```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 ) )
```

```
53 // Shape of the entire bunker.
54 static const int32_t bunker_24x18[BUNKER_HEIGHT] = {
55         packword24(0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0),
56         packword24(0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0),
57         packword24(0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0),
58         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
59         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
60         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
61         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
62         packword24(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
63         packword24(1,1,1,1,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,1,1,1),
64         packword24(1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,1,1),
65         packword24(1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1),
66         packword24(1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1),
67         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
68         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
69         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
70         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
71         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1),
72         packword24(1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1)};
73
74 // First time a bunker is hit, the first damage that happens
75 static const int32_t bunkerDamage0_6x6[BUNKER_DAMAGE_HEIGHT] = {
76         packword6(0,1,1,0,0,0), packword6(0,0,0,0,0,1), packword6(1,1,0,1,0,0),
77         packword6(1,0,0,0,0,0), packword6(0,0,1,1,0,0), packword6(0,0,0,0,1,0)};
78
79 // Second time a bunker is hit, this is its damage
80 static const int32_t bunkerDamage1_6x6[BUNKER_DAMAGE_HEIGHT] = {
81         packword6(1,1,1,0,1,0), packword6(1,0,1,0,0,1), packword6(1,1,0,1,1,1),
82         packword6(1,0,0,0,0,0), packword6(0,1,1,1,0,1), packword6(0,1,1,0,1,0)};
83
84 // Third time a bunker is hit, this is its damage
85 static const int32_t bunkerDamage2_6x6[BUNKER_DAMAGE_HEIGHT] = {
86         packword6(1,1,1,1,1,1), packword6(1,0,1,1,0,1), packword6(1,1,0,1,1,1),
87         packword6(1,1,0,1,1,0), packword6(0,1,1,1,0,1), packword6(1,1,1,1,1,1)};
88
89 // Fourth time a bunker is hit, this is its damage
90 static const int32_t bunkerDamage3_6x6[BUNKER_DAMAGE_HEIGHT] = {
91         packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1),
92         packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1), packword6(1,1,1,1,1,1)};
93
94 //  End hardcoded static const stuff
95 // ------------------------------------------------------
96
97 struct bunker{      // Holds the data for each bunker
98     int32_t row;        // Where it is
99     int32_t col;        // on the screen
100    int32_t damage;     // What damage level the bunker is at
101    int32_t pixel[];    // A bunker is made out of squares- whether it's alive/dead
102 }bunker[3];
103
104
105 // These arrays show how decayed each part of the bunker is.
106 int32_t bunker_zero[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
107 int32_t bunker_one[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
108 int32_t bunker_two[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
109 int32_t bunker_three[BUNKER_SIZE] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
110
```

```
111 // -----------------------------------------------------
112 // Declaration for internal functions
113 void bunkers_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t
    color);
114 void bunker0(int32_t r, uint32_t * framePointer);
115 void bunker1(int32_t r, uint32_t * framePointer);
116 void bunker2(int32_t r, uint32_t * framePointer);
117 void bunker3(int32_t r, uint32_t * framePointer);
118 void degrigation_patern(int32_t row, int32_t col, int32_t bunker_number, int32_t
    damage, uint32_t * framePointer);
119 void bunker_hit(uint32_t * framePointer, int32_t location, int32_t bunker_num);
120 // End internal function declaration
121 // -----------------------------------------------------
122
123 /*
124  * Draws a pixel on the screen. To compensate for our double-resolution screen,
125  * it must draw 4 real pixels for every in-came pixel.
