Cube Crusher

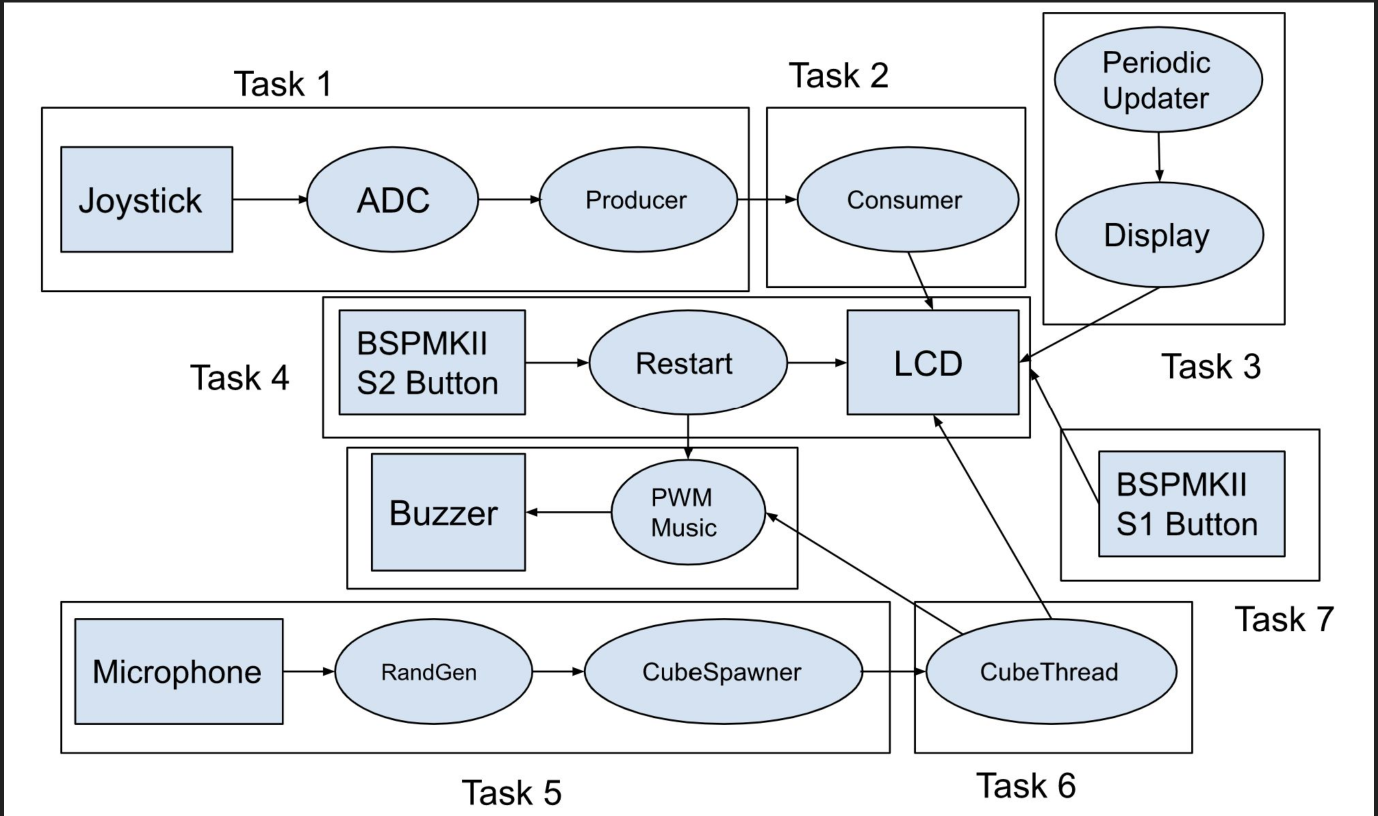
Team Alpha

John, James, Max, Chengyuan

Game Overview

- Moving cursor that crushes cubes
- Cube speed increases and duration decreases from levels 1 - 10
    - Level up occurs when your score reaches a multiple of 10
- High score displayed at end of the game

Game Design (Flowchart)



