Microcontrollers with C
ELEC2850-05
4/22/2021
Final Report

Space Invaders

Jesse Khoury

Brennan Reamer

Albert Duart Pallas

I. Brief Description of the project and game

In Space Invaders you play as a character at the bottom of the screen fighting off enemies as they move down towards you. The objective is to shoot all 20 enemies before they either shoot you or reach the bottom of the screen. If either of these conditions occur you lose a life. If you lose 3 lives it is game over.

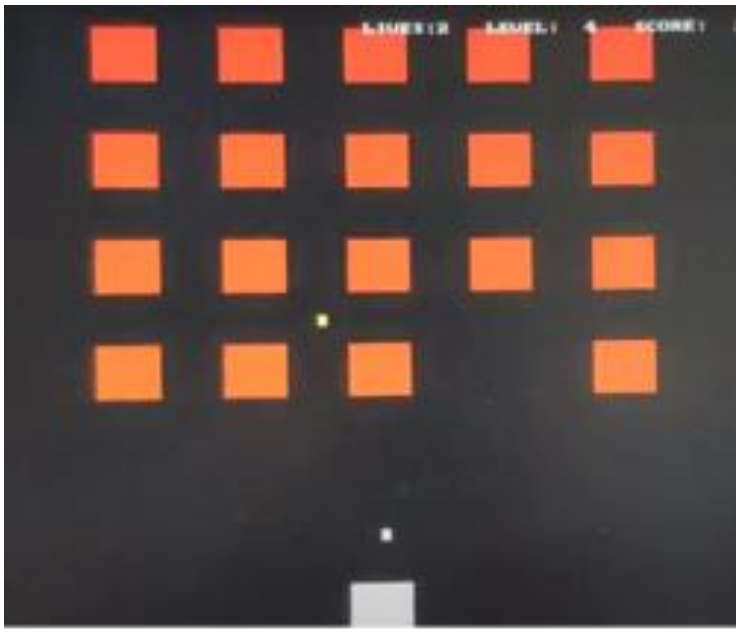


Figure 1: Starting a new game

After defeating all twenty enemies, the player reaches the next level. The rate of fire from the enemies increases each level, as does the rate of movement. The result is each level is more challenging as the enemy becomes more dangerous as the game progresses.

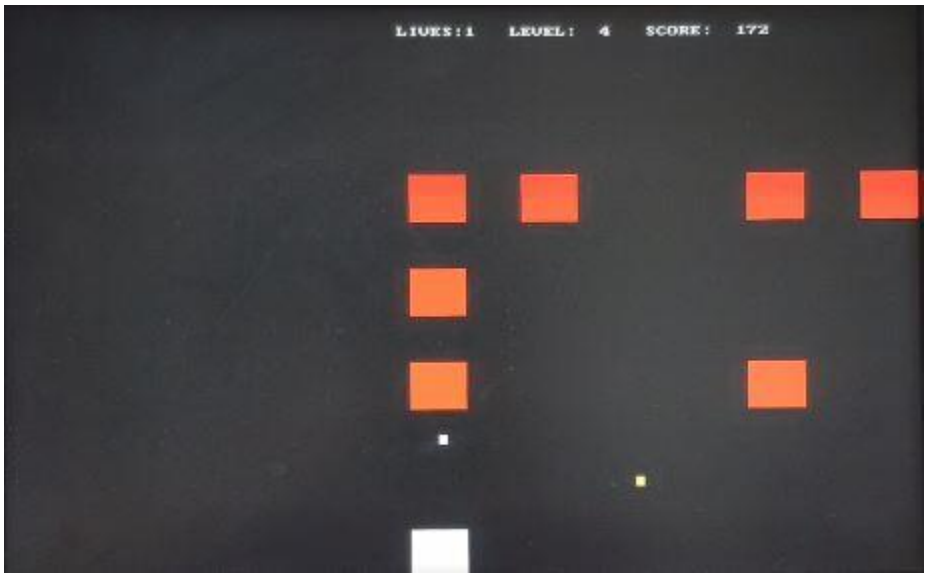


Figure 2: Game play showing projectiles and updated values at the top of the screen

The player's score increments after shooting an enemy. The points awarded per enemy are incremented one per level (on level 1 enemies are worth one point, on level 2 enemies are worth 2 points, ect).