126  */
127 void bunkers_draw_pixel(uint32_t *framePointer,uint32_t row,uint32_t col,uint32_t
    color){
128 #define DRAW_PIXEL_ROW_MULTIPLIER 1280  // 640 * 2 for screen doubling
129 #define DRAW_PIXEL_ROW 640              // one row offset
130 #define DRAW_PIXEL_DOUBLE 2             // for doubling
131
132     // We draw 4 pixels for every 1 small-screen pixel
133     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE] = color;
134     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER + col*DRAW_PIXEL_DOUBLE+1] = color;
135     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE]
    = color;
136     framePointer[row*DRAW_PIXEL_ROW_MULTIPLIER+DRAW_PIXEL_ROW+ col*DRAW_PIXEL_DOUBLE +
    1] = color;
137 }
138
139 // Initializes the bunkers
140 void bunkers_init(uint32_t * framePointer){
141     int32_t i, loc = LOC_BUNKER_ONE;    //
142     for(i = 0; i < NUM_OF_BUNKERS ; i++){
143         bunker[i].row = BUNKER_ROW; // Divided by 2 because screen is half
144         bunker[i].col = loc;        // which column it is at
145         bunker[i].damage = 0;       // Start undamaged
146         loc += LOC_BUNKER_ONE;      // Add by the offset
147     }
148     bunkers_build(framePointer);    // Draw the bunkers on the screen
149 }
150
151
152 void bunkers_build(uint32_t * framePointer){
153     int32_t row, col, b;                            // Declare loop vars
154     for(row=0;row<BUNKER_ROWS;row++){                   // Go through rows
155         for(col=0;col<BUNKER_COLS;col++){               // Go through cols
156             if ((bunker_24x18[row] & (1<<(WORD_WIDTH-col-1)))) {// if pixel
157                 for(b = 0; b <NUM_OF_BUNKERS; b++){// draw that pixel every time
158                     bunkers_draw_pixel(framePointer,row+bunker[b].row,col+bunker[b].co
    l,GREEN);
159                 }
160             }
161         }
162     }
```

```
163 }
164
165 // This randomly selects a bunker and randomly destroys part of it
166 void bunkers_hit_rand_bunker(uint32_t * framePointer){
167     int32_t r = rand()%NUM_OF_BUNKERS;  // Randomly pick a bunker
168
169     switch (r){
170     case BUNKER_0:              // Depending on the bunker number, destroy one in
    that.
171         bunker0(rand()%BUNKER_SIZE, framePointer);
172         break;
173     case BUNKER_1:              // bunker 1
174         bunker1(rand()%BUNKER_SIZE, framePointer);
175         break;
176     case BUNKER_2:              // bunker 2
177         bunker2(rand()%BUNKER_SIZE, framePointer);
178         break;
179     case BUNKER_3:              // bunker 3
180         bunker3(rand()%BUNKER_SIZE, framePointer);
181         break;
182     }
183 }
184
185
186 #define HIT_ROW_LOC_0 0     // For
187 #define HIT_ROW_LOC_1 0     // every
188 #define HIT_ROW_LOC_2 0     // location
189 #define HIT_ROW_LOC_3 0     // it has
190 #define HIT_ROW_LOC_4 6     // a specific
191 #define HIT_ROW_LOC_5 6     // row
192 #define HIT_ROW_LOC_6 6     // associated
193 #define HIT_ROW_LOC_7 6     // with
194 #define HIT_ROW_LOC_8 12    // each
195 #define HIT_ROW_LOC_9 12    // location
196
197 #define HIT_COL_LOC_0 0     // for
198 #define HIT_COL_LOC_1 6     // every
199 #define HIT_COL_LOC_2 12    // location
200 #define HIT_COL_LOC_3 18    // it has
201 #define HIT_COL_LOC_4 0     // a specific
202 #define HIT_COL_LOC_5 6     // column
203 #define HIT_COL_LOC_6 12    // associated
204 #define HIT_COL_LOC_7 18    // with
205 #define HIT_COL_LOC_8 0     // each
206 #define HIT_COL_LOC_9 18    // location
207
208 #define HIT_0 0             // There
209 #define HIT_1 1             // isn't
210 #define HIT_2 2             // an easy
211 #define HIT_3 3             // way to
212 #define HIT_4 4             // loop through
213 #define HIT_5 5             // all these
214 #define HIT_6 6             // yet,
215 #define HIT_7 7             // so we have
216 #define HIT_8 8             // every location
217 #define HIT_9 9             // hard-coded in.