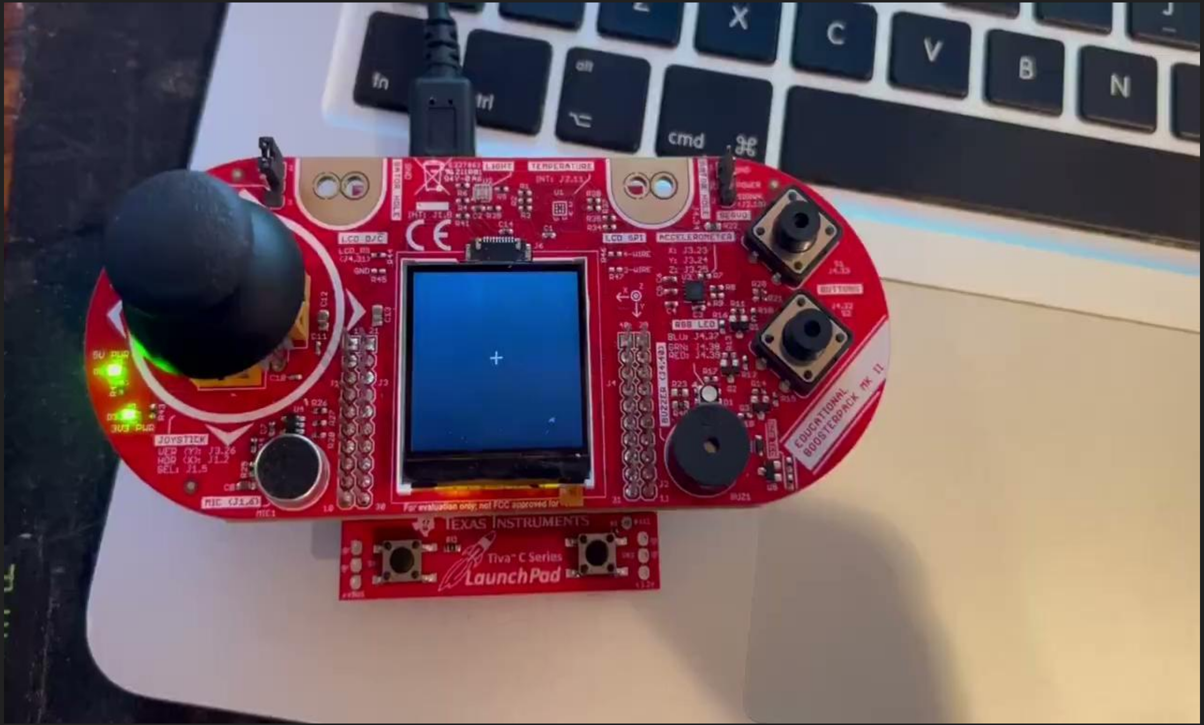
Team Responsibilities

- Chengyuan: Responsible for game scoring and display
- Max: Responsible for cube generation
- John: Responsible for random number generator and deadlock prevention
- James: Responsible for sound effects

Task 1 - Joystick Input & Crosshair Display

- Reused the implementation of mini project1
- Removed axis displays
- Decrease crosshair movement speed
- Change crosshair color

Task 1 - Demo


LIGHT
TEMPERATURE
LCD SPI
ACCELEROMETER
SERVO
BUTTONS
RGB LED
BUZZER
JOYSTICK
EDUCATIONAL BOOSTERPACK MK II
TEXAS INSTRUMENTS
Tiva C Series
LaunchPad

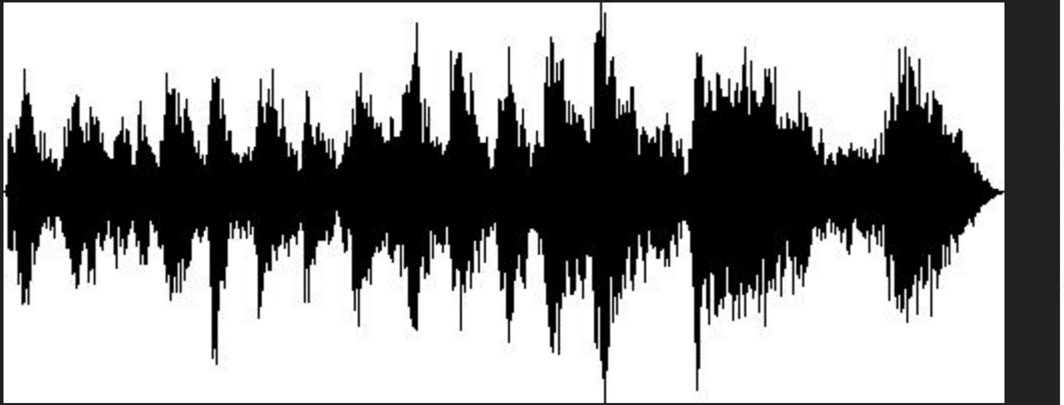
Task 2 - Game Scoring & Panel Display

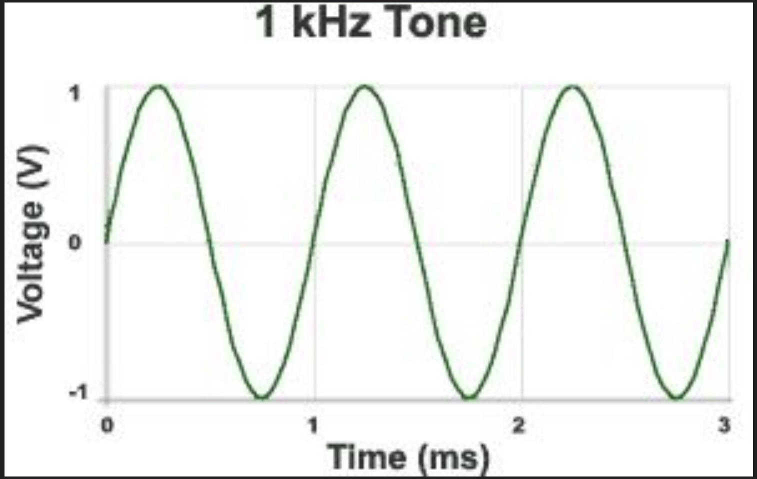
- Added Game Scoring Display
- Added Life Remaining Display
- These global variables will be decremented/incremented based on game states

Task 2 - Demo

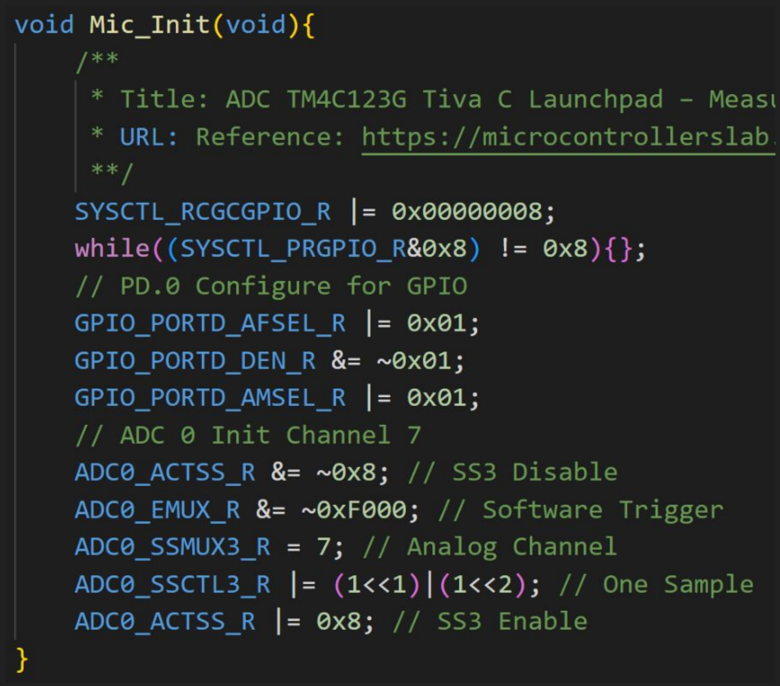
Task 3 - Random Number Generator

- Used pseudo-random number generator from C standard library
- Random seed set using sample from the microphone
  - Raw microphone readings are very noisy and inherently non-constant
  - 12-bit resolution = 4096 possible values

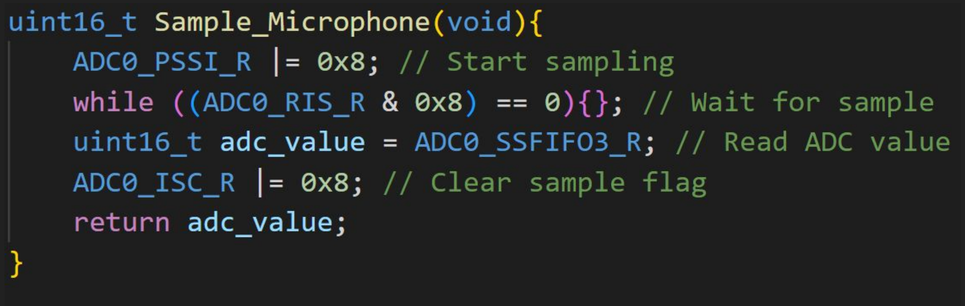




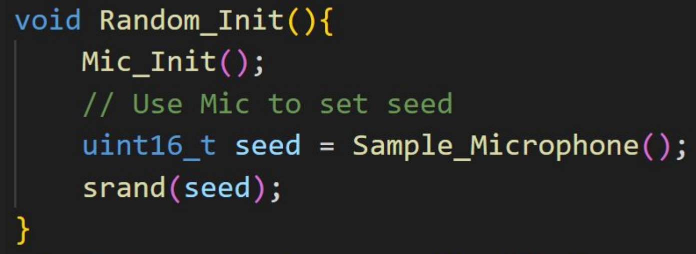
Task 3 - Random Number Generator



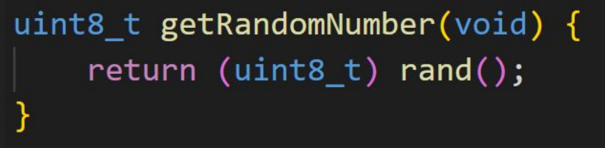
```c
void Mic_Init(void){
    /**
     * Title: ADC TM4C123G Tiva C Launchpad - Measu
     * URL: Reference: https://microcontrollerslab.
     **/
    SYSCTL_RCGCGPIO_R |= 0x00000008;
    while((SYSCTL_PRGPIO_R&0x8) != 0x8){};
    // PD.0 Configure for GPIO
    GPIO_PORTD_AFSEL_R |= 0x01;
    GPIO_PORTD_DEN_R &= ~0x01;
    GPIO_PORTD_AMSEL_R |= 0x01;
    // ADC 0 Init Channel 7
    ADC0_ACTSS_R &= ~0x8; // SS3 Disable
    ADC0_EMUX_R &= ~0xF000; // Software Trigger
    ADC0_SSMUX3_R = 7; // Analog Channel
    ADC0_SSCTL3_R |= (1<<1)|(1<<2); // One Sample
    ADC0_ACTSS_R |= 0x8; // SS3 Enable
}
```



```c
uint16_t Sample_Microphone(void){
    ADC0_PSSI_R |= 0x8; // Start sampling
    while ((ADC0_RIS_R & 0x8) == 0){}; // Wait for sample
    uint16_t adc_value = ADC0_SSFIFO3_R; // Read ADC value
    ADC0_ISC_R |= 0x8; // Clear sample flag
    return adc_value;
}
```



```c
void Random_Init(){
    Mic_Init();
    // Use Mic to set seed
    uint16_t seed = Sample_Microphone();
    srand(seed);
}
```

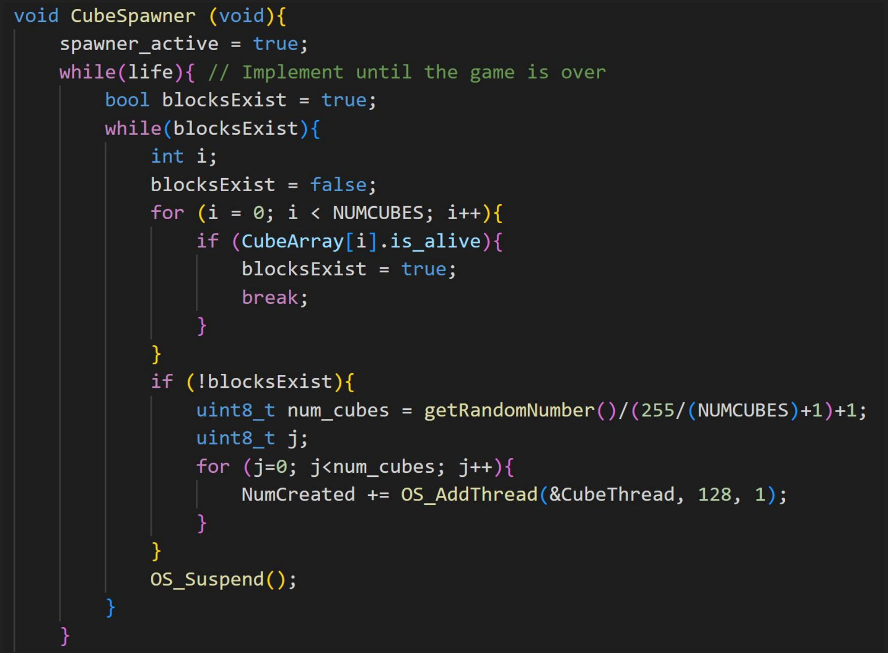


```c
uint8_t getRandomNumber(void) {
    return (uint8_t) rand();
}
```