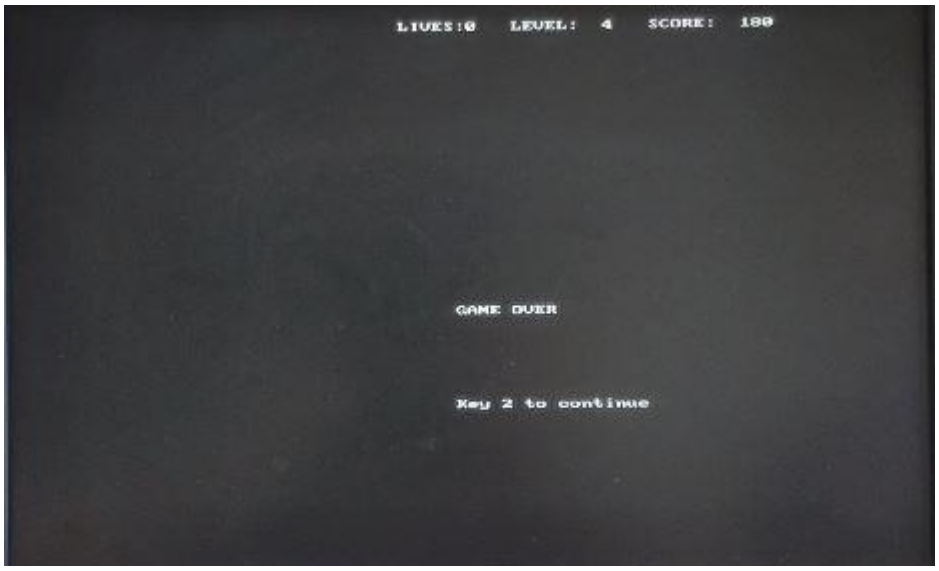


Figure 3: Game Over Screen after losing all three lives

The game is controlled using the 4 keys on the DE0-CV. Key 3 and 1 move the player left and right, key 0 fires the player projectile and key 2 resets the game after a Game Over. The board's interval timer is used to clock the game, interrupting every 200ms. After every interrupt the screen is cleared using the clearScreen function and the position of the enemies is updated and redrawn using the drawSquare function used in lab 9. The same function is also used to animate the player and projectiles. The score, current level and remaining lives are also erased and rewritten each interrupt at the top of the screen using the VGA_text function from lab 9.

II. Steps in creating the game

· Key steps in creating the game were as follows:

1. Animation of the enemies was the first task handled. We had to find a way to implement the drawSquare function from the previous lab in a way that could handle 20 squares rather than just 3. To do this we used a for loop with 20 indices to cycle through them all. The enemies start moving left a set x increment determined by the level. Once the left most enemy reaches the edge of the screen the enemies increment down 10 pixels and the x increment is inverted.

   To handle the enemy animation we implemented an array of 20 structs with each struct representing an enemy. The variables stored in each struct were the x1, x2, y1 and y2 positions of each enemy, the color to be written at those coordinates and an active/inactive status. When an enemy is hit with a projectile the enemy color is changed to black and the status set to an inactive 0. On the next interrupt the enemy would be redrawn as black so it would appear it wasn't there. Since the status is set to 0, projectiles would pass through this "dead" enemy.

2. Next we worked on the player's controls. Every interrupt, a pointer referencing the address of the keys is stored into an integer variable. That value is then used to decide if the player is moving left, right or firing. For example, if the player is pressing key 3 to move left, the key value will equal 8 which will tell the program to redraw the play 5 pixels to the left next interrupt. If the player presses key 0 a projectile will be produced in the middle of the player. Every interrupt the projectile will move 10 pixels up. Another projectile cannot be fired until this projectile has left the screen.

