Daniel Gerber
Tynan Smith
10/30/08

Poke Kirby with a Laser
Project Proposal and Block Diagram

Our project will be a hardware-implemented game. Specifically, it will likely be a multiplayer combat game featuring Kirby, but the rules and details of the game will be decided at a later time. Players will control the game through laser and keyboard input. Each player will wield a laser pointer that will be pointed at a location on screen. Control of the player's character will involve keystrokes followed by laser pointer gestures. The gestures will be processed and recorded and the character's actions will be dictated by the combination of these inputs.

This project will require five high level modules. Each of these five high level modules will very likely require several of their own modules as well. The five high level modules are the video analyzer, the gesture processor, the game logic, the keyboard input processor, and the graphics processor.

The video analyzer will take input from the camera's NTSC signal. This module will then produce five outputs. The first two outputs are the x and y pixel coordinates of the red laser dot and the other outputs are the x and y pixel coordinates of the green dot. The fifth output will be a simple ready signal that will be used by the gesture processor to synchronize with the video analyzer module. The video analyzer will ultimately receive several pixels of the red dot and the green dot since each dot will be several pixels in diameter. In addition, to average the pixels of the dot in order to find its actual coordinates, a divider will be required. However, it is easy to divide by powers of two, so we will likely just take the first four or eight pixels of that color that have been detected and shift the sum of the coordinates by the appropriate power in order to average the coordinates. The video analyzer will know whether a dot is not on the screen if it does or does not receive enough pixels of the expected color. Its output for that color's coordinates will be the highest possible output for that coordinate, and will be appropriately interpreted by the gesture recognizer as no dot. The coordinates of each dot will also be sent to the game logic for the purpose of pasting a target on screen and other such functions.

We believe that ultimately, we will not need to store the entire video frame in RAM for each frame that comes from the camera. We can probably intercept the input as it comes in and convert it into pixel data. From that, the module will increment a counter if it is red or green, save the coordinates, and then discard the pixel data altogether. This module will be tested first by outputting the video input to the monitor. After that, we will try to get the hex display to output the coordinates of a laser pointer. We will test the laser on white paper first because it will have a good contrast and very little interference. It has already been decided that Daniel Gerber will work on the video analyzer module.

We are currently planning on implementing the gesture recognition module as a fairly complex FSM. There will be one copy of the module to analyze the green laser and one copy to analyze the red laser. Each module will take as input the ready signal and

coordinates from the video processor as well as certain inputs from the keyboard. It will produce as output the detected gesture, a ready signal, and the relative speed of the gesture to the game logic which will use the input to control the characters. The gesture module will have two main components. The first component will be an FSM that uses successive coordinates to determine the motion of the laser in small chunks. The second component will look at the series of motions made and determine if a valid gesture was formed and send the appropriate signal to the game logic.

Whenever a ready signal is sent by the video processing module the first gesture module will grab the next set of coordinates. It will start in a default state of no gesture being input and will wait until valid coordinates are given. Then it will transition to a state where it will save the last few coordinates (a number yet to be determined) and analyze their motion until it determines the initial motion of the laser. At this point it will enter the main states of the FSM where it keeps a buffer of the last few coordinates given and looks at the motion and changes states based on the motion. The motion will be divided into 24 possible categories/states two each for the two possible directions of motion of the following possible paths: horizontal line, vertical line, 45 and 135 degree diagonal line, four quarter circles in vertical/horizontal orientation, and four quarter circles rotated by 45 degrees from the other four. The state machine will look at the motion and the current trajectory to determine when the coordinates have differed from the current trajectory and onto a new one. This will involve a fair amount of arithmetic, particularly calculating distances and vectors, but shouldn't take an excessive amount and shouldn't use too much memory. As the state machine changes states it will save a history of the states. It will also have a counter that keeps track of how many coordinates have been given since the start of the gesture. A gesture will be completed and the FSM will return to the original state when the laser pointer leaves the screen (invalid coordinates are sent) or when the laser stops moving (the distance between coordinates is less than some threshold for some amount of time as indicated by another counter). There will also be a maximum number of states that will be recorded, at which point the gesture will also be terminated. When the gesture is determined to have ended, the recorded segments of motion, as well as the number of coordinates in the gesture will be sent to the next gesture module. A ready signal will tell the next module to start processing the given data.