Task 3 - Demo

Task 4 - Cube Generation & Motion

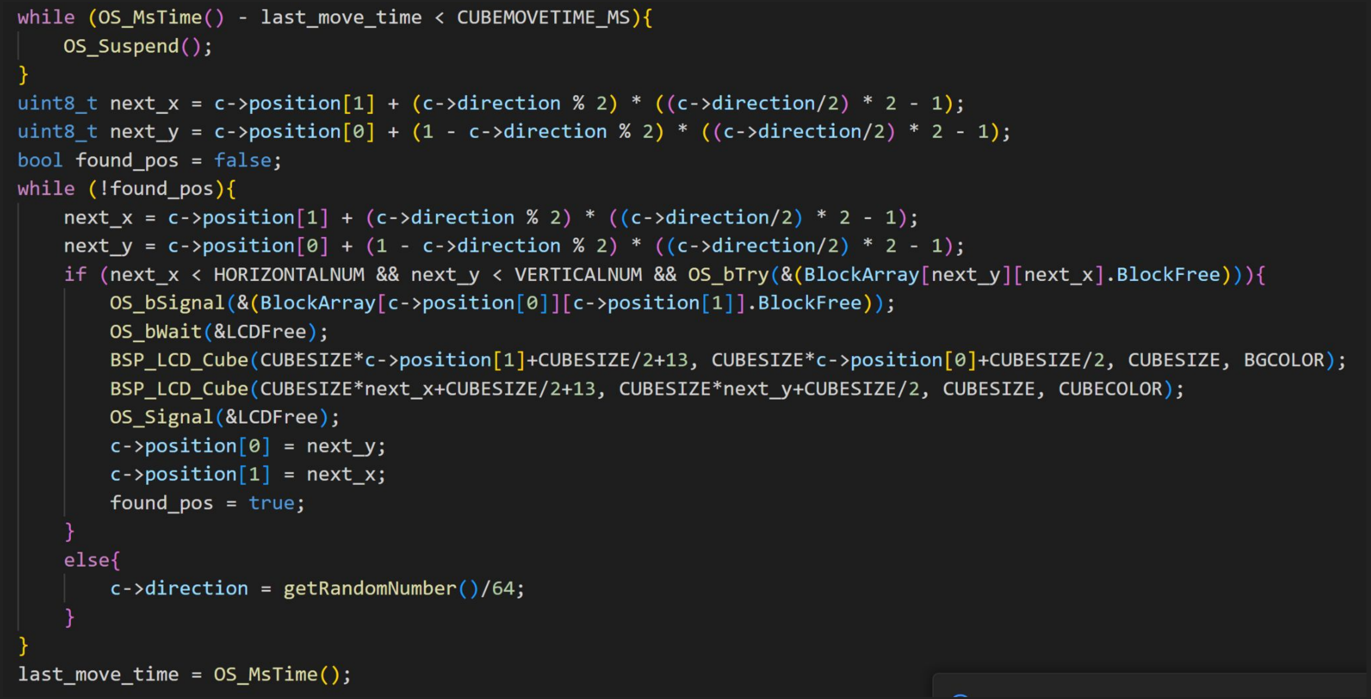
- CubeSpawner (P1) thread spawns a random number CubeThreads (P2)



```
void CubeSpawner (void){
    spawner_active = true;
    while(life){ // Implement until the game is over
        bool blocksExist = true;
        while(blocksExist){
            int i;
            blocksExist = false;
            for (i = 0; i < NUMCUBES; i++){
                if (CubeArray[i].is_alive){
                    blocksExist = true;
                    break;
                }
            }
            if (!blocksExist){
                uint8_t num_cubes = getRandomNumber()/(255/(NUMCUBES)+1)+1;
                uint8_t j;
                for (j=0; j<num_cubes; j++){
                    NumCreated += OS_AddThread(&CubeThread, 128, 1);
                }
            }
            OS_Suspend();
        }
    }
```

Task 4 - Cube Generation & Motion

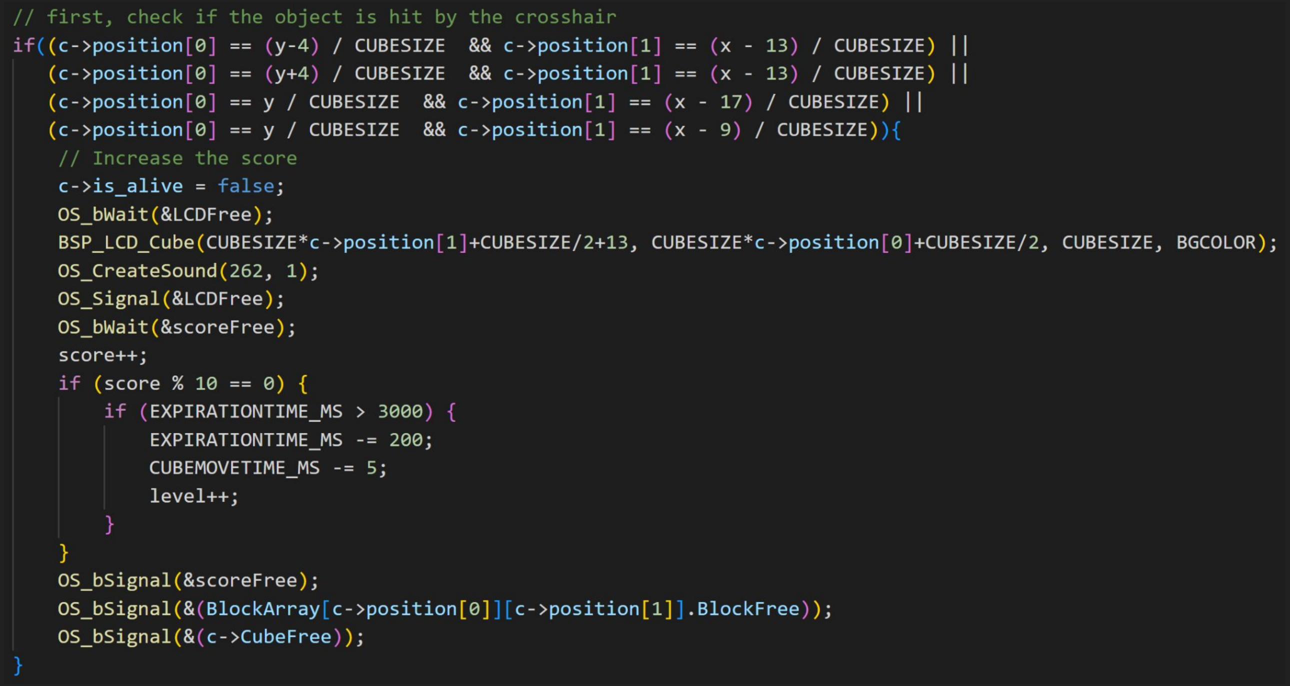
- CubeThread uses an array of cube structs to track cube positions and motion



```
while (OS_MsTime() - last_move_time < CUBEMOVETIME_MS){
    OS_Suspend();
}
uint8_t next_x = c->position[1] + (c->direction % 2) * ((c->direction/2) * 2 - 1);
uint8_t next_y = c->position[0] + (1 - c->direction % 2) * ((c->direction/2) * 2 - 1);
bool found_pos = false;
while (!found_pos){
    next_x = c->position[1] + (c->direction % 2) * ((c->direction/2) * 2 - 1);
    next_y = c->position[0] + (1 - c->direction % 2) * ((c->direction/2) * 2 - 1);
    if (next_x < HORIZONTALNUM && next_y < VERTICALNUM && OS_bTry(&(BlockArray[next_y][next_x].BlockFree))){
        OS_bSignal(&(BlockArray[c->position[0]][c->position[1]].BlockFree));
        OS_bWait(&LCDFree);
        BSP_LCD_Cube(CUBESIZE*c->position[1]+CUBESIZE/2+13, CUBESIZE*c->position[0]+CUBESIZE/2, CUBESIZE, BGCOLOR);
        BSP_LCD_Cube(CUBESIZE*next_x+CUBESIZE/2+13, CUBESIZE*next_y+CUBESIZE/2, CUBESIZE, CUBECOLOR);
        OS_Signal(&LCDFree);
        c->position[0] = next_y;
        c->position[1] = next_x;
        found_pos = true;
    }
    else{
        c->direction = getRandomNumber()/64;
    }
}
last_move_time = OS_MsTime();
```

