William Zhang, Jacob Blindenbach, James Houghton, Guy “Jack” Verrier
wyz3sp, jab7dq, jth5zs, gjv7qw
December 1, 2020

Final Project Part 1 Team Alpha Report

Overview:

Threads:

Consumer:

Draws cursor and display life / score

MoveCube (Threads 0 – 4):

Moves a cube a random direction that is not blocked

Draw Blocks:

Draws the cubes on the screen

Producer:

Updates the location of the crosshair based on the joystick input

SW1 Task:

Displays the life time, horizontal and vertical area, and elapsed times

SW2 Task:

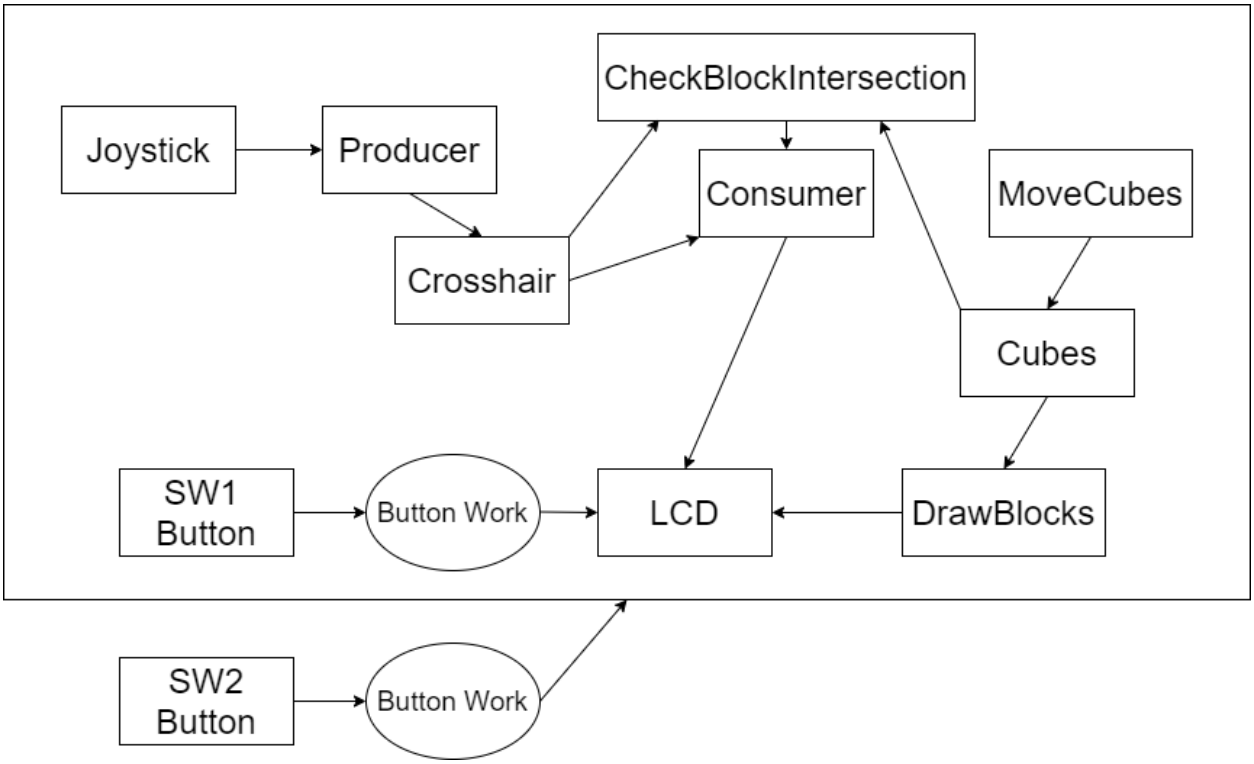
Resets game by resetting semaphores and restarting threads

Init And Sync Blocks:

Called in the beginning to initialize the threads for each cube/block

Ideal Thread:

A thread that does not do any work, but is necessary to prevent the OS from locking up when all the threads are killed during a restart



All of the code changes were in *Main.c*. We kept all the other files the same from the solution of mini project 4.

Roles and Responsibilities:

James: Parts 4 and 5, cube logic, deadlock prevention, reset functionality, debugging and finalizing the code
William: Crosshair cube detection, cube logic, video
Jacob: Revised miniproject 4 to do tasks 1 and 2, wrote report
Jack: Part 3, random number generator

Random Number Generator:

To generate random numbers, we used the LFSR implementation linked on the assignment. The only changes we made to the implementation was how they seeded their initial values. We modified the init function to get seeds from joystick positions.

```
void init_lfsrs(uint32_t x, uint32_t y) {
        lfsr32 = x;
        lfsr32 = y;
}

...

seedA = (rawX << 16 | rawY);
seedB = (rawY << 16 | rawX);
init_lfsrs(seedA, seedB);
```

The first seed is set to be the 16 bits of the joystick x position followed by the 16 bits of the joystick y position. The joystick positions are outputted as 16 bits so oring them together gives a full 32 bit output where each bit is randomized. The second seed is set to be the last 16 bits of joystick y position followed by the 16 bits of joystick x position.

Deadlock Prevention:

We prevent deadlock with an early detection mechanism that never allows a cube to wait for a semaphore (grid location) that is taken by another cube. In the method to move a cube, we disable interrupts.

```
status = StartCritical();
total_valid_dirs = get_movable_directions(cube, valid_directions);
...
OS_bWait(&blocks[new_y][new_x]); // this should never block
OS_bSignal(&blocks[cube->y][cube->x]);
EndCritical(status);
```

After disabling interrupts, we check all the available directions this cube can move.

```
if (cube->x > 0 && blocks[cube->y][cube->x - 1].Value) {
...
}

if (cube->x < HORIZONAL_NUM_BLOCKS - 1 && blocks[cube->y][cube->x + 1].Value) {
...
}
```

To check a direction, we not only see if the new location in that direction would be off the screen but also if the new locations semaphore is not blocked. If it is, the direction is not valid and ignored. Therefore, any directions we pick, the semaphore will not be blocked and not deadlock can happen. It is important that this calculation is in a critical section because this prevents other cubes from holding onto semaphores for directions which were initially valid.

Video:

https://drive.google.com/file/d/1A6c7ONvJpz8QVLXhMxelg6o2jh3J-3mz/view?usp=sharing

Survey: