Margaret Sands and Joanna Sands

Scream-y bird

Project Overview

For this project, we plan to make a game in which the player's sprite is controlled by vocal variation. Specifically, we hope to take in a player's vocal cues, identify the frequency and use the frequency in order to direct the sprite. Game play would consist of a side scrolling world in which the player can only move vertically in order to dodge incoming objects.

In order to modularize this design, we have identified different programmable game modes that increase in complexity but can build on one another as we move towards the final product.

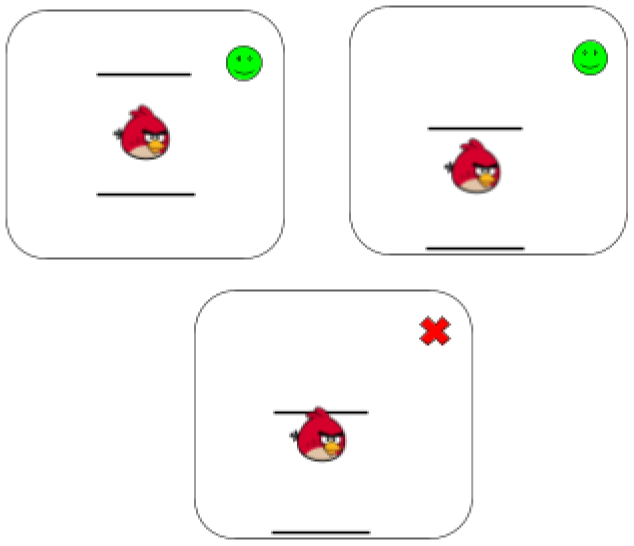

Figure 1. Basic Game Example

Basic Gamemode:

In the basic game mode, there are three blobs on screen, a character sprite and two walls that move vertically up and down in a periodic fashion. The goal of the game is for the player to last any many seconds as possible while preventing the sprite from touching the walls by moving the sprite in time with the motion. The sprite will be controlled by the player "screaming" into the microphone. As the player raises/lowers their pitch, the sprite will move up/down on the screen. A basic example of the rule set can be seen in the diagram below.