Task 4 - Cube Generation & Motion

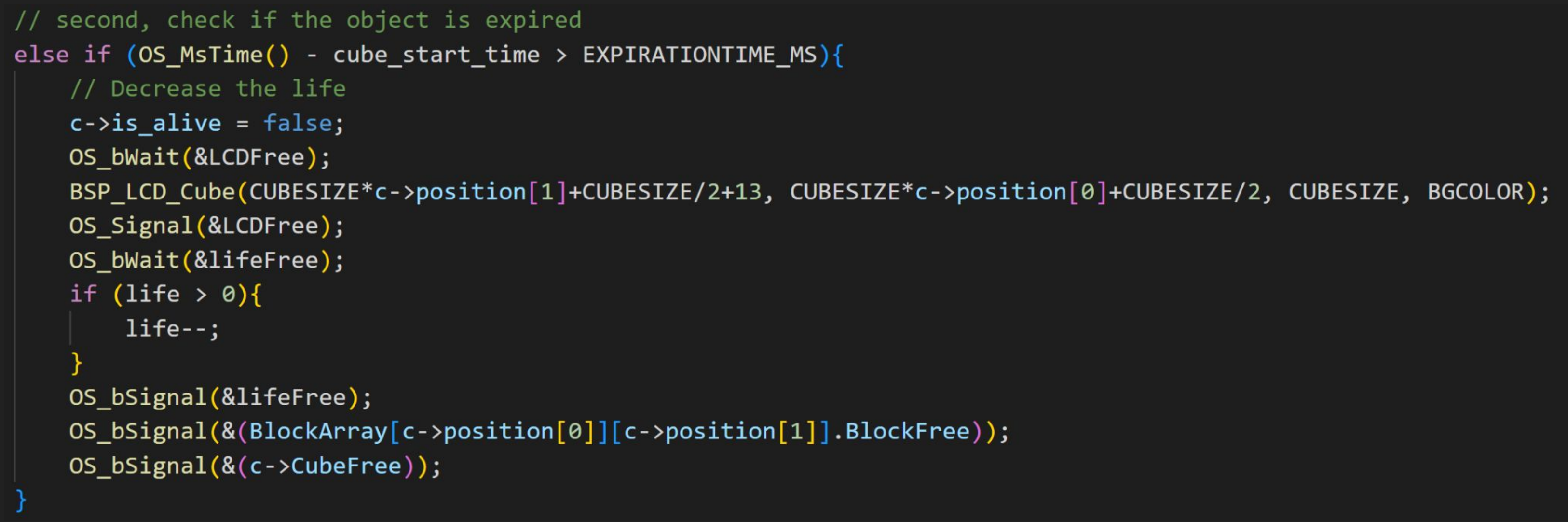
- CubeThread also handles scoring and life loss from cube interactions



```c
// first, check if the object is hit by the crosshair
if((c->position[0] == (y-4) / CUBESIZE  && c->position[1] == (x - 13) / CUBESIZE) ||
   (c->position[0] == (y+4) / CUBESIZE  && c->position[1] == (x - 13) / CUBESIZE) ||
   (c->position[0] == y / CUBESIZE  && c->position[1] == (x - 17) / CUBESIZE) ||
   (c->position[0] == y / CUBESIZE  && c->position[1] == (x - 9) / CUBESIZE)){
    // Increase the score
    c->is_alive = false;
    OS_bWait(&LCDFree);
    BSP_LCD_Cube(CUBESIZE*c->position[1]+CUBESIZE/2+13, CUBESIZE*c->position[0]+CUBESIZE/2, CUBESIZE, BGCOLOR);
    OS_CreateSound(262, 1);
    OS_Signal(&LCDFree);
    OS_bWait(&scoreFree);
    score++;
    if (score % 10 == 0) {
        if (EXPIRATIONTIME_MS > 3000) {
            EXPIRATIONTIME_MS -= 200;
            CUBEMOVETIME_MS -= 5;
            level++;
        }
    }
    OS_bSignal(&scoreFree);
    OS_bSignal(&(BlockArray[c->position[0]][c->position[1]].BlockFree));
    OS_bSignal(&(c->CubeFree));
}
```

