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

#include "bunkers.h"

#define BUNKER_HEIGHT 18        // Bunkers are 18 pixels high
#define BUNKER_DAMAGE_HEIGHT 6  // Each bunnker square is size 6
#define BUNKER_ROW 175          // All bunkers are at row 175
#define BUNKER_SIZE 10          // All bunkers have 10 sections
#define BUNKER_0 0              // Gotta have
#define BUNKER_1 1              // constants to
#define BUNKER_2 2              // represent
#define BUNKER_3 3              // each bunker
#define BUNKER_DAMAGE_0 0       // Gotta have
#define BUNKER_DAMAGE_1 1       // different
#define BUNKER_DAMAGE_2 2       // damage
#define BUNKER_DAMAGE_3 3       // values
#define BUNKER_DEAD     4       // Damage bunker has when it is dead
#define BUNKER_ROWS     18      // How many rows each bunker has
#define BUNKER_COLS     24      // How many columns each bunker has

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

#define DAMAGE_WORD_WIDTH 6
#define WORD_WIDTH 24
#define NUM_OF_BUNKERS 4
#define LOC_BUNKER_ONE 60   // Divided this by 2 because screen is half

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

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

// Necessary for storing the bunker data
#define packword24(b23,b22,b21,b20,b19,b18,b17,b16,b15,b14,b13,b12,b11,b10,b9,b8,b7,b6,b5,b4,b3,b2,b1,b0) \
        ((b23 << 23) | (b22 << 22) | (b21 << 21) | (b20 << 20) | (b19 << 19) | (b18 << 18) | (b17 << 17) | (b16 << 16) |                          \
                (b15 << 15) | (b14 << 14) | (b13 << 13) | (b12 << 12) | (b11 << 11) | (b10 << 10) | (b9  << 9 ) | (b8  << 8 ) |                          \
                (b7  << 7 ) | (b6  << 6 ) | (b5  << 5 ) | (b4  << 4 ) | (b3  << 3 ) | (b2  << 2 ) | (b1  << 1 ) | (b0  << 0 ) )
```

```c
// Shape of the entire bunker.
static const int32_t bunker_24x18[BUNKER_HEIGHT] = {
        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)};

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

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

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

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

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

struct bunker{      // Holds the data for each bunker
    int32_t row;        // Where it is
    int32_t col;        // on the screen
    int32_t damage;     // What damage level the bunker is at
    int32_t pixel[];    // A bunker is made out of squares- whether it's alive/dead
}bunker[3];


// These arrays show how decayed each part of the bunker is.
int32_t bunker_zero[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t bunker_one[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t bunker_two[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int32_t bunker_three[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

```

```c
// ----------------------------------------------------
// Declaration for internal functions
void bunkers_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t color);
void bunker0(int32_t r, uint32_t * framePointer);
void bunker1(int32_t r, uint32_t * framePointer);
void bunker2(int32_t r, uint32_t * framePointer);
void bunker3(int32_t r, uint32_t * framePointer);
void degrigation_patern(int32_t row, int32_t col, int32_t bunker_number, int32_t damage, uint32_t * framePointer);
void bunker_hit(uint32_t * framePointer, int32_t location, int32_t bunker_num);
// End internal function declaration
// ----------------------------------------------------

/*
 * Draws a pixel on the screen. To compensate for our double-resolution screen,
 * it must draw 4 real pixels for every in-came pixel.
 */
void bunkers_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t color){
#define DRAW_PIXEL_ROW_MULTIPLIER 1280  // 640 * 2 for screen doubling
#define DRAW_PIXEL_ROW 640              // one row offset
#define DRAW_PIXEL_DOUBLE 2             // for doubling

    // We draw 4 pixels for every 1 small-screen pixel
    framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE] = color;
    framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE+1] = color;
    framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE] = color;
    framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE + 1] = color;
}