3. The projectiles needed hit detection otherwise they would continue upwards without interacting with the enemies. On each interrupt the program checks the position of the projectile. If the x positions were within the bounds of any of the enemy's x values and the y positions of the projectile were within the bounds of any of the enemy's y values the

projectile hit its mark. The enemy color is then changed to black and the hit status to 0. The bullet is then transported off screen so that another projectile can be immediately fired.

4. Next the enemies projectiles were implemented. Using the rand function a random, active enemy is selected. A projectile is then produced in the middle of that enemy's position. Every interrupt the projectile is redrawn down 10 pixels. The same hit detection was used to determine if the positions of the player and enemy projectile coincided. If an enemy projectile does hit, the player's lives are decremented 1. When an enemy shoots a projectile is determined by a set counter. Every level this counter fires a projectile more quickly.

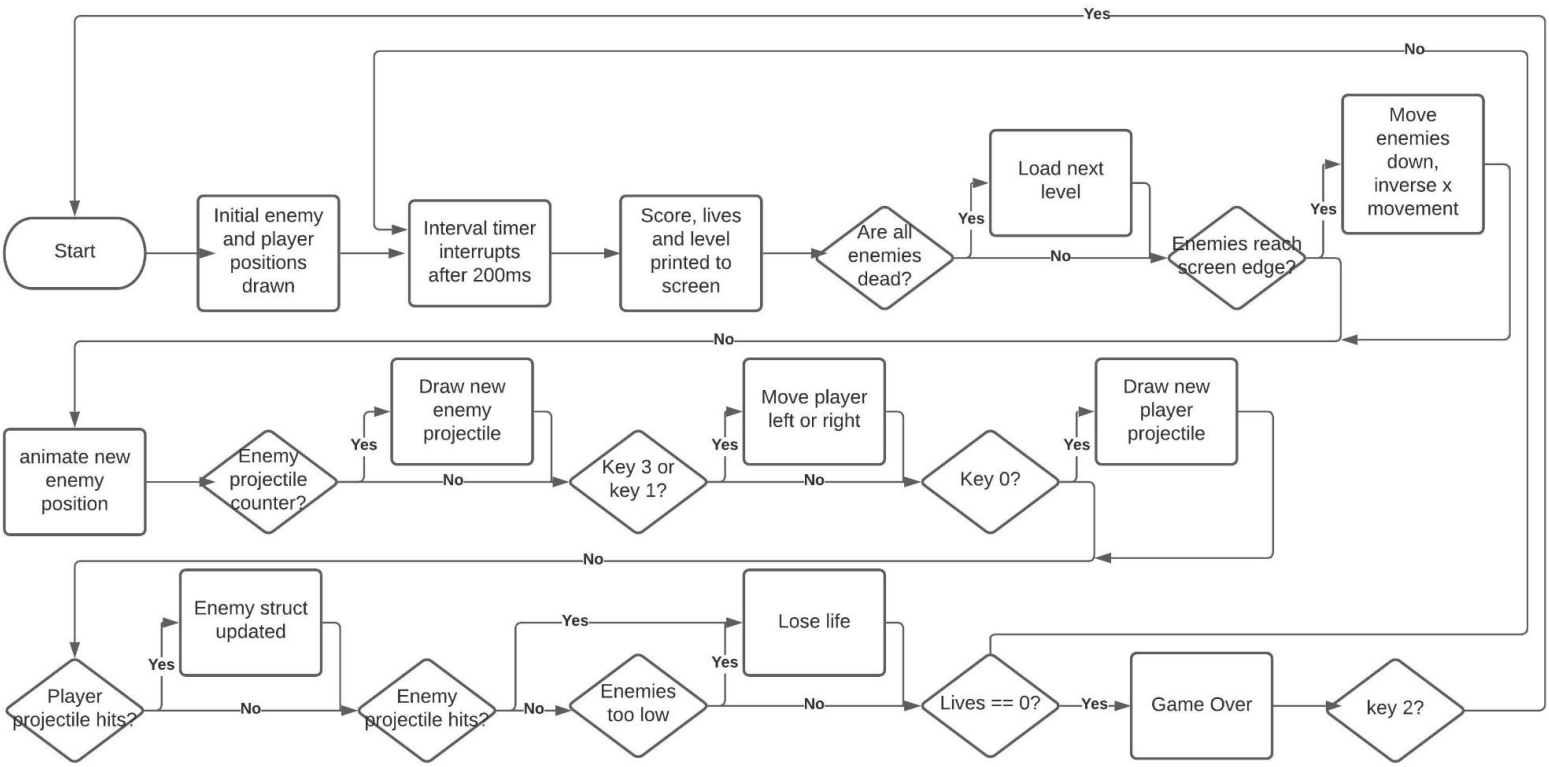
5. The final task was printing the score, level and lives to the top of the screen. Every interrupt these values would be converted from integer values to strings using the itoa function. The VGA_text function would then print labels for each value ("LEVEL:" "SCORE:", "LIVES:") followed by the values themselves. When a player is hit three times a game over message is printed to screen, prompting the player to press key 2 to reset the game. If the player presses key 2 the level is reset to 1, the lives reset to 3 and the score reset to 0.