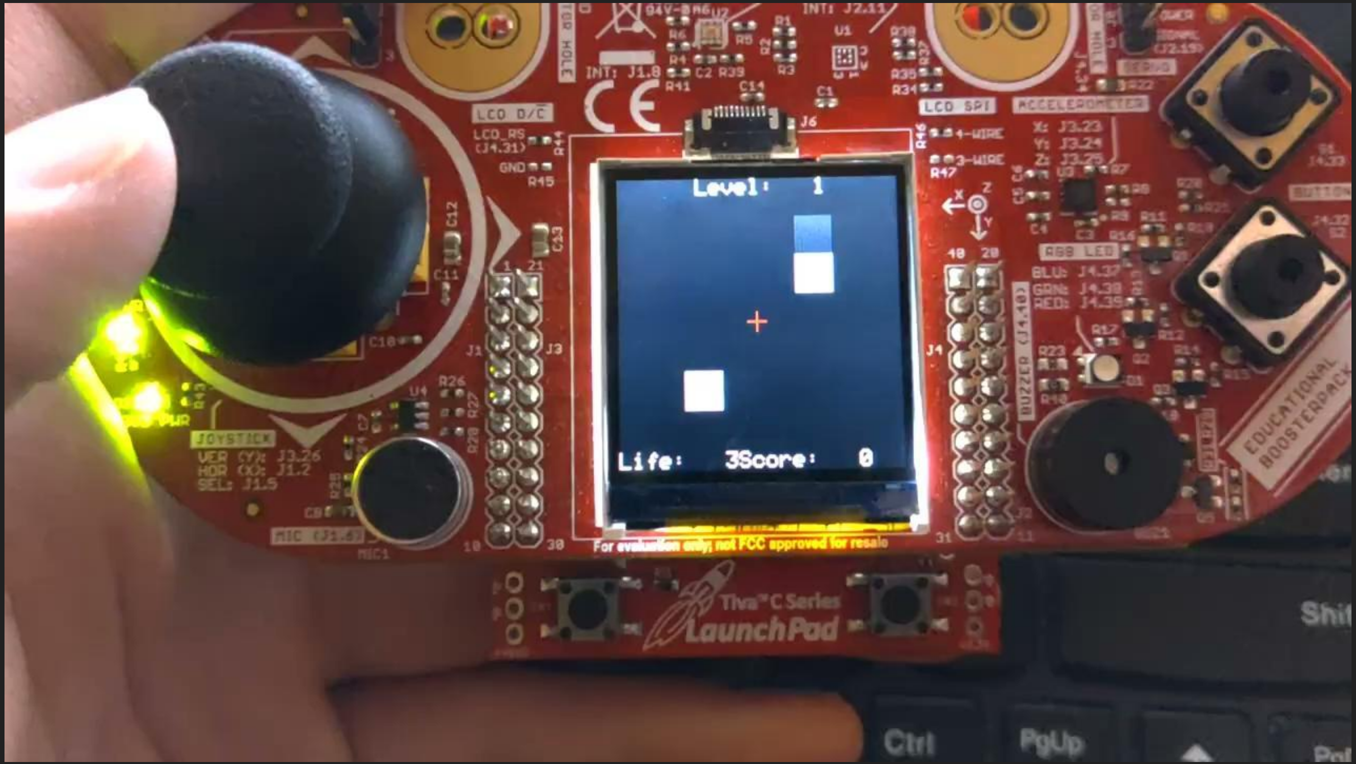
Task 4 - Cube Generation & Motion

- CubeThread also handles scoring and life loss from cube interactions



```
// second, check if the object is expired
else if (OS_MsTime() - cube_start_time > EXPIRATIONTIME_MS){
    // Decrease the life
    c->is_alive = false;
    OS_bWait(&LCDFree);
    BSP_LCD_Cube(CUBESIZE*c->position[1]+CUBESIZE/2+13, CUBESIZE*c->position[0]+CUBESIZE/2, CUBESIZE, BGCOLOR);
    OS_Signal(&LCDFree);
    OS_bWait(&lifeFree);
    if (life > 0){
        life--;
    }
    OS_bSignal(&lifeFree);
    OS_bSignal(&(BlockArray[c->position[0]][c->position[1]].BlockFree));
    OS_bSignal(&(c->CubeFree));
}
```

Task 4 - Demo


Level: 1
Life: 3Score: 0
JOYSTICK
LCD SPI
ACCELEROMETER
RGB LED
BUZZER
EDUCATIONAL BOOSTERPACK
For evaluation only; not FCC approved for resale
Tiva C Series
LaunchPad

Task 5 - Interactive Sound Effects

The TM4C123GH6PM has a pulse width modulated (PWM) signal generator:

- Need to configure hardware accordingly to produce sounds:
  - Piezo buzzer on the MKII boosterpack - connected to port pin PF2
  - This corresponds to generator module 1 block 3 on the microcontroller

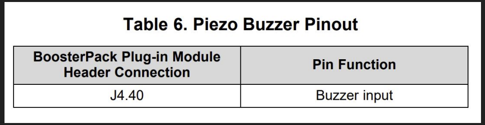
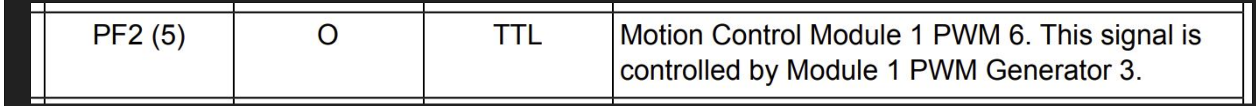
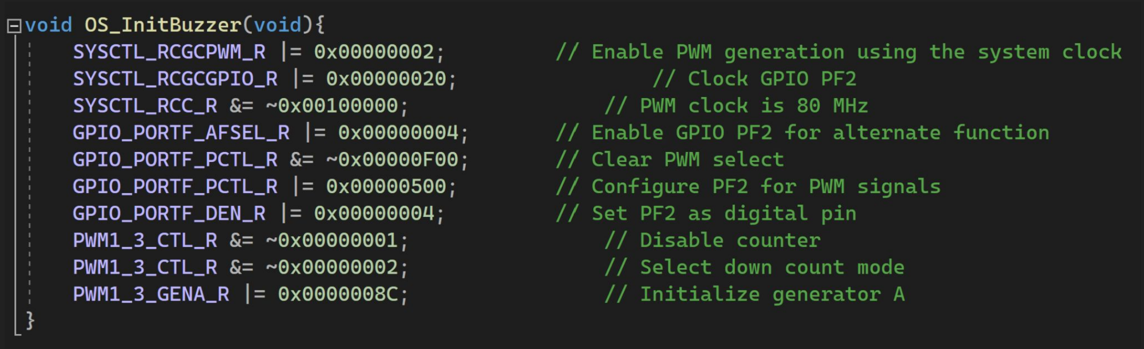


Table 6. Piezo Buzzer Pinout

BoosterPack Plug-in Module Header Connection	Pin Function
J4.40	Buzzer input



PF2 (5)	O	TTL	Motion Control Module 1 PWM 6. This signal is controlled by Module 1 PWM Generator 3.

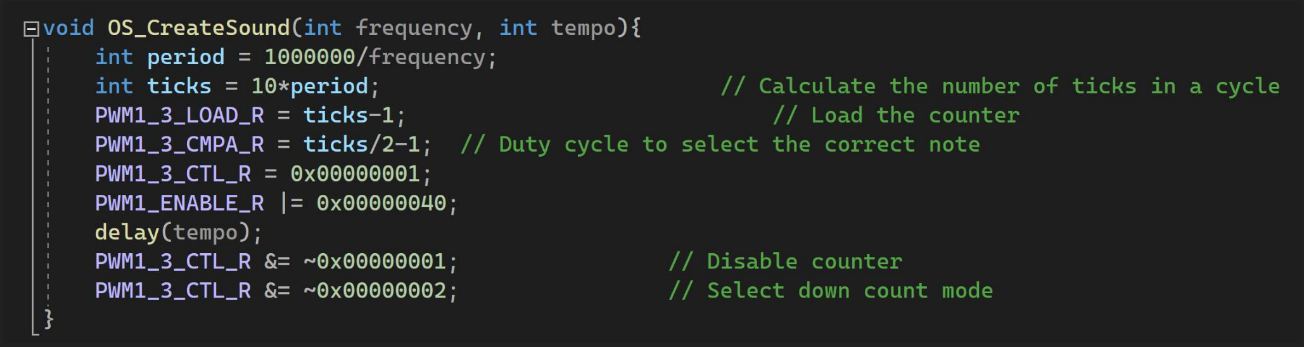


```
void OS_InitBuzzer(void){
    SYSCTL_RCGCPWM_R |= 0x00000002;         // Enable PWM generation using the system clock
    SYSCTL_RCGCGPIO_R |= 0x00000020;                // Clock GPIO PF2
    SYSCTL_RCC_R &= ~0x00100000;                // PWM clock is 80 MHz
    GPIO_PORTF_AFSEL_R |= 0x00000004;       // Enable GPIO PF2 for alternate function
    GPIO_PORTF_PCTL_R &= ~0x00000F00;       // Clear PWM select
    GPIO_PORTF_PCTL_R |= 0x00000500;        // Configure PF2 for PWM signals
    GPIO_PORTF_DEN_R |= 0x00000004;         // Set PF2 as digital pin
    PWM1_3_CTL_R &= ~0x00000001;                // Disable counter
    PWM1_3_CTL_R &= ~0x00000002;                // Select down count mode
    PWM1_3_GENA_R |= 0x0000008C;                // Initialize generator A
}
```

Task 5 - Interactive Sound Effects

To produce music, need to set the frequency and tempo of each note:

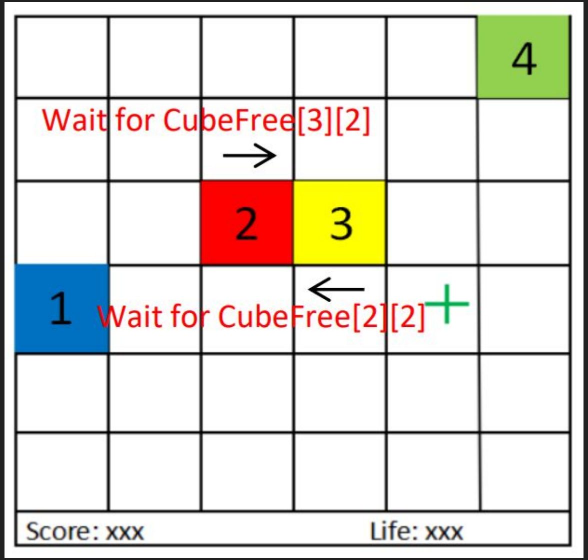
- A specific note is determined by its frequency
  - Choose a 50% duty cycle to keep the fundamental frequency
  - Other duty cycles will pass dissonant frequencies through the reconstruction filter - bad sound!
- Tempo is set by a delay function
  - Both frequency and tempo are combined into one function, OS_Music()

