Chapter 1: Space Invaders Overview

Section 1.1: Space Invaders History

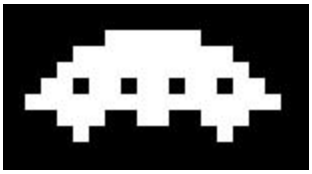
Space invaders may have been inspired by several other of video games (e.g. Breakout), but it was revolutionary. It was unique for its time. It was first released in 1978. When it was released it was one of the first video games to have enemies and levels that can be progressed through. While designing the game for the first time they planned on using human soldiers but they decided that they did not want to promote human killing so they changed them to aliens. Almost every video game that has been released since Space Invaders has added levels and typically has enemies that need to be killed. It also revolutionized the idea of games getting more difficult as you advance. This was originally a bug that turned out to be a great aspect to the game. As aliens were killed frames were easier for the processor to compute and the gameplay actually sped up. By the time there was only one alien left the game had become significantly faster.

Not only did Space Invaders revolutionize video games and inspire many more ideas for others, it also continued to evolve itself. It was first released in Japan in 1978. It did not make its way to the United States until 1980 where it was received ecstatically. It was released for arcade machines, Atari, and the original Nintendo Entertainment System. There was also a sequel released in 1980 and a third installment in the series in 1985. Since then there have been many spin offs of the original game.

Section 1.2: Game Play

There are four main objects in the game:

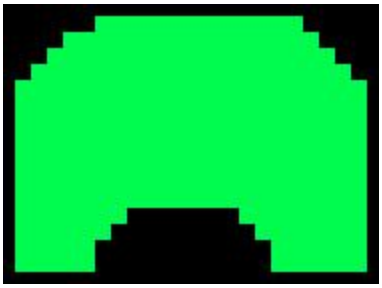
The Saucer is an enemy that you will encounter less frequently than the rest of the objects in the game. It only passes by every few seconds. It can be difficult to kill because it is farther away and moves faster than the rest of the aliens. It usually shoots as it goes across the screen. When it is shot it explodes. When you kill a saucer you will get a random bonus. It is not always worth the same score.



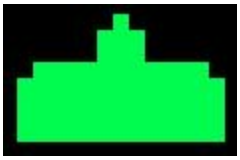
The Aliens are the main enemies encountered in the game. There are a total of 55 aliens in each level. They are all arranged in a swarm. They all move together in the same direction. They move across the screen and when they get to the opposite side they go down the screen towards the tank. If they get too close they will kill the tank. They also shoot randomly which if the tank is hit, it will kill the tank. If all of the aliens are killed. You advance to the next level.

When you kill an alien from the bottom two rows you get 10 points, for an alien in the middle two rows you get 20 points, and for the top row of aliens you get 30 points.

The Bunkers are spread across the screen between the aliens and the tank. There are four of them. As they are shot they will slowly be destroyed. This can be caused from shots from the aliens and the tanks. They are split into 10 segments which each have a health of 4 so if they are shot 4 times the segment will be destroyed. They are typically used to hide and protect the tank from bullets being shot from the aliens.



The Tank is the object actually controlled by the user. It can only move side to side and shoot. If the tank is hit by an alien or a bullet it will die. The game begins with three lives. So if you die three times you will get a Game Over and need to restart.



Section 1.3: Game Details and Specifications

Tanks Movement: The tank can only move horizontally. Its limits of movement are the edge of the screen in either horizontal direction. If the user attempts to move the tank off the screen, it will stay at the edge of the screen. If an alien bullet collides with the tank, the player loses a life.

Number of Bullets on Screen: The tank can only have one bullet on the screen at a time (the tank bullet must either collide with another object or leave the screen before another bullet can be fired). If the user attempts to fire a bullet when the tank bullet is on screen, the attempt is ignored. The aliens can have up to four alien bullets on screen.

Bullet Movement: Aliens bullets start right under a randomly selected alien on the bottom row and move downward. The tank's bullet starts right above the tank and exactly centered over the tank's gun. Tank bullets only move upward. Both bullets move at a constant rate.

Alien Movement: The aliens move from one side of the screen to the other. Once they reach the edge of the screen they move down one row and switch horizontal movement direction to go to the other side of the screen. Note that this will happen later when an entire column of aliens is destroyed. All aliens move as a swarm that is rectangular shaped. As the aliens move, they alternate between their "legs in" form and their "legs out" form. This gives them the appearance of walking.

Alien Types: There are three types of aliens. The first row (top) of the swarm are one type, the second and third row of the swarm are another type, and the bottom two rows are the final type. In addition to "legs in" and "legs out", the aliens have an exploding stage that flashes when they blow up.

Alien Destruction: When an alien is hit by a tank bullet, it blows up and is removed from the game. The bullet that hit it is also removed from the game.

Bunker Destruction: When a bunker is hit by any bullet (including a tank bullet), the segment that was hit progresses one erosion state and the bunker is re rendered accordingly. Once the erosion state of a segment of the bunker hits zero, that segment disappears and bullets can pass through it.

Saucer Behavior: The Saucer appears randomly. When it appears it moves across the screen horizontally then leaves the screen once it hits the edge. If the saucer is hit by a tank bullet it blows up and the user receives either 50, 100, or 150 points randomly chosen.

Scoreboard and Life Count: The word "SCORE" followed by a number is shown in the top left. When the user earns points, the score number is incremented accordingly. The top right has the word "LIVES" followed by three tank symbols. Each time the player loses a life (the tank is blown up by being hit by an alien bullet), one of the tank symbols is erased (from right to left).

Point Accumulation: The user gets 10 points for destroying an alien in the bottom two rows, 20 points for destroying an alien in the next two rows, and 30 points for destroying an alien in the top row. The user also accumulates either 50, 100, or 150 points (chosen at random) for destroying the saucer.

Ending The Game: The game ends when the player loses his/her final life or the aliens reach the bottom of the screen.

Bug Report: We had numerous bugs and challenges we had to overcome. The first challenge we ran into was deciding how to make the game objects twice as large. We considered remaking all the bitmaps to be twice as large, but then we decided that that would be a waste of time. Instead we created a function that takes in a normal size big maps and enlarges it as it draws it. This worked out very well.

The next challenge we had was avoiding duplication of code in the draw function. At first we considered having a draw function for each type of object in the game. We liked the simplicity of this idea, but then realized that the code for actually drawing the pixels to the frame buffer would be extremely redundant. After additional though, we made just one draw function that takes in a enum for the type of object to draw. We still had to use a switch statement to correlate the object type (alien_top_legs_out, tank, saucer, etc) with the object's height, width, and bitmap data, but this solution dramatically decreased our boilerplate code. Once this function was complete, it was used as a helper function to perform almost all of the requirements for the rendering.

One bug that our code had was our bullets having artifacts left behind. This was a simple matter of adjusting our draw algorithm to first draw the old bullet in black draw the new bullet in the correct color. A similar strategy was used to eliminate all artifacts in the game.