218
219 // Put a hit on bunker 0 at a certain location
```

```
220 void bunker0(int32_t r, uint32_t * framePointer){
221     if(bunker_zero[r]== BUNKER_DEAD){   // If our bunker is already dead here
222         bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
223         return;
224     }
225     switch(r){                          // Based on where the hit is
226     case HIT_0:     // Hit in location 0. Row 0 and column 0
227         degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_0, bunker_zero[r],
    framePointer);
228         break;
229     case HIT_1:     // Hit in location 1. Row 1 and column 1
230         degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_0, bunker_zero[r],
    framePointer);
231         break;
232     case HIT_2:     // Hit in location 2. Row 2 and column 2
233         degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_0, bunker_zero[r],
    framePointer);
234         break;
235     case HIT_3:     // Hit in location 3. Row 3 and column 3
236         degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_0, bunker_zero[r],
    framePointer);
237         break;
238     case HIT_4:     // Hit in location 4. Row 4 and column 4
239         degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_0, bunker_zero[r],
    framePointer);
240         break;
241     case HIT_5:     // Hit in location 5. Row 5 and column 5
242         degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_0, bunker_zero[r],
    framePointer);
243         break;
244     case HIT_6:     // Hit in location 6. Row 6 and column 6
245         degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_0, bunker_zero[r],
    framePointer);
246         break;
247     case HIT_7:     // Hit in location 7. Row 7 and column 7
248         degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_0, bunker_zero[r],
    framePointer);
249         break;
250     case HIT_8:     // Hit in location 8. Row 8 and column 8
251         degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_0, bunker_zero[r],
    framePointer);
252         break;
253     case HIT_9:     // Hit in location 9. Row 9 and column 9
254         degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_0, bunker_zero[r],
    framePointer);
255         break;
256     }
257     bunker_zero[r]++;
258 }
259
260 // Put a hit on bunker 1 at a certain location
261 void bunker1(int32_t r, uint32_t * framePointer){
262     if(bunker_one[r]== BUNKER_DEAD){    // If our bunker is already dead here
263         bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
264         return;
265     }
266     switch(r){                          // Based on where the hit is
267     case HIT_0:     // Hit in location 0. Row 0 and column 0
```

```
268         degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_1, bunker_one[r],
    framePointer);
269         break;
270     case HIT_1:     // Hit in location 1. Row 1 and column 1
271         degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_1, bunker_one[r],
    framePointer);
272         break;
273     case HIT_2:     // Hit in location 2. Row 2 and column 2
274         degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_1, bunker_one[r],
    framePointer);
275         break;
276     case HIT_3:     // Hit in location 3. Row 3 and column 3
277         degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_1, bunker_one[r],
    framePointer);
278         break;
279     case HIT_4:     // Hit in location 4. Row 4 and column 4
280         degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_1, bunker_one[r],
    framePointer);
281         break;
282     case HIT_5:     // Hit in location 5. Row 5 and column 5
283         degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_1, bunker_one[r],
    framePointer);
284         break;
285     case HIT_6:     // Hit in location 6. Row 6 and column 6
286         degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_1, bunker_one[r],
    framePointer);
287         break;
288     case HIT_7:     // Hit in location 7. Row 7 and column 7
289         degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_1, bunker_one[r],
    framePointer);
290         break;
291     case HIT_8:     // Hit in location 8. Row 8 and column 8
292         degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_1, bunker_one[r],
    framePointer);
293         break;
294     case HIT_9:     // Hit in location 9. Row 9 and column 9
295         degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_1, bunker_one[r],
    framePointer);
296         break;
297     }
298     bunker_one[r]++;
299 }
300
301 // Put a hit on bunker 2 at a certain location
302 void bunker2(int32_t r, uint32_t * framePointer){
303     if(bunker_two[r]== BUNKER_DEAD){    // If our bunker is already dead here
304         bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
305         return;
306     }
307     switch(r){                          // Based on where the hit is
308     case HIT_0:     // Hit in location 0. Row 0 and column 0
309         degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_2, bunker_two[r],
    framePointer);
310         break;
311     case HIT_1:     // Hit in location 1. Row 1 and column 1
312         degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_2, bunker_two[r],
    framePointer);
313         break;
```

```
314     case HIT_2:     // Hit in location 2. Row 2 and column 2
315         degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_2, bunker_two[r],
    framePointer);
316         break;
317     case HIT_3:     // Hit in location 3. Row 3 and column 3