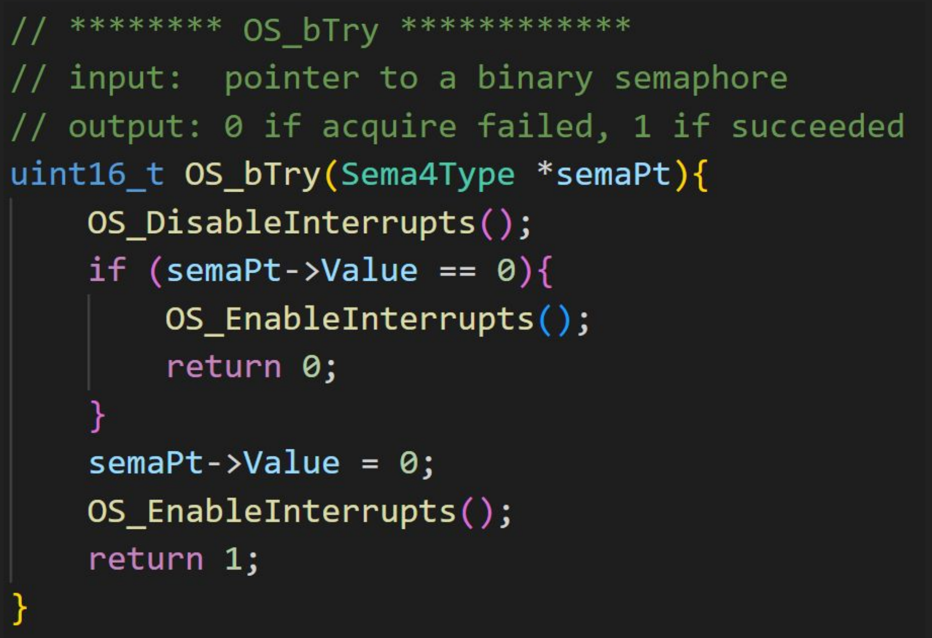


```c
void OS_CreateSound(int frequency, int tempo){
    int period = 1000000/frequency;
    int ticks = 10*period;                          // Calculate the number of ticks in a cycle
    PWM1_3_LOAD_R = ticks-1;                            // Load the counter
    PWM1_3_CMPA_R = ticks/2-1;  // Duty cycle to select the correct note
    PWM1_3_CTL_R = 0x00000001;
    PWM1_ENABLE_R |= 0x00000040;
    delay(tempo);
    PWM1_3_CTL_R &= ~0x00000001;                // Disable counter
    PWM1_3_CTL_R &= ~0x00000002;                // Select down count mode
}
```

Task 6 - Deadlock Prevention

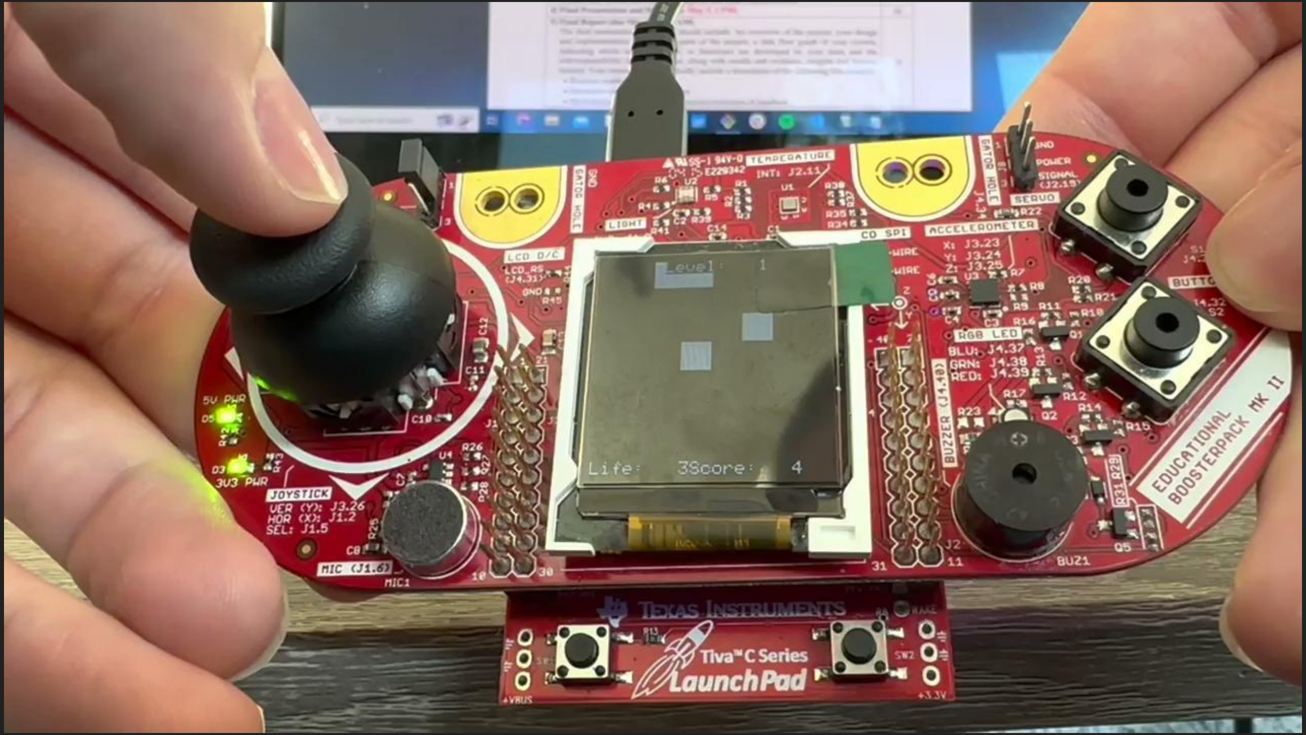
- Created new semaphore wait scheme
  - Return 0 if semaphore was not successfully acquired, 1 if successful





```
// ******** OS_bTry ************
// input:  pointer to a binary semaphore
// output: 0 if acquire failed, 1 if succeeded
uint16_t OS_bTry(Sema4Type *semaPt){
    OS_DisableInterrupts();
    if (semaPt->Value == 0){
        OS_EnableInterrupts();
        return 0;
    }
    semaPt->Value = 0;
    OS_EnableInterrupts();
    return 1;
}
```

Task 6 - Demo


Level: 1
Life: 3Score: 4
EDUCATIONAL BOOSTERPACK MK II
JOYSTICK
BUZZER (J4.40)
ACCELEROMETER
RGB LED
TEMPERATURE
MIC (J1.6)
SERVO
LIGHT
Texas Instruments
Tiva™ C Series
LaunchPad

Lessons Learned

- Learning the intricacies of game development, thinking about collisions, cursor and cube movements etc.
- Using semaphores in real-life applications
- Collaboration process through github
- The importance of meeting deadlines