Apart from these challenges, we found that everything worked very smoothly. The different levels of abstraction in our design helped to alleviate a lot of headache. We discovered that a good design strategy can save a lot of time in coding.

```
/*
 * 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.
 *
 */

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

#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xaxivdma.h"
#include "xio.h"
#include "time.h"
#include "unistd.h"
#include "aliens.h"
#include "screenState.h"

#define DEBUG

#define KILL_ALIEN '2'                  // Keyboard press to kill an alien
#define ZERO_ASCII_VAL '0'              // Used to get the integer value of a key press
#define USE_AS_MOST_SIG_DIGIT 10        // Puts the digit given into the tens place
#define NUM_ALIENS 55                   // Number of aliens in the swarm
#define ALIEN_FIRE '3'                  // Keyboard press to have alien shoot
#define MOVE_TANK_LEFT '4'              // Keyboard press to move the tank to the left
#define TANK_FIRE '5'                   // Keyboard press to have tank shoot
#define MOVE_TANK_RIGHT '6'             // Keyboard press to move the tank to the right
#define ERODE_BUNKER '7'                // Keyboard press to erode a bunker of choice
#define NUM_BUNKERS 4                   // Number of bunkers in the game
#define MOVE_SWARM '8'                  // Keyboard press to advance the swarm
#define UPDATE_BULLETS '9'              // Keyboard press to update the position of the
bullets and redraw

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

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;

    screenState_setFramePointer(framePointer0);

    // 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");
    }

    //Initializes all of the first values for the game and for the screen
    screenState_init();

    //Constantly run
    while (1) {
        char input = getchar(); // Recieve the character input from the UART
        if(input == KILL_ALIEN) { // Keyboard press to kill an alien
            char firstChar = getchar(); // get the most significant digit of the number
of alien to kill
            char secondChar = getchar(); // get the least significant digit of the number
of alien to kill
            uint16_t val = ((firstChar - ZERO_ASCII_VAL) * USE_AS_MOST_SIG_DIGIT) +
(secondChar - ZERO_ASCII_VAL); // Gets the integer value of the number entered
            if(val < NUM_ALIENS) { // If the value calculated is between 0 and 54
                screenState_killAlien(val); // Kills the alien selected
            }
        } else if(input == ALIEN_FIRE) { // Keyboard press to have alien shoot
            screenState_alienFire(); // Tells an alien to fire
        } else if(input == MOVE_TANK_LEFT){ // Keyboard press to move the tank to the left
            screenState_moveTankLeft(); // Moves the tank to the left
        } else if(input == TANK_FIRE){ // Keyboard press to have tank shoot
            screenState_tankFire(); // Has tank shoot
        } else if(input == MOVE_TANK_RIGHT){ // Keyboard press to move the tank to the
right
            screenState_moveTankRight(); // Moves the tank to the right
        } else if(input == ERODE_BUNKER) { // Keyboard press to erode a bunker of choice
            char val = getchar(); // Get the value of bunker to erode
            uint8_t valInt = val - ZERO_ASCII_VAL; // Gets the integer value of the
button pressed
            if(valInt < NUM_BUNKERS) { // Checks to make sure that the number is between
0 and 3
                screenState_erodeBunker(valInt); // Erode the bunker entered
            }
        } else if(input == MOVE_SWARM) { // Keyboard press to advance the swarm
```

```
            screenState_moveSwarm(); // Advances the swarm
        } else if(input == UPDATE_BULLETS) { // Keyboard press to update the position of
the bullets and redraw
            screenState_updateBullets(); // Updates the position of the bullets
        }
    }
    cleanup_platform(); // Cleans up everything to be ready to run again

    return 0;
}
```

```
#include "aliens.h"
#include "screenState.h"

#define ALIEN_HEIGHT 8                          // Height of an alien
#define ALIEN_WIDTH 12                          // Width of an alien
#define SAUCER_HEIGHT 7                         // Height of a saucer
#define SAUCER_WIDTH 16                         // Width of a saucer
#define BUNKER_HEIGHT 18                        // Height of a bunker
#define BUNKER_WIDTH 24                         // Width of a bunker
#define EXPLOSION_HEIGHT 10                     // Height of an explosion
#define TANK_HEIGHT 8                           // Height of the tank
#define TANK_WIDTH 15                           // Width of the tank
#define BULLET_HEIGHT 5                         // Height of a bullet
#define BULLET_WIDTH 3                          // Width of a bullet
#define BUNKER_DAMAGE_DIMENSION 6               // Height and width of a bunker segment
#define ORIGINAL_SCREEN_WIDTH 640               // The screen width before pixels have
been doubled in size
#define DOUBLE 2                                // Used to double a value
#define EVERY_OTHER_COLUMN 2                    // Used to decide what kind of alien to
draw
#define NUM_ALIENS_IN_ROW 11                    // The number of aliens in each row
#define TOTAL_NUM_ALIENS 55                     // Total number of aliens in the swarm
#define THIRD_ALIEN_ROW 2                       // Used to see if value is in the third
row
#define FOURTH_ALIEN_ROW 3                      // Used to see if value is in the fourth
row
#define FIFTH_ALIEN_ROW 4                       // Used to see if value is in the fifth
row
#define HORIZONTAL_SPACE_BETWEEN_ALIENS 16      // The space put between each alien

//Used to compress the array into a single 32-bit value
#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 ) )

//Used to compress the array into a single 16-bit value
#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 ) )

//Used to compress the array into a single 16-bit value with all values smashed together
#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 ) )

//Used to compress the array into a single 16-bit value with all values smashed together
```

```
#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 ) )

//Used to compress the array into a single 32-bit
#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 ) )

//Used to compress the array into a single 16-bit value with all values smashed together
#define PACKWORD6(b5,b4,b3,b2,b1,b0) \
((b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ) )

//Used to compress the array into a single 16-bit value with all values smashed together
#define PACKWORD3(b2,b1,b0) \
((b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ) )

//The value for each pixel for the sprite for the saucer
static const uint32_t saucer_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)
};

//The value for each pixel for the sprite for an explosion
static const uint32_t alien_explosion_12x10[] =
{
    PACKWORD12(0,0,0,0,0,0,1,0,0,0,0,0),
    PACKWORD12(0,0,0,1,0,0,1,0,0,0,1,0),
    PACKWORD12(1,0,0,1,0,0,0,0,0,1,0,0),
    PACKWORD12(0,1,0,0,1,0,0,0,1,0,0,0),
    PACKWORD12(0,0,0,0,0,0,0,0,0,0,1,1),
    PACKWORD12(1,1,0,0,0,0,0,0,0,0,0,0),
    PACKWORD12(0,0,0,1,0,0,0,1,0,0,1,0),
    PACKWORD12(0,0,1,0,0,0,0,0,1,0,0,1),
    PACKWORD12(0,1,0,0,0,1,0,0,1,0,0,0),
    PACKWORD12(0,0,0,0,0,1,0,0,0,0,0,0)
};

//The value for each pixel for the sprite for the top aliens with legs in
static const int alien_top_in_12x8[] =
{
    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(0,0,0,1,0,0,0,0,1,0,0,0)