318         degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_2, bunker_two[r],
    framePointer);
319         break;
320     case HIT_4:     // Hit in location 4. Row 4 and column 4
321         degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_2, bunker_two[r],
    framePointer);
322         break;
323     case HIT_5:     // Hit in location 5. Row 5 and column 5
324         degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_2, bunker_two[r],
    framePointer);
325         break;
326     case HIT_6:     // Hit in location 6. Row 6 and column 6
327         degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_2, bunker_two[r],
    framePointer);
328         break;
329     case HIT_7:     // Hit in location 7. Row 7 and column 7
330         degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_2, bunker_two[r],
    framePointer);
331         break;
332     case HIT_8:     // Hit in location 8. Row 8 and column 8
333         degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_2, bunker_two[r],
    framePointer);
334         break;
335     case HIT_9:     // Hit in location 9. Row 9 and column 9
336         degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_2, bunker_two[r],
    framePointer);
337         break;
338     }
339     bunker_two[r]++;
340 }
341
342 // Put a hit on bunker 3 at a certain location
343 void bunker3(int32_t r, uint32_t * framePointer){
344     if(bunker_three[r]== BUNKER_DEAD){  // If our bunker is already dead here
345         bunkers_hit_rand_bunker(framePointer);      // call rand kill bunker again
346         return;
347     }
348     switch(r){                          // Based on where the hit is
349     case HIT_0:     // Hit in location 0. Row 0 and column 0
350         degrigation_patern(HIT_ROW_LOC_0, HIT_COL_LOC_0, BUNKER_3, bunker_three[r],
    framePointer);
351         break;
352     case HIT_1:     // Hit in location 1. Row 1 and column 1
353         degrigation_patern(HIT_ROW_LOC_1, HIT_COL_LOC_1, BUNKER_3, bunker_three[r],
    framePointer);
354         break;
355     case HIT_2:     // Hit in location 2. Row 2 and column 2
356         degrigation_patern(HIT_ROW_LOC_2, HIT_COL_LOC_2, BUNKER_3, bunker_three[r],
    framePointer);
357         break;
358     case HIT_3:     // Hit in location 3. Row 3 and column 3
359         degrigation_patern(HIT_ROW_LOC_3, HIT_COL_LOC_3, BUNKER_3, bunker_three[r],
    framePointer);
```

```
360         break;
361     case HIT_4:     // Hit in location 4. Row 4 and column 4
362         degrigation_patern(HIT_ROW_LOC_4, HIT_COL_LOC_4, BUNKER_3, bunker_three[r],
    framePointer);
363         break;
364     case HIT_5:     // Hit in location 5. Row 5 and column 5
365         degrigation_patern(HIT_ROW_LOC_5, HIT_COL_LOC_5, BUNKER_3, bunker_three[r],
    framePointer);
366         break;
367     case HIT_6:     // Hit in location 6. Row 6 and column 6
368         degrigation_patern(HIT_ROW_LOC_6, HIT_COL_LOC_6, BUNKER_3, bunker_three[r],
    framePointer);
369         break;
370     case HIT_7:     // Hit in location 7. Row 7 and column 7
371         degrigation_patern(HIT_ROW_LOC_7, HIT_COL_LOC_7, BUNKER_3, bunker_three[r],
    framePointer);
372         break;
373     case HIT_8:     // Hit in location 8. Row 8 and column 8
374         degrigation_patern(HIT_ROW_LOC_8, HIT_COL_LOC_8, BUNKER_3, bunker_three[r],
    framePointer);
375         break;
376     case HIT_9:     // Hit in location 9. Row 9 and column 9
377         degrigation_patern(HIT_ROW_LOC_9, HIT_COL_LOC_9, BUNKER_3, bunker_three[r],
    framePointer);
378         break;
379     }
380     bunker_three[r]++;
381 }
382
383 // This goes through all the bunkers and destroys them according to our pattern
384 void degrigation_patern(int32_t row, int32_t col, int32_t bunker_num, int32_t damage,
    uint32_t * framePointer){
385     int32_t r,c;
386     for(r=0;r<BUNKER_DAMAGE_HEIGHT;r++){        // Go through rows
387         for(c=0;c<DAMAGE_WORD_WIDTH;c++){       // and columns
388             if (damage == BUNKER_DAMAGE_0       // 0 damage level
389                     && (bunkerDamage0_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
390                 // If we need to erase a pixel here, do so.
391                 bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
392                         ,c+col+bunker[bunker_num].col, BLACK);
393
394             }else if(damage == BUNKER_DAMAGE_1  // 1 damage level
395                     && (bunkerDamage1_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
396                 // If we need to erase a pixel here, do so.
397                 bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
398                         ,c+col+bunker[bunker_num].col, BLACK);
399
400             }else if(damage == BUNKER_DAMAGE_2  // 2 damage level
401                     && (bunkerDamage2_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
402                 // If we need to erase a pixel here, do so.
403                 bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
404                         ,c+col+bunker[bunker_num].col, BLACK);
405
406             }else if(damage == BUNKER_DAMAGE_3  // 3 damage level
407                     && (bunkerDamage3_6x6[r] & (1<<(DAMAGE_WORD_WIDTH-c-1)))){
408                 // If we need to erase a pixel here, do so.
409                 bunkers_draw_pixel(framePointer,r+row+bunker[bunker_num].row
410                         ,c+col+bunker[bunker_num].col, BLACK);
```

```
411             }
412         }
413     }
414 }
415
416
417
```