




ECE 4501 Spring 2024: Team Zeta Final Project

But can it run DOOM?

John Berberian, Sarah Hemler, Paul Karhnak, Casey Ladd

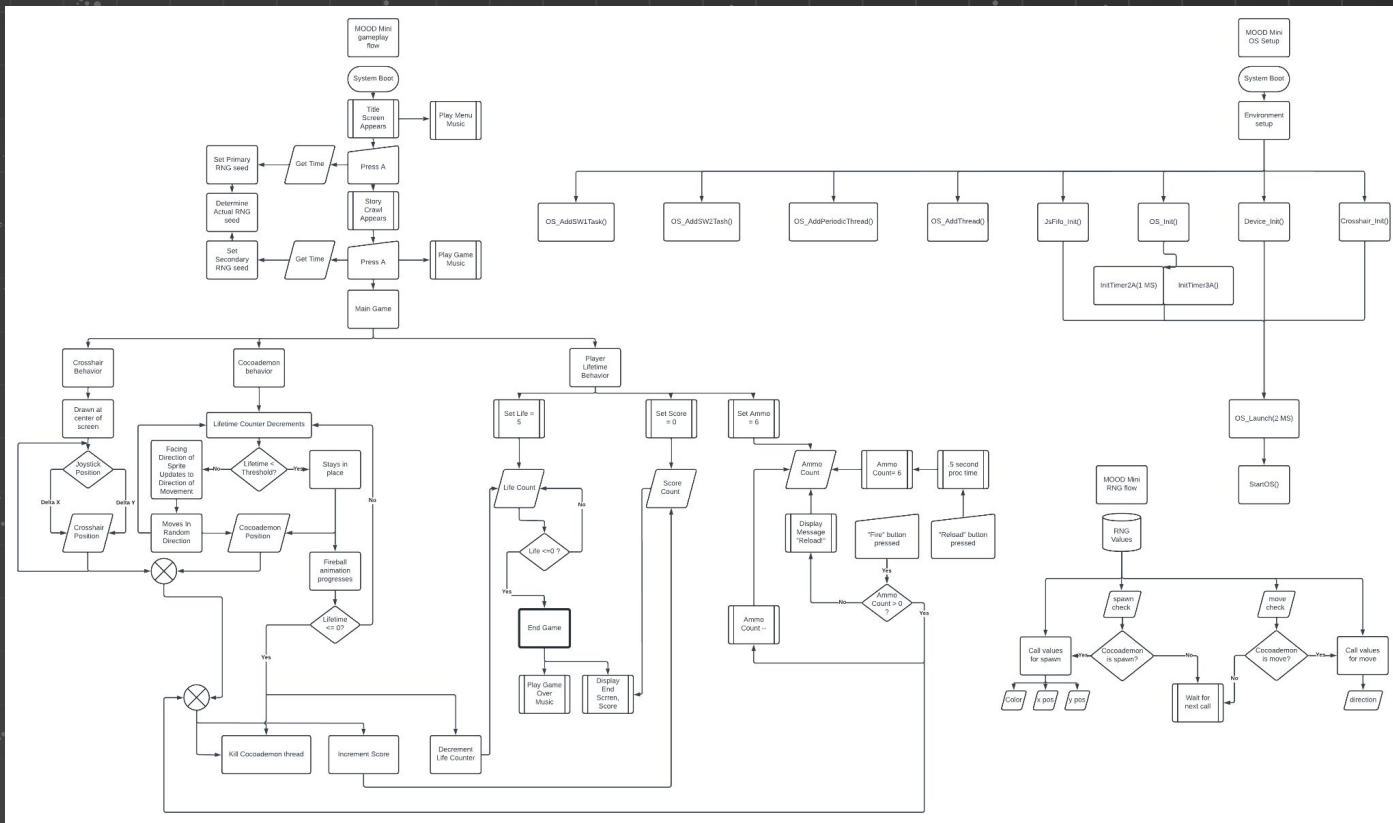


Concept

MOOD: DOOM on the Tiva Board

- Scale matters
 - Tiva board constraints: 256 KB flash, 32KB program memory
 - Full-fledged video game (sound, player input, graphics) required substantial use of onboard resources and peripherals
- Stretching the board (and our kernel) to its limits
 - PWM for sound
 - Timers for kernel and system time
 - GPIO for pushbuttons
 - Routines to handle joystick and produce LCD outputs

We started out a little ambitious...





Design

Implementing RNG

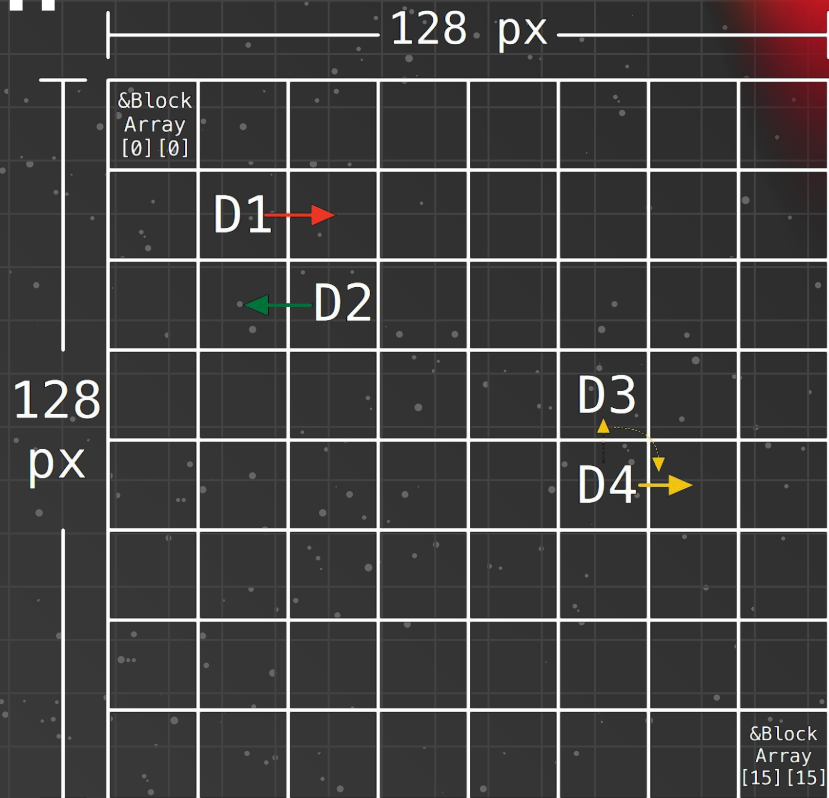
- XORshift generator from [Marsaglia\(2003\)](#)
 - Ideal for fast, efficient number generation: just moves and bitwise
 - High period even for 32-bit ($2^{32} - 1$).
 - Seeded with system time (from `OS_Time()`) when needed for better randomness

Rather than having our enemies move in a predictable path, we opted to have them move in random directions each step of the gameplay, making it harder to anticipate their movements, and thus, making them harder to target.

```
uint32_t rng(void) {  
    // not shown: seed CurrentRandomValue if needed  
  
    CurrentRandomValue ^= (CurrentRandomValue << 5);  
    CurrentRandomValue ^= (CurrentRandomValue >> 27);  
    CurrentRandomValue ^= (CurrentRandomValue << 25);  
  
    return CurrentRandomValue;  
}
```

Deadlock Prevention

- Dividing the LCD
 - Each demon (cube sprite) 16x16, so divide LCD into an 8x8 grid of 16x16 cells
 - Create array of corresponding semaphores
- OS_Try(): attempt to acquire semaphore without suspend
 - If demon cannot acquire semaphore, RNG new direction and repeat
- Preventing deadlock
 - Eliminates circular wait
 - Introduces a flavor of resource preemption



Sensory Feedback- Power to the Player

We noticed that, in the cube game demo, there were no clear indicators for when a cube object was reaching the end of its life.