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


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

// This randomly selects a bunker and randomly destroys part of it
void bunkers_hit_rand_bunker(uint32_t * framePointer){
    int32_t r = rand()%NUM_OF_BUNKERS;  // Randomly pick a bunker

    switch (r){
    case BUNKER_0:              // Depending on the bunker number, destroy one in that.
        bunker0(rand()%BUNKER_SIZE, framePointer);
        break;
    case BUNKER_1:              // bunker 1
        bunker1(rand()%BUNKER_SIZE, framePointer);
        break;
    case BUNKER_2:              // bunker 2
        bunker2(rand()%BUNKER_SIZE, framePointer);
        break;
    case BUNKER_3:              // bunker 3
        bunker3(rand()%BUNKER_SIZE, framePointer);
        break;
    }
}


#define HIT_ROW_LOC_0 0     // For
#define HIT_ROW_LOC_1 0     // every
#define HIT_ROW_LOC_2 0     // location
#define HIT_ROW_LOC_3 0     // it has
#define HIT_ROW_LOC_4 6     // a specific
#define HIT_ROW_LOC_5 6     // row
#define HIT_ROW_LOC_6 6     // associated
#define HIT_ROW_LOC_7 6     // with
#define HIT_ROW_LOC_8 12    // each
#define HIT_ROW_LOC_9 12    // location

#define HIT_COL_LOC_0 0     // for
#define HIT_COL_LOC_1 6     // every
#define HIT_COL_LOC_2 12    // location
#define HIT_COL_LOC_3 18    // it has
#define HIT_COL_LOC_4 0     // a specific
#define HIT_COL_LOC_5 6     // column
#define HIT_COL_LOC_6 12    // associated
#define HIT_COL_LOC_7 18    // with
#define HIT_COL_LOC_8 0     // each
#define HIT_COL_LOC_9 18    // location

#define HIT_0 0             // There
#define HIT_1 1             // isn't
#define HIT_2 2             // an easy
#define HIT_3 3             // way to
#define HIT_4 4             // loop through
#define HIT_5 5             // all these
#define HIT_6 6             // yet,
#define HIT_7 7             // so we have
#define HIT_8 8             // every location
#define HIT_9 9             // hard-coded in.

// Put a hit on bunker 0 at a certain location
```

```
void bunker0(int32_t r, uint32_t * framePointer){
    if(bunker_zero[r]== BUNKER_DEAD){   // If our bunker is already dead here
        bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
        return;
    }
    switch(r){                          // Based on where the hit is
    case HIT_0:     // Hit in location 0. Row 0 and column 0
        degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_1:     // Hit in location 1. Row 1 and column 1
        degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_2:     // Hit in location 2. Row 2 and column 2
        degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_3:     // Hit in location 3. Row 3 and column 3
        degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_4:     // Hit in location 4. Row 4 and column 4
        degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_5:     // Hit in location 5. Row 5 and column 5
        degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_6:     // Hit in location 6. Row 6 and column 6
        degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_7:     // Hit in location 7. Row 7 and column 7
        degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_8:     // Hit in location 8. Row 8 and column 8
        degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_0, bunker_zero[r], framePointer);
        break;
    case HIT_9:     // Hit in location 9. Row 9 and column 9
        degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_0, bunker_zero[r], framePointer);
        break;
    }
    bunker_zero[r]++;
}

// Put a hit on bunker 1 at a certain location
void bunker1(int32_t r, uint32_t * framePointer){
    if(bunker_one[r]== BUNKER_DEAD){    // If our bunker is already dead here
        bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
        return;
    }
    switch(r){                          // Based on where the hit is
    case HIT_0:     // Hit in location 0. Row 0 and column 0
```

```
        degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_1:     // Hit in location 1. Row 1 and column 1
        degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_2:     // Hit in location 2. Row 2 and column 2
        degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_3:     // Hit in location 3. Row 3 and column 3
        degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_4:     // Hit in location 4. Row 4 and column 4
        degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_5:     // Hit in location 5. Row 5 and column 5
        degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_6:     // Hit in location 6. Row 6 and column 6
        degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_7:     // Hit in location 7. Row 7 and column 7
        degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_8:     // Hit in location 8. Row 8 and column 8
        degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_1, bunker_one[r], framePointer);
        break;
    case HIT_9:     // Hit in location 9. Row 9 and column 9
        degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_1, bunker_one[r], framePointer);
        break;
    }
    bunker_one[r]++;
}