};

//The value for each pixel for the sprite for the top aliens with legs out
static const int alien_top_out_12x8[] =
{
    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(0,0,0,1,1,1,1,1,1,1,0,0),
    PACKWORD12(0,0,1,1,0,1,1,0,1,1,0,0),
    PACKWORD12(0,0,1,1,1,1,1,1,1,1,1,0,0),
    PACKWORD12(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)
};

//The value for each pixel for the sprite for the middle aliens with legs in
static const int alien_middle_in_12x8[] =
{
    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)
};

//The value for each pixel for the sprite for the middle aliens with legs out
static const uint32_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)
};

//The value for each pixel for the sprite for the bottom aliens with legs in
static const uint32_t alien_bottom_in_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(0,0,1,1,1,0,0,1,1,1,0,0),
    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)
};
```

```
//The value for each pixel for the sprite for the bottom aliens with legs out
static const uint32_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)
};

//The value for each pixel for the sprite for the tank
static const uint32_t tank_15x8[] =
{
    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)
};

//The value for each pixel for the sprite for the first alien bullet
static const uint32_t alien_bullet_0_3x5[] =
{
    PACKWORD3(0,1,1),
    PACKWORD3(0,1,0),
    PACKWORD3(1,1,0),
    PACKWORD3(0,1,1),
    PACKWORD3(1,1,0)
};

//The value for each pixel for the sprite for the second alien bullet
static const uint32_t alien_bullet_1_3x5[] =
{
    PACKWORD3(0,0,0),
    PACKWORD3(0,1,0),
    PACKWORD3(0,1,0),
    PACKWORD3(1,1,1),
    PACKWORD3(0,1,0)
};

//The value for each pixel for the sprite for the tank bullets
static const uint32_t tank_bullet_3x5[] =
{
    PACKWORD3(0,1,0),
    PACKWORD3(1,1,1),
    PACKWORD3(0,1,0),
    PACKWORD3(0,1,0),
    PACKWORD3(0,1,0)
};


// Shape of the entire bunker.
```

```c
static const uint32_t bunker_24x18[] =
{
    PACKWORD24(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0),
    PACKWORD24(0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
    PACKWORD24(0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0),
    PACKWORD24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
    PACKWORD24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1)
};

// These are the blocks that comprise the bunker and each time a bullet
// strikes one of these blocks, you erod the block as you sequence through
// these patterns.
static const uint32_t bunkerDamage0_6x6[] =
{
    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)
};

//The value for each pixel for the sprite for the second hit on bunker segment
static const uint32_t bunkerDamage1_6x6[] =
{
    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)
};

//The value for each pixel for the sprite for the first hit on bunker segment
static const uint32_t bunkerDamage2_6x6[] =
{
    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)
};

//The value for each pixel for the sprite for the full health or destroyed bunker segment
```

```
static const uint32_t bunkerDamage3_6x6[] =
{
    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)
};

//Draw an object on screen.
//NOTE x, y coordinates are not pixels but chunk of four pixels
//(aka 1,1 will print at pixel 2,2 and 5,5 will print at pixel 10,10)
void aliens_drawObject(uint32_t x, uint32_t y, aliens_objectId object, aliens_color
color){
    uint32_t row = 0; // The row where it will draw
    uint32_t col = 0; // The column where it will draw
    uint32_t height;  // The height of the object to be drawn
    uint32_t width;   // The width of the object to be drawn
    uint32_t * objectData; // The type of object to be drawn
    switch(object){ // Find which object would like to be drawn
    case alien_top: // If it is the top alien
        height = ALIEN_HEIGHT; // Use the alien height
        width = ALIEN_WIDTH; // Use the Alien width
        if(x % EVERY_OTHER_COLUMN) { // Decide whether to do legs in or legs out
            objectData = (uint32_t *)alien_top_out_12x8; // Assign the type of object to
draw
        } else {
            objectData = (uint32_t *)alien_top_in_12x8; // Assign the type of object to
draw
        }
        break;
    case alien_middle: // If drawing the middle aliens
        height = ALIEN_HEIGHT; // Use the alien height
        width = ALIEN_WIDTH; // Use the Alien width
        if(x % EVERY_OTHER_COLUMN) { // Decide whether to do legs in or legs out
            objectData = (uint32_t *)alien_middle_out_12x8; // Assign the type of object
to draw
        } else {
            objectData = (uint32_t *)alien_middle_in_12x8; // Assign the type of object
to draw
        }
        break;
    case alien_bottom: // If drawing the bottom aliens
        height = ALIEN_HEIGHT; // Use the alien height
        width = ALIEN_WIDTH; // Use the Alien width
        if(x % EVERY_OTHER_COLUMN) { // Decide whether to do legs in or legs out
            objectData = (uint32_t *)alien_bottom_out_12x8; // Assign the type of object
to draw
        } else {
            objectData = (uint32_t *)alien_bottom_in_12x8; // Assign the type of object
to draw
        }
        break;
    case tank: // If drawing the tank
        height = TANK_HEIGHT; // Use the height of the tank
        width = TANK_WIDTH; // Use the Width of the tank
        objectData = (uint32_t *)tank_15x8; // Assign the type of object to draw
```

```
        break;
    case saucer: // If drawing the saucer
        height = SAUCER_HEIGHT; // Use the saucer Height
        width = SAUCER_WIDTH; // Use the saucer width
        objectData = (uint32_t *)saucer_16x7; // Assign the type of object to draw
        break;
    case bunker: // If drawing the entire bunker
        height = BUNKER_HEIGHT; // Use the full bunker height
        width = BUNKER_WIDTH; // Use the full bunker height
        objectData = (uint32_t *)bunker_24x18; // Assign the type of object to draw
        break;
    case explosion: // If drawing the explosion
        height = EXPLOSION_HEIGHT; // Use the Explosion Height
        width = ALIEN_WIDTH; // The the alien width
        objectData = (uint32_t *)alien_explosion_12x10; // Assign the type of object to
draw
        break;
    case alien_bullet_0: // If drawing the first type of alien bullet
        height = BULLET_HEIGHT; // Use height of the bullet
        width = BULLET_WIDTH; // Use the bullet width
        objectData = (uint32_t *)alien_bullet_0_3x5; // Assign the type of object to draw
        break;
    case alien_bullet_1: // If drawing the second type of alien bullet
        height = BULLET_HEIGHT; // Use height of the bullet
        width = BULLET_WIDTH; // Use the bullet width
        objectData = (uint32_t *)alien_bullet_1_3x5; // Assign the type of object to draw
        break;
    case tank_bullet: // If drawing the tank bullet
        height = BULLET_HEIGHT; // Use height of the bullet
        width = BULLET_WIDTH; // Use the bullet width
        objectData = (uint32_t *)tank_bullet_3x5; // Assign the type of object to draw
        break;
    case bunker_damage0: // If drawing a segment of the bunker
        height = BUNKER_DAMAGE_DIMENSION; // Use the bunker width/height
        width = BUNKER_DAMAGE_DIMENSION; // Use the bunker width/height
        objectData = (uint32_t *)bunkerDamage0_6x6; // Assign the type of object to draw
        break;
    case bunker_damage1: // If drawing a segment of the bunker
        height = BUNKER_DAMAGE_DIMENSION; // Use the bunker width/height