In order to enhance gameplay experience and communicate status of objects and the gameplay loop to the player, we implemented the following features:

- As an enemy nears the end of its lifespan, it begins to play an attack animation
- If an enemy is defeated, a high-pitched note plays, informing the player that they successfully eliminated the enemy
- If an enemy reaches the end of its life, a lower-pitched tone plays, informing the player that they failed to defeat the target in time and have taken damage
- At the end of the game, the player's overall score is visible at the bottom of the screen, along with their life count (telling them whether the game over was due to loss of life or a successful traversal of all levels)
- We also include a small ammo readout at the bottom of the screen. If the player runs out of ammo, they will not be able to defeat enemies. Keeping track of their ammo is key to success!

Graphics



Bitmaps

**16-bit high
color!**

**High-Quality
Graphics!**

**Using arrays
to animate**

The LCD screen supports a range of colors that use 5 bits for red, 6 bits for green, and 5 bits for blue

The large sprites from DOOM were not viable for our tiny display, and they were too large to automatically scale down, so we recreated miniature sprites from scratch

We arranged our sprites into 16x16 pixel “frames” and then used code to move through an array to choose different frame for different situations, allowing us to animate frame by frame. “Blocks” were a set height and width, and were defined by an offset of where they started in the array



Sound

Pulse Width Modulation (PWM)

- Digitally encoded analog signal levels
 - High-resolution counter to generate a square wave
 - Duty cycle to modulate a square wave
- Contains two PWM modules each with 4 generator blocks and a control block that controls polarity
- The buzzer is connected to pin J4-40 which is equivalent to pin PF2 on the microcontroller
 - Motion Control Module 1 PWM 6
 - Controlled by Module 1 PWM Generator 3
- The load value was the value that represented the frequency based on the clock frequency:
 $\text{clock_frequency_with_divider} / \text{pitch_frequency}$
- The duty cycle was always 50% of the load value to create a square wave with 50% on the rising edge
- A clock division of 8 allows for the generation of lower frequencies that couldn't be created using the regular clock frequency

Game Music and Sound Effects

- Every song was coded note by note, as objects in an array, with three data points per element: pitch, duration, and pause afterwards.
- All pitches were defined in one place, allowing us to indicate pitches by note names everywhere else.
- We have three songs: E2M6 (menu screen), E1M1 (main gameplay loop), and E2M3 (endgame screen). We also have two pitches that play depending on whether an enemy is defeated or timed out. An upward pitch plays if they're defeated, while a downward pitch plays if they time out and damage the player.
- The program iterates through the current tune, and resets the iteration counter whenever songs are swapped (the swap is indicated by a single variable)

Lessons Learned

(Un)expected Snags

- The ARM Compiler v5.07 only allowed a lite version of Keil compilation, limiting the size of our game image
- Getting sound to work took significantly more time and effort than expected- we had to switch from song functions to defining our songs as arrays
- Bitmaps occupy tens of kilobytes of flash memory in the final game, so we had to switch compilers to be able to use the open license version of Keil
 - Keil's community license did not work with compiler v 5.06, and we spent the better part of a day figuring out how to migrate our work to a new compiler version (we ended up needing to make a new file, LCDasm.s, to accommodate)
- While we had originally planned to require the user to push a button to attack enemies rather than just touching them with a crosshair, that design element was ultimately abandoned.
- We also planned to add a second buzzer to our board in order to achieve harmonies in the soundtrack, but we ran out of time

Team Responsibilities

- Joystick Input & Crosshair Display
 - Sarah, John, Paul, and Casey
- Game Scoring and Panel Display
 - John and Sarah
- Random Number Generator
 - Paul
- Cube Generation & Motion
 - John and Sarah
- Interactive Sound Effects, Sound Array Creation, and Implementation
 - Casey and Sarah
- Deadlock Prevention
 - John and Paul
- Graphic Design
 - Sarah
- Graphic Implementation and Animation
 - Sarah, John, and Paul