){
		for (col = colStart; col < colStart + 6; col++){
			if ((t_bullet_3x5_0[(row - rowStart)/scalingFactor] &
(1<<((col-colStart)/scalingFactor))) == 0){
				framePointer0[row*pixelWidth + col] = BLACK;
			}
			else{
				if (reset) {
					framePointer0[row*pixelWidth + col] = BLACK;
				} else {
					framePointer0[row*pixelWidth + col] = WHITE;
				}
			}
		}
	}
}

// Draw a T_type bullet at stage 1
void drawTBullet1(bullet b, bool reset, unsigned int * framePointer0) {
	int rowStart = b.point.y;
	int colStart = b.point.x;
	int row, col;
```

```
	for (row = rowStart; row < rowStart + 10; row++){
		for (col = colStart; col < colStart + 6; col++){
			if ((t_bullet_3x5_1[(row - rowStart)/scalingFactor] &
(1<<((col-colStart)/scalingFactor))) == 0){
				framePointer0[row*pixelWidth + col] = BLACK;
			}
			else{
				if (reset) {
					framePointer0[row*pixelWidth + col] = BLACK;
				} else {
					framePointer0[row*pixelWidth + col] = WHITE;
				}
			}
		}
	}
}

// Draw a T_type bullet at stage 2
void drawTBullet2(bullet b, bool reset, unsigned int * framePointer0) {
	int rowStart = b.point.y;
	int colStart = b.point.x;
	int row, col;
	for (row = rowStart; row < rowStart + 10; row++){
		for (col = colStart; col < colStart + 6; col++){
			if ((t_bullet_3x5_2[(row - rowStart)/scalingFactor] &
(1<<((col-colStart)/scalingFactor))) == 0){
				framePointer0[row*pixelWidth + col] = BLACK;
			}
			else{
				if (reset) {
					framePointer0[row*pixelWidth + col] = BLACK;
				} else {
					framePointer0[row*pixelWidth + col] = WHITE;
				}
			}
		}
	}
}

// Draw a S_type bullet at stage 0
void drawSBullet0(bullet b, bool reset, unsigned int * framePointer0) {
	int rowStart = b.point.y;
	int colStart = b.point.x;
	int row, col;
	for (row = rowStart; row < rowStart + 10; row++){
		for (col = colStart; col < colStart + 6; col++){
			if ((s_bullet_3x5_0[(row - rowStart)/scalingFactor] &
(1<<((col-colStart)/scalingFactor))) == 0){
				framePointer0[row*pixelWidth + col] = BLACK;
			}
			else{
				if (reset) {
					framePointer0[row*pixelWidth + col] = BLACK;
				} else {
					framePointer0[row*pixelWidth + col] = WHITE;
				}
			}
```

```
            }
      }
}

// Draw a S_type bullet at stage 1
void drawSBullet1(bullet b, bool reset, unsigned int * framePointer0) {
      int rowStart = b.point.y;
      int colStart = b.point.x;
      int row, col;
      for (row = rowStart; row < rowStart + 10; row++){
            for (col = colStart; col < colStart + 6; col++){
                  if ((s_bullet_3x5_1[(row - rowStart)/scalingFactor] &
(1<<((col-colStart)/scalingFactor))) == 0){
                        framePointer0[row*pixelWidth + col] = BLACK;
                  }
                  else{
                        if (reset) {
                              framePointer0[row*pixelWidth + col] = BLACK;
                        } else {
                              framePointer0[row*pixelWidth + col] = WHITE;
                        }
                  }
            }
      }
}

// Draw a S_type bullet at stage 2
void drawSBullet2(bullet b, bool reset, unsigned int * framePointer0) {
      int rowStart = b.point.y;
      int colStart = b.point.x;
      int row, col;
      for (row = rowStart; row < rowStart + 10; row++){
            for (col = colStart; col < colStart + 6; col++){
                  if ((s_bullet_3x5_2[(row - rowStart)/scalingFactor] &
(1<<((col-colStart)/scalingFactor))) == 0){
                        framePointer0[row*pixelWidth + col] = BLACK;
                  }
                  else{
                        if (reset) {
                              framePointer0[row*pixelWidth + col] = BLACK;
                        } else {
                              framePointer0[row*pixelWidth + col] = WHITE;
                        }
                  }
            }
      }
}

// Draw a S_type bullet at stage 3
void drawSBullet3(bullet b, bool reset, unsigned int * framePointer0) {
      int rowStart = b.point.y;
      int colStart = b.point.x;
      int row, col;
      for (row = rowStart; row < rowStart + 10; row++){
            for (col = colStart; col < colStart + 6; col++){
                  if ((s_bullet_3x5_3[(row - rowStart)/scalingFactor] &
(1<<((col-colStart)/scalingFactor))) == 0){
```

```
                        framePointer0[row*pixelWidth + col] = BLACK;
                  }
                  else{
                        if (reset) {
                              framePointer0[row*pixelWidth + col] = BLACK;
                        } else {
                              framePointer0[row*pixelWidth + col] = WHITE;
                        }
                  }
            }
      }
}

// Draw all alien bullets
void drawBullets(unsigned int * framePointer0) {
      int i;
      for (i=0; i<4; i++) {
            if (getAlienBullet(i).is_in_flight) {
                  if (getAlienBullet(i).type == T_type) {
                        switch(getAlienBullet(i).bullet_stage) {
                        case (0):
                              drawTBullet0(getAlienBullet(i), false,
framePointer0);
                              break;
                        case (1):
                              drawTBullet1(getAlienBullet(i), false,
framePointer0);
                              break;
                        case (2):
                              drawTBullet2(getAlienBullet(i), false,
framePointer0);
                              break;
                        default:
                              break;
                        }
                  } else {
                        switch(getAlienBullet(i).bullet_stage) {
                        case (0):
                              drawSBullet0(getAlienBullet(i), false,
framePointer0);
                              break;
                        case (1):
                              drawSBullet1(getAlienBullet(i), false,
framePointer0);
                              break;
                        case (2):
                              drawSBullet2(getAlienBullet(i), false,
framePointer0);
                              break;
                        case (3):
                              drawSBullet3(getAlienBullet(i), false,
framePointer0);
                              break;
                        default:
                              break;
                        }
                  }
```

```
            }
      }
}

// Create a new bullet if there are less than 4 alien bullets on the screen,
// then display the new array of bullets on the screen.
void alienFire(unsigned int * framePointer0) {
      addAlienBullet();
      drawBullets(framePointer0);
}

// Update the locations of all alien bullets.
// This generally means moving the bullets down by two pixels.
// However, it also erases the bullets when they reach the bottom line.
void updateBullets(unsigned int * framePointer0) {
      int i;
      for (i=0; i<4; i++) {
            if (getAlienBullet(i).is_in_flight) {
                  drawTBullet0(getAlienBullet(i), true, framePointer0);
                  shiftAlienBullet(i);
                  incrementBulletStage(i);
                  if (getAlienBullet(i).point.y < 430) {
                        if (getAlienBullet(i).type == T_type) {
                              switch(getAlienBullet(i).bullet_stage) {
                              case (0):
                                    drawTBullet0(getAlienBullet(i), false,
framePointer0);
                                    break;
                              case (1):
                                    drawTBullet1(getAlienBullet(i), false,
framePointer0);
                                    break;
                              case (2):
                                    drawTBullet2(getAlienBullet(i), false,
framePointer0);
                                    break;
                              default:
                                    break;
                              }
                        } else {
                              switch(getAlienBullet(i).bullet_stage) {
                              case (0):
                                    drawSBullet0(getAlienBullet(i), false,
framePointer0);
                                    break;
                              case (1):
                                    drawSBullet1(getAlienBullet(i), false,
framePointer0);
                                    break;
                              case (2):
                                    drawSBullet2(getAlienBullet(i), false,
framePointer0);
                                    break;
                              case (3):
                                    drawSBullet3(getAlienBullet(i), false,
framePointer0);
                                    break;
```

```
                              default:
                                    break;
                              }
                        }
                  } else {
                        disableBullet(i);
                  }
            }
      }

}

// Draw the tank bullet on the screen.
void drawTankBullet(bool reset, unsigned int * framePointer0){
      int row;
     for (row = getTankBulletPosition().y-6; row < getTankBulletPosition().y;
row++) {
            if (reset) {
                  framePointer0[(row+3)*pixelWidth +
getTankBulletPosition().x] = BLACK;
                  framePointer0[(row+3)*pixelWidth +
getTankBulletPosition().x-1] = BLACK;
                  framePointer0[(row)*pixelWidth + getTankBulletPosition().x]
= WHITE;
                  framePointer0[(row)*pixelWidth + getTankBulletPosition().x-
1] = WHITE;
            } else {
                  framePointer0[row*pixelWidth + getTankBulletPosition().x] =
WHITE;
                  framePointer0[row*pixelWidth + getTankBulletPosition().x-1]
= WHITE;

            }
      }
      if (getTankBulletPosition().y < 30 || getTankBulletPosition().y > 480){
            // Blank the bullet
            for (row = getTankBulletPosition().y-6 ; row <
getTankBulletPosition().y; row++){
                  framePointer0[(row)*pixelWidth + getTankBulletPosition().x]
= BLACK;
                  framePointer0[(row)*pixelWidth + getTankBulletPosition().x-
1] = BLACK;
            }
            setBulletStatus(false);
      }
}

// Draw a letter. This is used to display the Score and Lives
void drawLetter(bool isNumber, int start_row, int start_col, const int
array[], unsigned int* framePointer0){
      int row;
      int col;
      for (row = start_row; row < start_row + 10; row++){
            for (col = start_col; col < start_col + 12; col++){
                  if ((array[(row - start_row)/scalingFactor] & (1<<((col-
start_col)/scalingFactor))) == 0){
                        framePointer0[row*pixelWidth + col] = BLACK;
```

```
                } else {
                        if (isNumber){
                                framePointer0[row*pixelWidth + col] = GREEN;
                        } else {
                                framePointer0[row*pixelWidth + col] = WHITE;
                        }
                }
        }
    }
}

// Draw the stats (score and lives) on the screen.
void drawStats(unsigned int * framePointer0){
    int startRow = 10;
    int startCol = 20;

    // Draw the Score
    drawLetter(false, startRow, startCol, S_6x5, framePointer0); // Draw S
    startCol += 15;
    drawLetter(false, startRow, startCol, C_6x5, framePointer0); // Draw C
    startCol += 15;
    drawLetter(false, startRow, startCol, O_6x5, framePointer0); // Draw O
    startCol += 15;
    drawLetter(false, startRow, startCol, R_6x5, framePointer0); // Draw R
    startCol += 15;
    drawLetter(false, startRow, startCol, E_6x5, framePointer0); // Draw E
    startCol += 30;
    drawLetter(true, startRow, startCol, O_6x5, framePointer0); // Draw
Zero
    startCol += 300;

    // Draw the Lives
    drawLetter(false, startRow, startCol, L_6x5, framePointer0); // Draw L
    startCol += 15;
    drawLetter(false, startRow, startCol, I_6x5, framePointer0); // Draw I
    startCol += 5;
    drawLetter(false, startRow, startCol, V_6x5, framePointer0); // Draw V
    startCol += 15;
    drawLetter(false, startRow, startCol, E_6x5, framePointer0); // Draw E
    startCol += 15;
    drawLetter(false, startRow, startCol, S_6x5, framePointer0); // Draw S
    startCol += 30;
    startRow -= 5;
    drawTank(true, startRow, startCol, framePointer0);
    startCol += 40;
    drawTank(true, startRow, startCol, framePointer0);
    startCol += 40;
    drawTank(true, startRow, startCol, framePointer0);



}

// Render the screen.
void render(unsigned int * framePointer0) {
    initAliveAliens();
    drawAliens(framePointer0);
```

```
        drawTank(false, 0, 0, framePointer0);
        drawBunkers(framePointer0);
        drawStats(framePointer0);
    }
```