        width = BUNKER_DAMAGE_DIMENSION; // Use the bunker width/height
        objectData = (uint32_t *)bunkerDamage1_6x6; // Assign the type of object to draw
        break;
    case bunker_damage2: // If drawing a segment of the bunker
        height = BUNKER_DAMAGE_DIMENSION; // Use the bunker width/height
        width = BUNKER_DAMAGE_DIMENSION; // Use the bunker width/height
        objectData = (uint32_t *)bunkerDamage2_6x6; // Assign the type of object to draw
        break;
    case bunker_damage3: // If drawing a segment of the bunker
        height = BUNKER_DAMAGE_DIMENSION; // Use the bunker width/height
        width = BUNKER_DAMAGE_DIMENSION; // Use the bunker width/height
        objectData = (uint32_t *)bunkerDamage3_6x6; // Assign the type of object to draw
        break;
    default:
        //Do nothing for now
        break;
    }
```

```
    // Iterate through the dimensions of the object being drawn
    for (row = y; row < height + y; row++) {
        for (col = x; col < width + x; col++) { // Iterate through the dimensions of the
object being drawn
            if ((objectData[row - y] & (1 << (width - 1 - col + x )))) { // If the pixel
should be drawn
                screenState_getFramePointer()[(row * DOUBLE) * ORIGINAL_SCREEN_WIDTH +
(col * DOUBLE)] = color; // colors the pixel that it is supposed to
                screenState_getFramePointer()[(row * DOUBLE + 1) * ORIGINAL_SCREEN_WIDTH
+ (col * DOUBLE)] = color; // Colors the pixel next to it to double the size of each pixel
                screenState_getFramePointer()[(row * DOUBLE) * ORIGINAL_SCREEN_WIDTH +
(col * DOUBLE + 1)] = color; // Colors the pixel below to double the size
                screenState_getFramePointer()[(row * DOUBLE + 1) * ORIGINAL_SCREEN_WIDTH
+ (col * DOUBLE + 1)] = color; // Colors the pixel next to the last

            } else { // Draw black everywhere else
                screenState_getFramePointer()[(row * DOUBLE) * ORIGINAL_SCREEN_WIDTH +
(col * DOUBLE)] = 0; // colors the pixel that it is supposed to
                screenState_getFramePointer()[(row * DOUBLE + 1) * ORIGINAL_SCREEN_WIDTH
+ (col * DOUBLE)] = 0; // Colors the pixel next to it to double the size of each pixel
                screenState_getFramePointer()[(row * DOUBLE) * ORIGINAL_SCREEN_WIDTH +
(col * DOUBLE + 1)] = 0; // Colors the pixel below to double the size
                screenState_getFramePointer()[(row * DOUBLE + 1) * ORIGINAL_SCREEN_WIDTH
+ (col * DOUBLE + 1)] = 0; // Colors the pixel next to the last
            }
        }
    }
}

//Draw the alien swarm using the x,y values of the top left alien
//The array is to be able to draw the aliens that are alive
void aliens_drawSwarm(uint32_t x, uint32_t y, uint8_t alien_array[], uint32_t color)
{
    uint16_t alienX = x; // horizontal origin of the swarm
    uint16_t alienY = y; // vertical origin of the swarm
    uint16_t i = 0; // Initialize variable for the for-loop
    for (i = 0; i < TOTAL_NUM_ALIENS; i++) { // Iterates through all of the aliens
        if(i % NUM_ALIENS_IN_ROW == 0 && i != 0) { // if needs to start drawing the next
row of aliens
            alienX = x; // Reset the horizontal index
            alienY = y + i; // Increase the vertical index to go to next row
        }
        if(alien_array[i]) { // If the alien is still alive
            if(i / NUM_ALIENS_IN_ROW == 0) { // If it is in the first row
                aliens_drawObject(alienX, alienY, alien_top, color); // Draw the top
aliens
            } else if(i / NUM_ALIENS_IN_ROW == 1 || i / NUM_ALIENS_IN_ROW ==
THIRD_ALIEN_ROW) { // if it is in the middle rows
                aliens_drawObject(alienX, alienY, alien_middle, color); // Draw the
middle aliens
            } else if(i / NUM_ALIENS_IN_ROW == FOURTH_ALIEN_ROW || i / NUM_ALIENS_IN_ROW
== FIFTH_ALIEN_ROW) { //If it is the last rows
                aliens_drawObject(alienX, alienY, alien_bottom, color); // Draw the
bottom aliens
            }
        }
        alienX += HORIZONTAL_SPACE_BETWEEN_ALIENS; // Increase the horizontal index to
draw the next index
```

```
    }
}
```

```c
//This file has functions for drawing objects in the game

#include <stdint.h>

#define ALIENS_WHITE_COLOR 0xFFFFFFFF // Color codex for white
#define ALIENS_GREEN_COLOR 0x0000FF00 // Color codex for green
#define ALIENS_BLACK_COLOR 0x00000000 // Color codex for black

//Enum to define all of the alien objects
typedef enum aliens_objectId {
    alien_top,              // Alien sprite at the top
    alien_middle,           // Alien sprite in the middle
    alien_bottom,           // Alien sprite at the bottom
    tank,                   // Tank sprite
    saucer,                 // Saucer sprite
    bunker,                 // Bunker sprite
    explosion,              // Alien Explosion sprite
    alien_bullet_0,         // First type of alien bullet
    alien_bullet_1,         // Second type of alien bullet
    tank_bullet,            // Tank bullet
    bunker_damage0,         // Third bullet hit bunker
    bunker_damage1,         // Second Bullet hit bunker
    bunker_damage2,         // First bullet hit the bunker
    bunker_damage3          // Bunker piece has full health, or destroyed depending on
color
}aliens_objectId;

//Enum to define the colors of used to draw
typedef enum aliens_color {
    WHITE = ALIENS_WHITE_COLOR,    // Draw using white
    GREEN = ALIENS_GREEN_COLOR,    // Draw using green
    BLACK = ALIENS_BLACK_COLOR     // Draw using black
}aliens_color;


//Draw an object on screen.
//NOTE x, y coordinates are not pixels but chunk of four pixels
//(aka 1,1 will print at pixel 2,2 and 5,5 will print at pixel 10,10)
void aliens_drawObject(uint32_t x, uint32_t y, objectId object, uint32_t color);

//Draw the alien swarm using the x,y values of the top left alien
//The array is to be able to draw the aliens that are alive
void aliens_drawSwarm(uint32_t x, uint32_t y, uint8_t alien_array[], uint32_t color);
```

```c
#include "scoreBoard.h"
#include <stdint.h>
#include "screenState.h"
#include "aliens.h"

//For storing a bit map row that is 14 pixels wide
#define PACKWORD14(b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0) \
((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 ) )

//For storing a bit map row that is 12 pixels wide
#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 ) )

//For storing a bit map row that is 10 pixels wide
#define PACKWORD10(b9,b8,b7,b6,b5,b4,b3,b2,b1,b0) \
((b9  << 9 ) | (b8  << 8 ) |                                \
 (b7  << 7 ) | (b6  << 6 ) | (b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1
<< 1 ) | (b0  << 0 ) )

#define TANK_LIFES_START_X 252              //X position for lives shown by tank symbols
on screen
#define LIFE_TANK_WIDTH 18                  //Y position for lives shown by tank symbols
on screen
#define NUMBER_HEIGHT 14                    //The height of a number bitmap
#define WIDE_CHAR 14                        //The width of a wide character
#define MEDIUM_CHAR 12                      //The width of a medium width character
#define NARROW_CHAR 10                      //The width of a narrow character (I)
#define WHITE  0xFFFFFFFF                   //Color white
#define GREEN  0x0000FF00                   //Color green
#define BLACK  0x00000000                   //Color Black
#define SCORE_BEGIN_X 25                    //X pixel location of scoreboard
#define SCORE_BEGIN_Y 20                    //Y pixel location of scoreboard
#define LIVES_BEGIN_X 400                   //Starting x position for LIVES word
#define LIFE_TANK_Y  SCORE_BEGIN_Y / 2 - 1  //Starting y position for LIVES word
#define SCREEN_WIDTH 640                    //Width in pixels of the screen

//Bitmap for the character 0
const uint32_t score_digit_0[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
```

```
};
//Bitmap for the character 1
const uint32_t score1[NUMBER_HEIGHT] =
{
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,1,1,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,1,1,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character 2
const uint32_t score2[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character 3
const uint32_t score3[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};
```

```
//Bitmap for the character 4
const uint32_t score4[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};


//Bitmap for the character 5
const uint32_t score5[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character 6
const uint32_t score6[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};
```

```
//Bitmap for the character 7
const uint32_t score7[NUMBER_HEIGHT] =
{
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD12(0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character 8
const uint32_t score8[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character 9
const uint32_t score9[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};
```

```
//Bitmap for the character ,
const uint32_t scoreComma[NUMBER_HEIGHT] =
{
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,0,0,1,1,0,0,0,0),
        PACKWORD12(0,0,0,0,1,1,0,0,0,0,0,0),
        PACKWORD12(0,0,0,0,1,1,0,0,0,0,0,0),
};

//Bitmap for the character S
const uint32_t scoreS[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character C
const uint32_t scoreC[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character O
```

```
const uint32_t scoreO[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),

};

//Bitmap for the character R
const uint32_t scoreR[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character E
const uint32_t scoreE[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character L
```

```
const uint32_t scoreL[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character I
const uint32_t scoreI[NUMBER_HEIGHT] =
{
        PACKWORD10(0,0,0,0,0,0,0,0,0,0),
        PACKWORD10(0,0,0,0,0,0,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,1,1,0,0,0,0),
        PACKWORD10(0,0,0,0,0,0,0,0,0,0),
        PACKWORD10(0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character V
const uint32_t scoreV[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,0,0,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,0,0,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,1,1,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,1,1,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character G
const uint32_t scoreG[NUMBER_HEIGHT] =
```

```
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character A
const uint32_t scoreA[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,0,0,1,1,1,1,1,1,0,0,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,1,1,1,1,1,1,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,1,1,0,0,0,0,0,0,1,1,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Bitmap for the character M
const uint32_t scoreM[NUMBER_HEIGHT] =
{
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(1,1,1,1,1,1,0,0,1,1,1,1,0,0),
        PACKWORD14(1,1,1,1,1,1,0,0,1,1,1,1,0,0),
        PACKWORD14(1,1,0,0,0,0,1,1,0,0,0,0,1,1),
        PACKWORD14(1,1,0,0,0,0,1,1,0,0,0,0,1,1),
        PACKWORD14(1,1,0,0,0,0,1,1,0,0,0,0,1,1),
        PACKWORD14(1,1,0,0,0,0,1,1,0,0,0,0,1,1),
        PACKWORD14(1,1,0,0,0,0,1,1,0,0,0,0,1,1),
        PACKWORD14(1,1,0,0,0,0,1,1,0,0,0,0,1,1),
        PACKWORD14(1,1,0,0,0,0,1,1,0,0,0,0,1,1),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
        PACKWORD14(0,0,0,0,0,0,0,0,0,0,0,0,0,0),
};

//Draws an object from a scoreboard bitmap
void scoreBoard_drawObject(uint32_t x, uint32_t y, scoreBoard_objectId object, uint32_t
color){
```

```
    uint32_t row = 0;       //Row of character
    uint32_t col = 0;       //Column of character
    uint32_t height;        //Height of character
    uint32_t width;         //Width of character
    uint32_t * objectData;  //Pointer to the bitmap for the character
    height = NUMBER_HEIGHT; //Set the height (all character have the same height)

    switch(object){
    case scoreBoard_objectId_zero://For char 0
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)score_digit_0;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_one://For char 1
        width = MEDIUM_CHAR;    //Set the width
        objectData = (uint32_t *)score1;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_two://For char 2
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)score2;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_three://For char 3
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)score3;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_four://For char 4
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)score4;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_five://For char 5
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)score5;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_six://For char 6
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)score6;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_seven://For char 7
        width = MEDIUM_CHAR;    //Set the width
        objectData = (uint32_t *)score7;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_eight://For char 8
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)score8;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_nine://For char 9
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)score9;//Set the object data pointer to the
cooresponding bitmap
        break;
```

```
    case scoreBoard_objectId_s://For char S
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)scoreS;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_c://For char C
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)scoreC;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_o://For char O
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)scoreO;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_r://For char R
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)scoreR;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_e://For char E
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)scoreE;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_comma://For char ','
        width = MEDIUM_CHAR;    //Set the width
        objectData = (uint32_t *)scoreComma;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_l://For char L
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)scoreL;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_i://For char I
        width = NARROW_CHAR;    //Set the width
        objectData = (uint32_t *)scoreI;//Set the object data pointer to the
cooresponding bitmap
        break;
    case scoreBoard_objectId_v://For char V
        width = WIDE_CHAR;      //Set the width
        objectData = (uint32_t *)scoreV;//Set the object data pointer to the
cooresponding bitmap
        break;
    default:
        //Do nothing for now
        break;
    }
    //For each row and column in bitmap
    for (row = y; row < height + y; row++) {
        for (col = x; col < width + x; col++) {//For each row
            if ((objectData[row - y] & (1 << (width - 1 - col + x )))) {//If this pixel
should be filled in
                screenState_getFramePointer()[(row) * SCREEN_WIDTH + (col)] =
color;//Draw it with correct color
            } else {
```

```
                screenState_getFramePointer()[(row) * SCREEN_WIDTH + (col)] =