// Put a hit on bunker 2 at a certain location
void bunker2(int32_t r, uint32_t * framePointer){
    if(bunker_two[r]== BUNKER_DEAD){    // If our bunker is already dead here
        bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
        return;
    }
    switch(r){                          // Based on where the hit is
    case HIT_0:     // Hit in location 0. Row 0 and column 0
        degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_1:     // Hit in location 1. Row 1 and column 1
        degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_2:     // Hit in location 2. Row 2 and column 2
        degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_3:     // Hit in location 3. Row 3 and column 3
        degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_4:     // Hit in location 4. Row 4 and column 4
        degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_5:     // Hit in location 5. Row 5 and column 5
        degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_6:     // Hit in location 6. Row 6 and column 6
        degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_7:     // Hit in location 7. Row 7 and column 7
        degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_8:     // Hit in location 8. Row 8 and column 8
        degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_2, bunker_two[r], framePointer);
        break;
    case HIT_9:     // Hit in location 9. Row 9 and column 9
        degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_2, bunker_two[r], framePointer);
        break;
    }
    bunker_two[r]++;
}

// Put a hit on bunker 3 at a certain location
void bunker3(int32_t r, uint32_t * framePointer){
    if(bunker_three[r]== BUNKER_DEAD){  // If our bunker is already dead here
        bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
        return;
    }
    switch(r){                          // Based on where the hit is
    case HIT_0:     // Hit in location 0. Row 0 and column 0
        degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_3, bunker_three[r], framePointer);
        break;
    case HIT_1:     // Hit in location 1. Row 1 and column 1
        degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_3, bunker_three[r], framePointer);
        break;
    case HIT_2:     // Hit in location 2. Row 2 and column 2
        degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_3, bunker_three[r], framePointer);
        break;
    case HIT_3:     // Hit in location 3. Row 3 and column 3
        degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_3, bunker_three[r], framePointer);
```

```c
        break;
    case HIT_4:     // Hit in location 4. Row 4 and column 4
        degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_3, bunker_three[r], framePointer);
        break;
    case HIT_5:     // Hit in location 5. Row 5 and column 5
        degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_3, bunker_three[r], framePointer);
        break;
    case HIT_6:     // Hit in location 6. Row 6 and column 6
        degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_3, bunker_three[r], framePointer);
        break;
    case HIT_7:     // Hit in location 7. Row 7 and column 7
        degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_3, bunker_three[r], framePointer);
        break;
    case HIT_8:     // Hit in location 8. Row 8 and column 8
        degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_3, bunker_three[r], framePointer);
        break;
    case HIT_9:     // Hit in location 9. Row 9 and column 9
        degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_3, bunker_three[r], framePointer);
        break;
    }
    bunker_three[r]++;
}

// This goes through all the bunkers and destroys them according to our pattern
void degrigation_patern(int32_t row, int32_t col, int32_t bunker_num, int32_t damage, uint32_t * framePointer){
    int32_t r,c;
    for(r=0;r<BUNKER_DAMAGE_HEIGHT;r++){       // Go through rows
        for(c=0;c<DAMAGE_WORD_WIDTH;c++){       // and columns
            if (damage == BUNKER_DAMAGE_0       // 0 damage level
                    && (bunkerDamage0_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
                // If we need to erase a pixel here, do so.
                bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
                        ,c+col+bunker[bunker_num].col, BLACK);

            }else if(damage == BUNKER_DAMAGE_1  // 1 damage level
                    && (bunkerDamage1_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
                // If we need to erase a pixel here, do so.
                bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
                        ,c+col+bunker[bunker_num].col, BLACK);

            }else if(damage == BUNKER_DAMAGE_2  // 2 damage level
                    && (bunkerDamage2_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
                // If we need to erase a pixel here, do so.
                bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
                        ,c+col+bunker[bunker_num].col, BLACK);

            }else if(damage == BUNKER_DAMAGE_3  // 3 damage level
                    && (bunkerDamage3_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
                // If we need to erase a pixel here, do so.
                bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
                        ,c+col+bunker[bunker_num].col, BLACK);
            }
        }
    }
}


```