III. Task Distribution among team members

Most of the project was worked on together on a single computer and work done as a team. A further break down of this work is as follows:

- Jesse: Bullet collision, Arrays of structs, Player Control
- Brennan: Enemy Animation, Scores & Levels, Player animation
- Albert: Bullet animation, Game Over Screen, Level Difficulty

IV. Software Flowchart



V. Difficulties encountered and Solutions

One challenge was keeping track of all twenty enemies. Having to have twenty sets of coordinates and colors would have gotten confusing fast. The solution was to compile this data as an array of structs.

Another issue was taking the integer values for score, lives and level and finding a way to use these with the VGA_text function which accepts strings. The solution was the itoa() function. The function inputs are an integer value, a string and the base system you want the string to be represented in (in this case 10).

At higher levels the bullet would also disappear before it reached the bottom of the screen. This is because another enemy would fire before the previous bullet had time to make contact with the player. To solve this we added an if statement that checked if the enemy bullet was not below the lower bound of the screen at 240 it would not spawn another bullet.

VI. New Acquired skills/knowledge

This project helped reinforce the many programming topics we have learned over the semester, especially concerning microcontrollers. We had to understand interrupts in order to use the board's various devices such as keys and interval timers. While we all had varying degrees of experience programming software, adding in hardware components created new challenges.

This project also helped solidify our understanding of the C programming language and its intricacy. Spanning multiple C and header files, this project was by far the largest assignment we have done in this class and required an understanding of all the variable types and coding techniques discussed.

A new technique we used in this project was arrays of structs which helped us keep track of all the enemy data. We also expanded our knowledge of animation using C, with this project featuring many different objects on screen at once, moving in different directions.

VII. Code with comments

```
#include "nios2_ctrl_reg_macros.h"
#include "address_map.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

volatile int timeout;			//global timeout variable for synchronization
int inc_x;				//set it to 2 to move the enemies left two pixels at a time
volatile int pixel_buffer_start = FPGA_ONCHIP_BASE;
int resolution_x = 320;		//max x
int resolution_y = 240; //max y

//Prototypes
void clearScreen();
void drawSquare(int, int, int, int, short);
void VGA_text(int, int, char*);

typedef struct //Struct defining enemy attributes
{
	int x1;
	int x2;
	int y1;
	int y2;
	short color;
	int hit;
}ENEMY;

int main()
{
	volatile int* interval_timer_ptr = (int*)TIMER_BASE;
	volatile int* key_ptr = (int*)KEY_BASE;
	timeout = 0;
```

```
srand(time(NULL));

ENEMY foes[20];
int j, i = 0;
int enemyx1 = 70, enemyx2 = 89, enemyy1 =  0, enemyy2 = 19;
int inc_x   = -2;
int playx1 = 100, playx2 = 119, playy1 = 221, playy2 = 240;
int bulletx1, bulletx2, bullety1 = 0, bullety2 = 0;
int enemyBulletx1, enemyBulletx2, enemyBullety1 = 250, enemyBullety2;
int bulletActive = 0;
short red = 0xFA00, yellow = 0xFF00, white = 0xFFFF, black = 0x0000;
int key_val = 0;
int refresh = 0;
int scoreInt = 0;
char scoreStr[5];
char livesStr[1];
char levelStr[2];
char text_erase[10] = "        \0";
int level = 1;
int lives = 3;
double count = 50;
int countIncrement = 0;
int random;
int gameOver = 0;
int wait = 0;
double temp;

int counter = 0x930000;                    //approx 200ms
*(interval_timer_ptr + 2) = (counter & 0xFFFF);
*(interval_timer_ptr + 3) = (counter >> 16) & 0xFFFF;
*(interval_timer_ptr + 1) = 0x7;

NIOS2_WRITE_IENABLE(0x1);
NIOS2_WRITE_STATUS(1);

clearScreen();
//Draw enemies initial position
for (int row = 0; row < 4; row++)
{
for (int col = 0; col < 5; col++)
{
            foes[i].x1          = enemyx1;
            foes[i].x2          = enemyx2;
            foes[i].y1          = enemyy1;
            foes[i].y2          = enemyy2;
            foes[i].color = red;
            foes[i].hit   = 1;

            drawSquare(enemyx1, enemyx2, enemyy1, enemyy2, red);
            enemyx1 += 40;
            enemyx2 += 40;
            i++;
}
enemyx1 -= 200;
enemyx2 -= 200;
enemyy1 += 40;
enemyy2 += 40;
}
enemyy1 -= 160;
```

```
enemyy2 -= 160;
drawSquare(playx1, playx2, playy1, playy2, white);  //Draw Player

while (1)
{
while (!timeout);
clearScreen();
//Print Score and Level
itoa(scoreInt, scoreStr, 10);
VGA_text(315, 0, text_erase);
VGA_text(312, 0, "SCORE: ");
VGA_text(320, 0, "       ");
VGA_text(320, 0, scoreStr);
itoa(level, levelStr, 10);
VGA_text(300, 0, "LEVEL: ");
VGA_text(308, 0, levelStr);
itoa(lives, livesStr, 10);
VGA_text(290, 0, "LIVES: ");
VGA_text(296, 0, livesStr);
i = 0;

//If all enemies dead, load next level
if (refresh == 1)
{
		enemyx1 = 70;
		enemyx2 = 89;
		enemyy1 = 0;
		enemyy2 = 19;
		for (int row = 0; row < 4; row++)
		{
		for (int col = 0; col < 5; col++)
		{
		foes[i].x1 = enemyx1;
		foes[i].x2 = enemyx2;
		foes[i].y1 = enemyy1;
		foes[i].y2 = enemyy2;
		foes[i].color = red;
		foes[i].hit = 1;

		drawSquare(enemyx1, enemyx2, enemyy1, enemyy2, red);
		enemyx1 += 40;
		enemyx2 += 40;
		i++;
		}
		enemyx1 -= 200;
		enemyx2 -= 200;
		enemyy1 += 40;
		enemyy2 += 40;
		}
		enemyy1 -= 160;
		enemyy2 -= 160;
		i = 0;
		if (gameOver != 1) //After every new level, update these varaibles:
		{
		level++;
		inc_x = -1*((abs(inc_x) + 1));
		count = 50 - (level * 7);
		if (count < 0)
		count = 1;
```

```
            countIncrement = 0;
            }
}
gameOver = 0;