BLACK;//Otherwise draw it in black
            }
        }
    }
}

void scoreBoard_drawScoreBoard(){
    uint32_t i = 0;//This will be used to determin which character in the word we are
drawing (for example C in S C O R E will be i=1)

    //Draw the word SCORE
    scoreBoard_drawObject(SCORE_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_s, WHITE);//Draw the character
    i++;//Move to next character position
    scoreBoard_drawObject(SCORE_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_c, WHITE);//Draw the character
    i++;//Move to next character position
    scoreBoard_drawObject(SCORE_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_o, WHITE);//Draw the character
    i++;//Move to next character position
    scoreBoard_drawObject(SCORE_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_r, WHITE);//Draw the character
    i++;//Move to next character position
    scoreBoard_drawObject(SCORE_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_e, WHITE);//Draw the character
    i++;//Move to next character position
    i++;//Move to next character position
    scoreBoard_drawObject(SCORE_BEGIN_X + (i) * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_zero, GREEN);//Draw the character

    //Draw the word LIVES
    i = 0;//Reset Character Position
    scoreBoard_drawObject(LIVES_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_l, WHITE);//Draw the character
    i++;//Move to next character position
    scoreBoard_drawObject(LIVES_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_i, WHITE);//Draw the character
    i++;//Move to next character position
    scoreBoard_drawObject(LIVES_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_v, WHITE);//Draw the character
    i++;//Move to next character position
    scoreBoard_drawObject(LIVES_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_e, WHITE);//Draw the character
    i++;//Move to next character position
    scoreBoard_drawObject(LIVES_BEGIN_X + i * WIDE_CHAR,SCORE_BEGIN_Y,
scoreBoard_objectId_s, WHITE);//Draw the character

    //Draw the life tanks
    i = 0;//Reset character position
    aliens_drawObject(TANK_LIFES_START_X + i * LIFE_TANK_WIDTH, LIFE_TANK_Y, tank,
GREEN);//Draw the character
    i++;//Move to next character position
    aliens_drawObject(TANK_LIFES_START_X + i * LIFE_TANK_WIDTH, LIFE_TANK_Y, tank,
GREEN);//Draw the character
    i++;//Move to next character position
    aliens_drawObject(TANK_LIFES_START_X + i * LIFE_TANK_WIDTH, LIFE_TANK_Y, tank,
GREEN);//Draw the character
```

```
}
```

```c
//This enum is used to tell the draw function which object to draw
typedef enum scoreBoard_objectId {
    scoreBoard_objectId_s,          //Chracter 'S'
    scoreBoard_objectId_c,          //Chracter 'C'
    scoreBoard_objectId_o,          //Chracter 'O'
    scoreBoard_objectId_r,          //Chracter 'R'
    scoreBoard_objectId_e,          //Chracter 'E'
    scoreBoard_objectId_comma,      //Chracter ','
    scoreBoard_objectId_zero,       //Chracter '0'
    scoreBoard_objectId_one,        //Chracter '1'
    scoreBoard_objectId_two,        //Chracter '2'
    scoreBoard_objectId_three,      //Chracter '3'
    scoreBoard_objectId_four,       //Chracter '4'
    scoreBoard_objectId_five,       //Chracter '5'
    scoreBoard_objectId_six,        //Chracter '6'
    scoreBoard_objectId_seven,      //Chracter '7'
    scoreBoard_objectId_eight,      //Chracter '8
    scoreBoard_objectId_nine,       //Chracter '9'
    scoreBoard_objectId_l,          //Chracter 'L'
    scoreBoard_objectId_i,          //Chracter 'I'
    scoreBoard_objectId_v,          //Chracter 'V'
}scoreBoard_objectId;

//This function draws the entire scoreboard
void scoreBoard_drawScoreBoard();
```

```c
//This file is for functions for clearing the screen and storing the state of the screen

#include "screenState.h"
#include "aliens.h"
#include <stdlib.h>
#include "scoreBoard.h"

#define TANK_Y 200                                          //Starting Y position of the tank
#define TANK_START_POS 160                                  //Starting X position of the tank
#define BUNKER_HEIGHT 150                                   //Heigh of bunker in coords
#define BUNKER_SPACE 75                                     //Space between each bunker
#define BUNKER_OFFSET 30                                    //How far the first bunker is
from the left side of the screen
#define BULLET_OFF_SCREEN 0                                 //Indicates that the bullet with
this value is off screen
#define BULLET_OFFSET 55                                    //Space between left side of
screen and left most possible alien bullet
#define GROUND_Y 450                                        //Y Coordinate of the ground
#define GROUND_START_X 50                                   //Starting X coord of the ground
#define GROUND_STOP_X 640 - 50                              //Where the ground stops X
#define ALIEN_BULLET_STOP 230                               //How far the alien bullets can
fall
#define SCREEN_HEIGHT_PIXELS 480                            //Height of screen in pixels
#define SCREEN_WIDTH_PIXELS 640                             //Width of screen in pixels
#define NUMBER_OF_BULLETS_TOTAL 5                           //Total number of bullets in game
#define ALIEN_SWARM_SIZE 55                                 //Aliens in a swarm
#define NUMBER_OF_DIMENSION_OF_BULLET_COORDINATES 2         //The bullets are on a 2
dimentional plane
#define NUMBER_OF_BUNKERS 4                                 //Number of bunkers
#define NUMBER_OF_BUNKER_SEGMENTS 10                        //Number of bunker segments in
the game
#define SWARM_START_X 72                                    //X coord of starting point of
the swarm
#define SWARM_START_Y 20                                    //Y coord of starting point of
the swarm
#define TANK_BULLET_INDEX 0                                 //Index in the bullet array of
the tank's bullet
#define TANK_BULLET_TYPE 3                                  //Index in the bullet type array
of a tank bullet
#define NOT_ERRODED 4                                       //Bunker not erroded reference
#define SWARM_RIGHT_EDGE 144                                //How far right the swarm can go
#define SWARM_MOVE_DISTANCE_PER_STEP 3                      //Coordinate steps per tick of
aliens
#define TANK_RIGHT_LIMIT 305                                //How far right the tank can go
#define TANK_BULLET_START_X_OFFSET_FROM_TANK 4              //Distance from edge of tank to
edge of bullet X
#define TANK_BULLET_START_Y_OFFSET_FROM_TANK 6              //Distance from edge of tank to
edge of bullet Y
#define BULLETS_MOVEMENT_PER_TICK 3                         //How many coordinates the
bullets move per tick
#define BULLET_MIN_POS_Y 20                                 //How far down the bullets can
fall
#define NUMBER_OF_SEGMENTS_IN_BUNKER 4                      //How many shots you need to
destroy a segment of the bunker
#define BUNKER_SEGMENT_PIXEL_WIDTH 6                        //How wide each bunker segment is
#define FIRST_EROSION_STATE 3                               //If it has been shot once
#define SECOND_EROSION_STATE 2                              //If it has been shot twice
```

```
#define THIRD_EROSION_STATE 1                           //If it has been shot three times
#define EMPTY_BUNKER_SEGMENT 9                          //The middle two bottom segments
do not exist. We use this value to skip them
#define EMPTY_BUNKER_SEGMENT_OFFSET 12                  //Skips to the last bunker segment