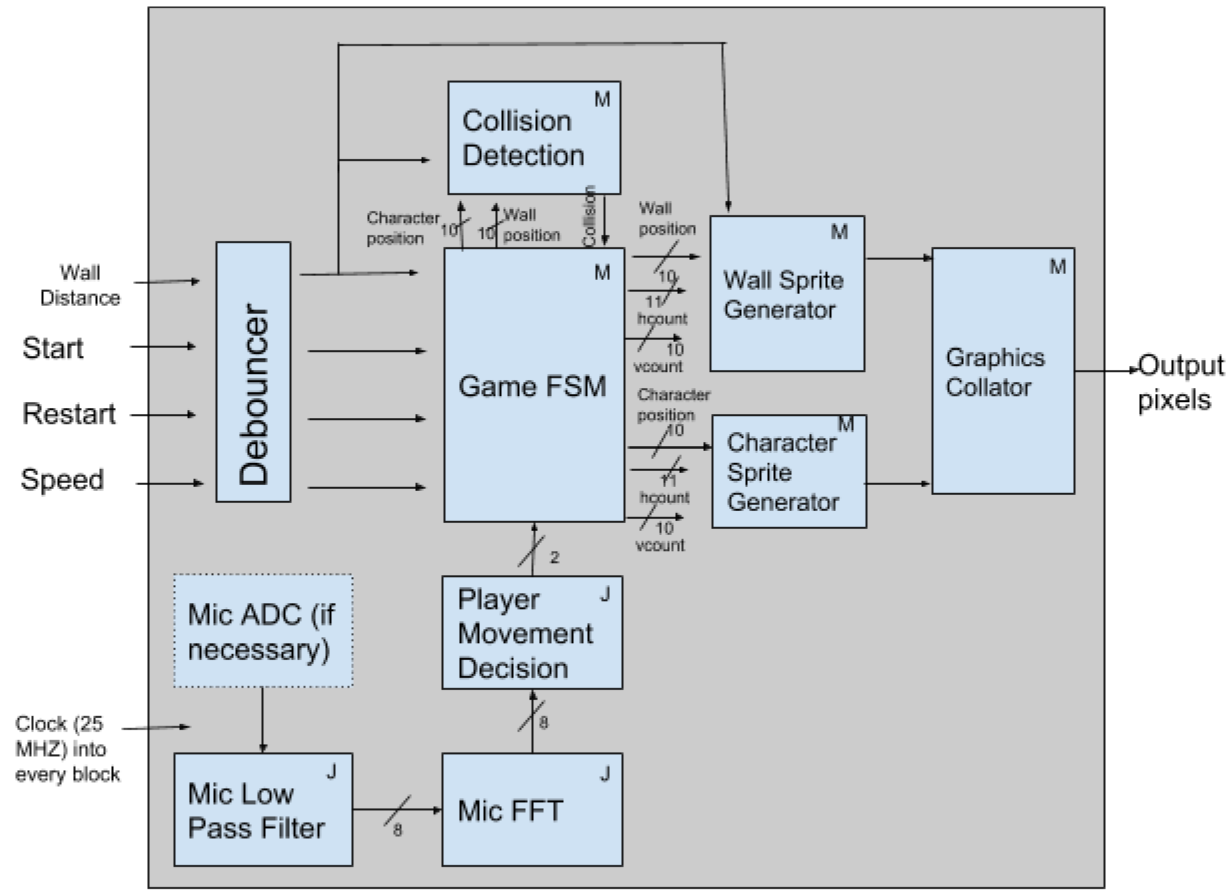

Figure 2. Basic Game Block Diagram

Modules for basic gameplay:

In this minimalized gameplay, we will fully implement our movement functionality, build an FSM that will keep track of game state, create a system for detecting collisions and make our first steps towards obstacle generation.

To do this, we will focus on creating 8 specific modules and testing their functionalities both physically and in simulation.

1. The Mic Low Pass filter will take in the signal from the microphone and pass it through a low pass filter in order to remove noise in the signal that could continue on into the output of the FFT.
    a. To test this module we can process the signal and send it to an audio output located on the Nexys 4 board. If the output is an analog wave with the same base frequency as the input wave, then we will know it is working.
2. The Mic FFT module will take in the filtered signal and use the Nexys 4 FFT IP module to extract the pitch from the mic input.
    a. To isolate the FFT module, we can use the test bench code from Lab 5 in order to make sure the result is as we expect.
3. The Movement Decision module takes in the dominant pitch from the FFT and determines the direction the sprite should move in the next movement step.

    a. We can create a test bench for this with different FFT result formats along with their desired results. To test the integration for these first three steps, we can create a smaller system in which a player “screams” into the microphone and we display the determined movement direction using LEDs.

4. The Game FSM is responsible for keeping track of the location of the character sprite, of whether or not the game has started, and of whether or not the player has lost.
    a. Before we get the other parts working, we can test the game FSM using debounced buttons to simulate different desired inputs that come from other modules, like movement direction. Once this is done, we can use the hex print out on the seven-segment LEDs to monitor game state transitions as well player and obstacle positions.
5. The Collision Detection module takes in a player’s position and the position of their obstacles and raises it’s output high if the players sprite overlaps with any of the obstacles or the top and bottom of the screen.
    a. For simulation testing, we can create arrays of example positions and check the simulation to make sure the collision output is high when necessary.
6. The Character Sprite Generator uses the sprite bitmap image and the player position in order to set the pixels to display the character.
    a. To test this, we can use the module from pong to make sure that the sprite can display at different heights on the monitor.
7. The Wall Sprite Generator creates two elongated blobs that are a set distance apart and will act as moving barriers to the player.
    a. Like the character generator, we can test this by making sure the blobs display properly when given different heights and initialized to different widths.
8. The Graphics Collator takes in the generated pixel arrays and combines them to make the full display.

Game Goals

In the effort to move towards full desired gameplay, we want to create a second game mode that adds side scrolling obstacles that move towards the player over time as well as a scoring system based on how long the player can survive that will display on the 7-segment displays on the nexus7. We also want to add a background that improves the ambiance of the game.

In order to make side scrolling possible we essentially want a game map that is much larger that the size of the screen and we want to consider the player view to be the areas of the map that are within a given number of pixels from the players sprite. By keeping track of the player’s x position in this larger map, we can determine which of the equidistant obstacles are visible at a given time. Additionally, by taking into account the obstacles position on the larger map, and the players location, we can determine where on the screen to display the obstacles. This will be done in the Obstacle Position module.

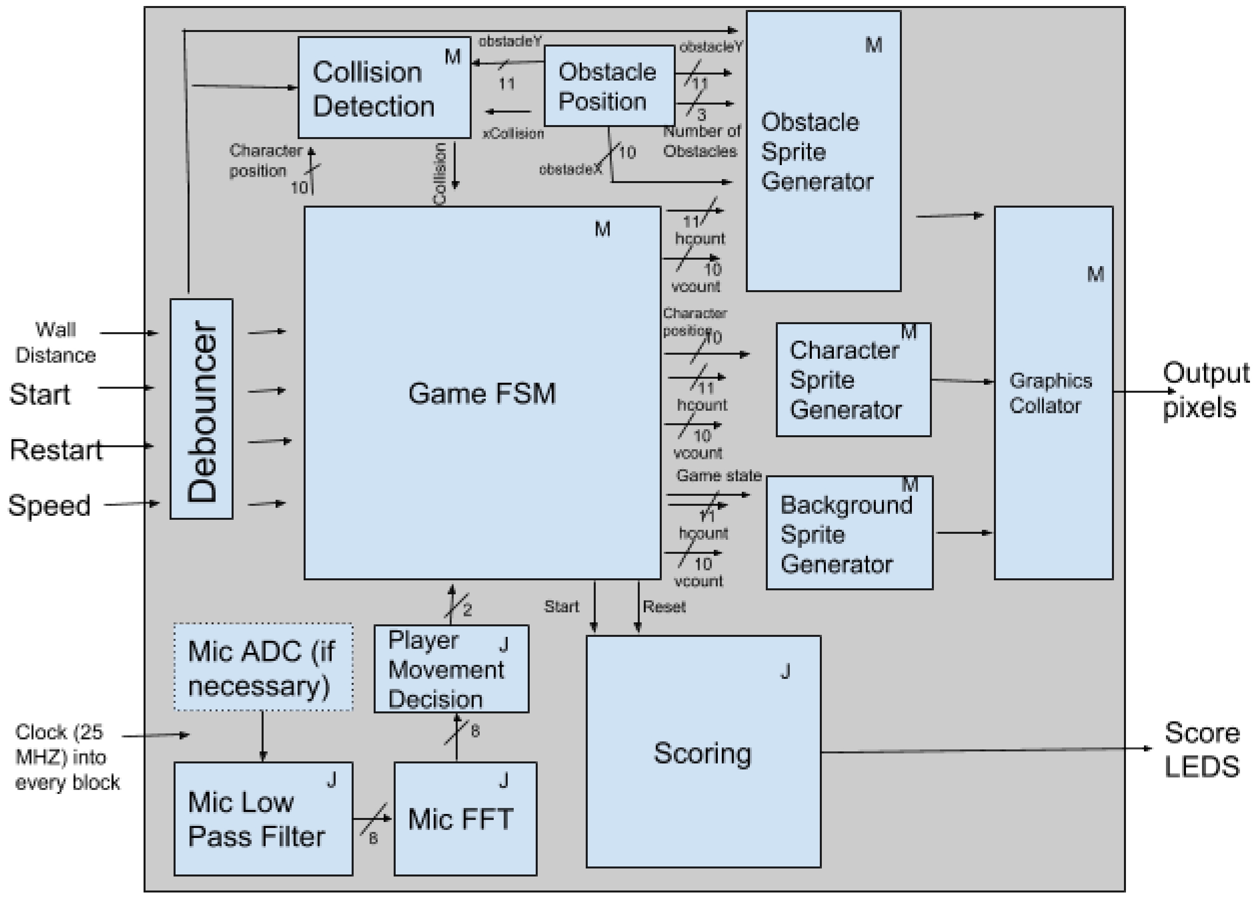

Figure 3. Game Block Diagram

To actually generate the pixel array, we will modify replace the wall sprite generator module with an obstacle sprite generator that will take in the number of obstacles and will sequentially read in the the leftmost x-value and the bottom value of the obstacle gap for each obstacle.

For updated collision detection, we will leave the collision detection module almost the same, but will instead use a signal to tell it when the bird and an obstacle are in the same horizontal range, and will take in the y-value of the bottom of the closest opening.

We also want to make some details of the map like the distance between obstacles and the speed at which the bird moves between them parameters that can be set using inputs from switches.

To make this transition, we will add three modules, one for each additional goal.

1. The Obstacle Position module will perform the necessary actions described above to provide information to both the collision detection and obstacle sprite generator.

2. The Background Sprite generator module which will display a different background based on game state. We plan to start out with very minimal states so the background would only be a single image the size of the screen that loops.
3. The Scoring module which will count the number of seconds that the player has stayed in the game without hitting an obstacle and use the display_8hex module that was used in lab 4. It will need to take in a score reset signal as well as a start signal in order to count correctly. We can visually test it using the hex print out.

Things to add to the last one: sound, additional game modes (continuous wall), second axis of movement, vertically moving enemies that are side scrolling, game mode w/ gravity

Reach goals:
If we can make a fully functioning version of our mid-tier goals, there are a lot of came aspects that we would like to add. These include making new game modes like having a continuous wall that needs to be followed instead of rectangular obstacles, creating vertically moving enemies instead of static ones,implementing gravity or allowing movement in the x-axis as well as the y.

Possible Issues:

We think most of the issues we will run into will involve managing memory for obstacle and background designs and may result in having to think of ways to add more memory. That being said, after identifying a specific microphone, we may run into issue of incompatible data types between the FFT IP and the chip that feeds in microphone input.