//Check if enemies reach edge of screen
if ((foes[0].x1 <= 0) && (wait == 0))
{
            inc_x = -inc_x;

            enemyy1 += 10;
            enemyy2 += 10;
            wait++;
}
else if ((foes[4].x2 >= 320) && (wait == 0))
{
            inc_x = -inc_x;
            enemyy1 += 10;
            enemyy2 += 10;
            wait++;
}
if (wait != 0)   //Wait 5 interrupts till edge is checked again
{
            wait++;
            if (wait == 5)
            wait = 0;
}

//Animate next enemy position
for (int row = 0; row < 4; row++)
{
            for (int col = 0; col < 5; col++)
            {
            foes[i].x1 = enemyx1;
            foes[i].x2 = enemyx2;
            foes[i].y1 = enemyy1;
            foes[i].y2 = enemyy2;

            drawSquare(enemyx1, enemyx2, enemyy1, enemyy2, foes[i].color);
            enemyx1 += 40;
            enemyx2 += 40;
            i++;
            }
            enemyx1 -= 200;
            enemyx2 -= 200;
            enemyy1 += 40;
            enemyy2 += 40;
}
enemyy1 -= 160;
enemyy2 -= 160;
enemyx1 += inc_x;
enemyx2 += inc_x;

//Initialize Enemy Bullet after set amount of timer interrupts
while (countIncrement == count)
{
            random = (int)rand() % 20;
            if ((foes[random].hit == 1) && (enemyBullety1 >= 240))
            {
```

```
                enemyBulletx1 = foes[random].x1 + 10;
                enemyBulletx2 = enemyBulletx1 + 2;
                enemyBullety1 = foes[random].y2;
                enemyBullety2 = enemyBullety1 + 3;
                drawSquare(enemyBulletx1, enemyBulletx2, enemyBullety1, enemyBullety2, yellow);
                countIncrement = 0;
                }
                else if (enemyBullety1 < 240)
                countIncrement = 0;
        }
        countIncrement++;

        //Check if player bullet is on screen
        if (bullety2 > 0)
        {
                bullety1 -= 10;
                bullety2 -= 10;
                drawSquare(bulletx1, bulletx2, bullety1, bullety2, white);      //Draw Bullet
        }
        //Check if enemy bullet is on screen
        if (enemyBullety1 < 240)
        {
                enemyBullety1 += 10;
                enemyBullety2 += 10;
                drawSquare(enemyBulletx1, enemyBulletx2, enemyBullety1, enemyBullety2, yellow);
        }

        key_val = *(key_ptr);
        //Check value of keys
        if ((key_val == 8 || key_val == 9) && (playx1 > 0))
        {
                playx1 -= 5;
                playx2 -= 5;
        }
        if ((key_val == 2 || key_val == 3) && (playx2 < 320))
        {
                playx1 += 5;
                playx2 += 5;
        }
        if (((key_val == 1) && (bullety2 <= 0))|| ((key_val == 3) && (bullety2 <= 0)) || ((key_val == 9) && (bullety2 <=
0)))
        {
                bulletx1 = playx1 + 10;
                bulletx2 = bulletx1 + 2;
                bullety1 = 221;
                bullety2 = bullety1 + 3;
                drawSquare(bulletx1, bulletx2, bullety1, bullety2, white);      //Draw Bullet

        }

        drawSquare(playx1, playx2, playy1, playy2, white); //Draw Player

        //Check Player Bullet Collision
        for (i = 0; i < 20; i++)
        {
                if ((bulletx1 >= foes[i].x1) & (bulletx2 <= foes[i].x2) & (bullety1 >= foes[i].y1) & (bullety2 <=
foes[i].y2) & (foes[i].hit == 1))
                {
                foes[i].color = black;
```

```
                foes[i].hit = 0;
                scoreInt += level;
                bullety1 = -10;
                bullety2 = -10;
                }