#define BULLET_X_COORDINATE 0                           //First position in the array is
the X coordinate of the bullet
#define BULLET_Y_COORDINATE 1                           //Second position in the array is
the Y coordinate of the bullet

#define FIRST_ALIEN_BULLET_INDEX 1                      //First alien bullet
#define SECOND_ALIEN_BULLET_INDEX 2                     //Second alien bullet
#define THIRD_ALIEN_BULLET_INDEX 3                      //Third alien bullet

#define NUMBER_OF_ALIEN_BULLET_TYPES 2                  //There are two types of alien
bullets

#define RANDOME_SEED 354354351                          //Used to calculate a random
alien to shoot a bullet

#define ALIEN_BULLET_TYPE_ONE 0                         //First alien bullet type
#define ALIEN_BULLET_TYPE_TWO 1                         //Second alien bullet type

#define ALIENS_PER_ROW 11                               //Number of aliens in each row
#define ALIEN_WIDTH_IN_SWARM 16                         //Distance between each alien
#define ALIEN_BULLET_START_OFFSET 5                     //The offset for the first alien
bullet shooter

static uint32_t screenState_swarmX = 0;                 //The current X coordinate of the
swarm
static uint32_t screenState_swarmY = 0;                 //The current Y coordinate of the
swarm
static uint8_t screenState_swarmArray[ALIEN_SWARM_SIZE];//Keeps track of which aliens are
still alive
static uint8_t goneDown = 0;                            //Makes it so the swarm does not
move diagonally at the side of the screen
static uint32_t tankPos = TANK_START_POS;               //The X position of the tank
static uint32_t bullets[NUMBER_OF_BULLETS_TOTAL]
[NUMBER_OF_DIMENSION_OF_BULLET_COORDINATES];    //Keeps track of all of the bullets
static uint8_t bunkers[NUMBER_OF_BUNKERS]
[NUMBER_OF_BUNKER_SEGMENTS];                            //Keeps track of all of the
bunkers
static uint32_t * framePointer_g;                       //Global variable that is used to
access the screen

uint8_t bulletTypes[NUMBER_OF_BULLETS_TOTAL];           //Keeps track of all of the
bullet types

//Draws the line at the bottom of the screen
static void drawGround(){
    uint32_t row = GROUND_Y; //Sets the Y coordinate of the ground
    uint32_t col = GROUND_START_X; //Sets the beginning x coordinate of the ground
    for( ; col < GROUND_STOP_X; col++){ //Iterates through the length of the ground
        screenState_getFramePointer()[(row) * SCREEN_WIDTH_PIXELS + (col)] = GREEN;
//Draws the pixel green
    }
}
```

```c
//Initaializes everything needed for the game and the screen
void screenState_init(){
    srand(RANDOME_SEED); //Sets the random seed to get a random value
    uint16_t row, col, i; //Initializes the variables needed for this function
    for (row = 0; row < SCREEN_HEIGHT_PIXELS; row++) { //Iterates through the height of
the screen
        for (col = 0; col < SCREEN_WIDTH_PIXELS; col++) { //Iterates through the width of
the screen
            screenState_getFramePointer()[row * SCREEN_WIDTH_PIXELS + col] = BLACK; //
Draws the pixel black
        }
    }
    for(i = 0; i < ALIEN_SWARM_SIZE; i++) { //Iterates through all of the aliens
        screenState_swarmArray[i] = 1; //Marks them as alive
    }
    screenState_setSwarmX(SWARM_START_X); //Sets the initial X position of the swarm
    screenState_setSwarmY(SWARM_START_Y); //Sets the initial Y position of the swarm

    bulletTypes[TANK_BULLET] = TANK_BULLET_TYPE; //Initializes the bullet for the tank to
the be the tank bullet
    for(i = FIRST_ALIEN_BULLET_INDEX; i < NUMBER_OF_BULLETS_TOTAL; i++) { //Iterates
through the rest of the bullets
        bulletTypes[i] = rand() % NUMBER_OF_ALIEN_BULLET_TYPES; //Gives them a random
value to define the type of alien bullet
    }

    //Draws the swarm in its initial position
    aliens_drawSwarm(screenState_getSwarmX(), screenState_getSwarmY(),
screenState_getSwarmArray(), WHITE);

    for(i = 0; i < NUMBER_OF_BUNKERS; i++) { //Iterates through all of the bunkers
        aliens_drawObject((i * BUNKER_SPACE) + BUNKER_OFFSET, BUNKER_HEIGHT, bunker,
GREEN); //Draws each bunker
        uint8_t j; //Initializes variable used to iterate through each segment
        for(j = 0; j < NUMBER_OF_BUNKER_SEGMENTS; j++) { //Iterates through each segment
of the bunker
            bunkers[i][j] = NOT_ERRODED; //Marks that it has not been shot
        }
    }
    screenState_moveTankLeft(); //Draws the tank in its initial position
    for(i = 0; i < NUMBER_OF_BULLETS_TOTAL; i++) { //Iterates through all of the bullets
        bullets[i][BULLET_X_COORDINATE] = 0; //Initializes the position
        bullets[i][BULLET_Y_COORDINATE] = 0; //Initializes the position
    }
    scoreBoard_drawScoreBoard(); //Draws the scoreboard
    drawGround(); //Draws the ground
}

//Sets the global variable for the frame pointer
void screenState_setFramePointer(unsigned int * framePointer){
    framePointer_g = framePointer; //Sets the value to the passed pointer
}

//Returns the frame pointer
unsigned int * screenState_getFramePointer(){
    return framePointer_g; //Returns the fram pointer
}
```

```
//Sets the swarm X position
void screenState_setSwarmX(uint32_t swarmX){
    screenState_swarmX = swarmX; //Assigns the value to the passed value
}
//Returns the swarm X position
uint32_t screenState_getSwarmX(){
    return screenState_swarmX;//Returns the swarm X position
}

//Sets the swarm Y position
void screenState_setSwarmY(uint32_t swarmY){
    screenState_swarmY = swarmY; //Assigns the value to the passed value
}

//Returns the swarm Y position
uint32_t screenState_getSwarmY(){
    return screenState_swarmY;//Returns the swarm Y position
}

//Moves the swarm to its next position
void screenState_moveSwarm() {
    //Erases the current swarm
    aliens_drawSwarm(screenState_getSwarmX(), screenState_getSwarmY(),
screenState_getSwarmArray(), BLACK);
    //If the swarm is on the left or right edge and has not been moved down yet
    if((screenState_getSwarmX() == SWARM_RIGHT_EDGE || screenState_getSwarmX() == 0) &&
!goneDown) {
        screenState_setSwarmY(screenState_getSwarmY() + SWARM_MOVE_DISTANCE_PER_STEP);
//Moves the Y position closer to the tank
        aliens_drawSwarm(screenState_getSwarmX(), screenState_getSwarmY(),
screenState_getSwarmArray(), WHITE); //Redraws the swarm
        goneDown = 1; //One for true
        return;//Leaves the function
    }
    if(screenState_swarmY % NUMBER_OF_DIMENSION_OF_BULLET_COORDINATES) { //Decides if it
should move left or right
        screenState_setSwarmX(screenState_getSwarmX() + SWARM_MOVE_DISTANCE_PER_STEP);
//Assigns the new x value
    } else {
        screenState_setSwarmX(screenState_getSwarmX() - SWARM_MOVE_DISTANCE_PER_STEP);
//Assigns the new X value
    }
    goneDown = 0;//Zero for false
    aliens_drawSwarm(screenState_getSwarmX(), screenState_getSwarmY(),
screenState_getSwarmArray(), WHITE); //Redraws the swarm  in the new location
}

//Returns the array of the swarm
uint8_t * screenState_getSwarmArray() {
    return screenState_swarmArray; //Returns the swarm array
}

//Moves the tank to the right
void screenState_moveTankRight(){
    if(tankPos >= TANK_RIGHT_LIMIT ){ //If it is the farthest right it can go
        return;//Leave the function
    }
```

```
    aliens_drawObject(tankPos, TANK_Y, tank, BLACK);//Erase the tank
    tankPos++;//Increment the x position
    aliens_drawObject(tankPos, TANK_Y, tank, GREEN);//Redraw the tank in the new position
}

//Move the tank left
void screenState_moveTankLeft(){
    if(tankPos <= 0){ //if it is the farthest left it can go
        return; //Leave the function
    }
    aliens_drawObject(tankPos, TANK_Y, tank, BLACK); //Erase the tank
    tankPos--; //Move the position left
    aliens_drawObject(tankPos, TANK_Y, tank, GREEN); //Redraw the tank in the new position
}

//Kill the alien specified
void screenState_killAlien(uint8_t alienToKill){
    aliens_drawSwarm(screenState_getSwarmX(), screenState_getSwarmY(),
screenState_getSwarmArray(), BLACK); //Erase the swarm
    screenState_swarmArray[alienToKill] = 0; //Kill the alien
    aliens_drawSwarm(screenState_getSwarmX(), screenState_getSwarmY(),
screenState_getSwarmArray(), WHITE); //Redraw the swarm without the alien
}

//Draw all of the bullets
void drawBullets(uint32_t color) {
    uint8_t i; //Initialize the variable that will iterate through the bullets
    for(i = 0; i < NUMBER_OF_BULLETS_TOTAL; i++) { //Iterate through all of the bullets
        if(bullets[i][1]) { //if the bullet Y position is not 0
            switch (bulletTypes[i]){ //Decide which bullet it is
            case ALIEN_BULLET_TYPE_ONE: //If it is the first type of bullet
                aliens_drawObject(bullets[i][BULLET_X_COORDINATE], bullets[i]
[BULLET_Y_COORDINATE], alien_bullet_0, color); //Draw the bullet
                break;//Leave the switch
            case ALIEN_BULLET_TYPE_TWO: //If it is the second type of bullet
                aliens_drawObject(bullets[i][BULLET_X_COORDINATE], bullets[i]
[BULLET_Y_COORDINATE], alien_bullet_1, color); //Draw the bullet
                break;//Leave the switch
            case TANK_BULLET_TYPE: //If it is the tank bullet
                aliens_drawObject(bullets[i][BULLET_X_COORDINATE], bullets[i]
[BULLET_Y_COORDINATE], tank_bullet, color); //Draw the bullet
                break;//Leave the switch
            default://If not any of the previous options
                break; //Leave the switch
            }
        }
    }
}

//Have alien shoot
void screenState_alienFire() {
    uint8_t i; //Initialize variable for for loop
    for(i = 1; i < NUMBER_OF_BULLETS_TOTAL; i++) { //Iterate through all of the bullets
        if(!bullets[i][BULLET_Y_COORDINATE]) { // If there is a bullet that has not been
shot
            //Get a random alien X position
            bullets[i][BULLET_X_COORDINATE] = screenState_getSwarmX() + (rand() %
ALIENS_PER_ROW) * ALIEN_WIDTH_IN_SWARM + ALIEN_BULLET_START_OFFSET;
```

```
            bullets[i][BULLET_Y_COORDINATE] = screenState_getSwarmY() + BULLET_OFFSET;
//Get the y position
            drawBullets(WHITE); //Draw the bullet
            return; //Leave the function
        }
    }
}

//Shoot from tank
void screenState_tankFire() {
    if(!bullets[TANK_BULLET_INDEX][BULLET_Y_COORDINATE]) { //If the tank has not already
shot recently
        bullets[TANK_BULLET_INDEX][BULLET_X_COORDINATE] = tankPos +
TANK_BULLET_START_X_OFFSET_FROM_TANK; //Assign the X position
        bullets[TANK_BULLET_INDEX][BULLET_Y_COORDINATE] = TANK_Y -
TANK_BULLET_START_Y_OFFSET_FROM_TANK; //Assign the Y position
        drawBullets(WHITE); //Draw the bullets
    }
}

//Update the position of the bullets
void screenState_updateBullets() {
    drawBullets(BLACK); //Erase the bullets
    uint8_t i; //Initialize the variable for the for loop
    if(bullets[TANK_BULLET_INDEX][BULLET_Y_COORDINATE]) { //If the tank bullet is in
motion
        if(bullets[TANK_BULLET_INDEX][BULLET_Y_COORDINATE] > BULLET_MIN_POS_Y){ //If the
bullet is still on the screen
            bullets[TANK_BULLET_INDEX][BULLET_Y_COORDINATE] -= BULLETS_MOVEMENT_PER_TICK;
//Update the bullet position
        } else{
            bullets[TANK_BULLET_INDEX][BULLET_Y_COORDINATE] = 0; //Make sure the bullet
is off the screen
        }
    }
    for(i = FIRST_ALIEN_BULLET_INDEX; i < NUMBER_OF_BULLETS_TOTAL; i++) { //Iterate
through all of the alien bullets
        if(bullets[i][1] && (bullets[i][BULLET_Y_COORDINATE] < ALIEN_BULLET_STOP)) { //If
the bullet is on the screen
            bullets[i][1] += BULLETS_MOVEMENT_PER_TICK; //Update the position
        } else {
            bullets[i][0] = 0; //Take the bullet completely off the screen
            bullets[i][1] = 0; //Take the bullet completely off the screen
        }
    }
    drawBullets(WHITE); Redraw all of the bullets
}

//Erode a segment of a bunker
void erodeChunk(uint8_t chunk, uint8_t state, uint8_t bunker) {
    //Get the X coordinate of the segment
    uint32_t chunkX = (bunker * BUNKER_SPACE) + BUNKER_OFFSET +
(BUNKER_SEGMENT_PIXEL_WIDTH * (chunk % NUMBER_OF_SEGMENTS_IN_BUNKER));
    if(chunk == EMPTY_BUNKER_SEGMENT) { //If it is the empty piece of the segment
        chunkX += EMPTY_BUNKER_SEGMENT_OFFSET; //Go to the last segment
    }
    uint32_t chunkY = BUNKER_HEIGHT + (BUNKER_SEGMENT_PIXEL_WIDTH * (chunk /
NUMBER_OF_SEGMENTS_IN_BUNKER)); //Get the vertical position of the segment
```

```
    switch (state) { //Decide which state the segment is in
    case NOT_ERRODED: //If never been shot
        aliens_drawObject(chunkX, chunkY, bunker_damage3, BLACK); //Erase the segment
        aliens_drawObject(chunkX, chunkY, bunker_damage2, GREEN); //Redraw the new view
of the segment
        break; //Leave switch
    case FIRST_EROSION_STATE: //If it has been shot once
        aliens_drawObject(chunkX, chunkY, bunker_damage2, BLACK); //Erase the segment
        aliens_drawObject(chunkX, chunkY, bunker_damage1, GREEN); //Redraw the new view
of the segment
        break; //Leave switch
    case SECOND_EROSION_STATE: //If it has been shot twice
        aliens_drawObject(chunkX, chunkY, bunker_damage1, BLACK); //Erase the segment
        aliens_drawObject(chunkX, chunkY, bunker_damage0, GREEN); //Redraw the new view
of the segment
        break; //Leave switch
    case THIRD_EROSION_STATE: //If it has been shot three times
        aliens_drawObject(chunkX, chunkY, bunker_damage0, BLACK); //Erase the segment
        break; //Leave switch
    default: //If none of the above
        break; //Leave switch
    }
}

//Bunker has been hit by a bullet
void screenState_erodeBunker(uint8_t bunker) {
    uint8_t i; //Initialize the variable for the for loop
    for(i = 0; i < NUMBER_OF_BUNKER_SEGMENTS; i++) { //Iterate through all of the
segments in the bunker
        uint8_t state = bunkers[bunker][i]; //Get the state of the segment
        if(state) { //If the chunk is not destroyed
            erodeChunk(i, state, bunker); //Erode that segment
            bunkers[bunker][i]--; //Decrement the health of bunker
            return; //leave the function
        }
    }
}
```

```c
#include <stdint.h>

//Returns the frame pointer
unsigned int * screenState_getFramePointer();

//Sets the global variable for the frame pointer
void screenState_setFramePointer(unsigned int * framePointer);

//Initaializes everything needed for the game and the screen
void screenState_init();

//Moves the swarm to its next position
void screenState_moveSwarm();

//Sets the swarm X position
void screenState_setSwarmX(uint32_t swarmX);

//Returns the swarm X position
uint32_t screenState_getSwarmX();

//Sets the swarm Y position
void screenState_setSwarmY(uint32_t swarmY);

//Returns the swarm Y position
uint32_t screenState_getSwarmY();

//Returns the array of the swarm
uint8_t * screenState_getSwarmArray();

//Moves the tank to the right
void screenState_moveTankRight();

//Move the tank left
void screenState_moveTankLeft();

//Kill the alien specified
void screenState_killAlien(uint8_t alienToKill);

//Have alien shoot
void screenState_alienFire();

//Shoot from tank
void screenState_tankFire();

//Update the position of the bullets
void screenState_updateBullets();

//Bunker has been hit by a bullet
void screenState_erodeBunker(uint8_t bunker);
```