        }
        //Check Enemy Bullet Collision
        if ((enemyBulletx1 >= playx1) & (enemyBulletx2 <= playx2) & (enemyBullety1 >= playy1) & (enemyBullety2
<= playy2))
        {
                lives--;
                enemyBullety1 = 300;
                enemyBullety2 = 300;
        }

        //Check if enemies are all dead to reload next level
        refresh = 1;
        for (i = 0; i < 20; i++)
        {
                if (foes[i].hit == 1)
                refresh = 0;
        }
        i = 0;

        //Check if enemies get too low (lose life)
        for (j = 0; j < 20; j++)
        {
                if ((foes[j].y2 >= 210) && (foes[j].hit == 1))
                {
                enemyx1 = 70;
                enemyx2 = 89;
                enemyy1 = 0;
                enemyy2 = 19;
                for (int row = 0; row < 4; row++)
                {
                for (int col = 0; col < 5; col++)
                {
                        foes[i].x1 = enemyx1;
                        foes[i].x2 = enemyx2;
                        foes[i].y1 = enemyy1;
                        foes[i].y2 = enemyy2;

                        drawSquare(enemyx1, enemyx2, enemyy1, enemyy2, foes[i].color);
                        enemyx1 += 40;
                        enemyx2 += 40;
                        i++;
                }
                enemyx1 -= 200;
                enemyx2 -= 200;
                enemyy1 += 40;
                enemyy2 += 40;
                }
                enemyy1 -= 160;
                enemyy2 -= 160;
                lives--;
                }
        }

        //If lives == 0 GAME OVER
```

```
		if (lives == 0)
		{
			clearScreen();
			itoa(scoreInt, scoreStr, 10);
			itoa(level, levelStr, 10);
			itoa(lives, livesStr, 10);

			while (key_val != 4)
			{
				gameOver = 1;
				VGA_text(312, 0, "SCORE: ");
				VGA_text(320, 0, scoreStr);
				VGA_text(300, 0, "LEVEL: ");
				VGA_text(308, 0, levelStr);
				VGA_text(290, 0, "LIVES: ");
				VGA_text(296, 0, livesStr);
				VGA_text(295, 30, "GAME OVER");
				VGA_text(295, 40, "Key 2 to continue");
				key_val = *(key_ptr);
			}
			scoreInt = 0;
			lives	= 3;
			level	= 1;
			refresh  = 1;
			count = 50;
			inc_x = -2;
			VGA_text(295, 30, "         ");
			VGA_text(295, 40, "                  ");
		}
		timeout = 0;  //Reset timeout for next interval interrupt
		}
		return 0;
}

//Function Definitions
void clearScreen()
{
		int clearColor = 0;
		int pixel_ptr, row, col;

		for (row = 0; row < resolution_y; row++)
		{
			for (col = 0; col < resolution_x; col++)
			{
				pixel_ptr = pixel_buffer_start + (row << 10) + (col << 1);
				*(short*)pixel_ptr = clearColor;
			}
		}
}
void drawSquare(int x1, int x2, int y1, int y2, short color)
{
		int offset;
		int row, col; //use for the for loops
		volatile short* pixel_buffer = (short*)0x08000000;

		for (row = y1; row <= y2; row++) //for each row
		{
			for (col = x1; col <= x2; col++) //go through each column
			{
```

```
                        //offset y must be shifted left 10 times and x shifted once
                        offset = (row << 9) + (col);
                        *(pixel_buffer + offset) = color;
                }
        }
}
void VGA_text(int x, int y, char* text_ptr)
{
        int offset;
        volatile char* character_buffer = (char*)FPGA_CHAR_BASE;

        offset = (y << 7) + x;

        while (*(text_ptr))
        {
                *(character_buffer + offset) = *(text_ptr);
                ++text_ptr;
                ++offset;
        }
}
```