The second gesture module will analyze the series of segments given by the FSM gesture module and compare them to sequences of segments that correspond to gestures. It will use a little bit of error minimization logic to determine the most likely gesture if there is no perfect match. This logic will take into consideration extra motion at the beginning and end of gestures, misinterpreting similar segments, and similar things. If no gesture is a close match, the data will be disregarded. If there is a match then the relative speed of the gesture will be determine by comparing the number of coordinates it contained to certain thresholds. The gesture and its speed will then be sent to the game logic and indicated by a ready signal. Both gesture modules will be created by Tynan.

The game logic module will take four coordinate inputs from the video analyzer, some number of inputs from the keyboard module, and three inputs from each gesture module. Its outputs will be the x and y coordinates for all of the sprites. The role of the

game logic is to synthesize all of the inputs to the game and calculate where the objects in the game should be, what they should be doing, and what their status should be. This module is where all of the game programming will be. The rules of the game have not yet been decided, but it will probably be a combat game with each player controlling a different character. The game logic will position the sprites based on internal variables and inputs and will output these positions. All of these internal variables can probably be stored in registers. All of the calculations will probably involve nothing more than a lot of signed addition and comparisons (for collision detection). To test the game logic module, we will use the hex display to output various internal variables. If the sprites have been developed and tested properly, we can even test the game logic on the monitor. Tynan Smith will program the game logic module.

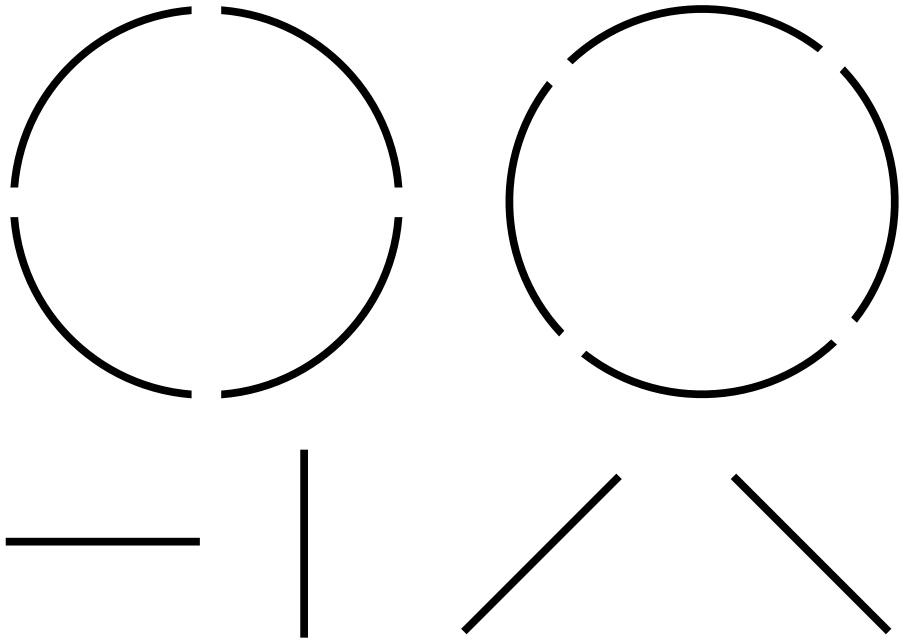
The keyboard module will handle keyboard input and will send it to both the game logic and the gesture modules. This module may already exist, but it will probably have to be modified slightly in order to make it output the proper signals for our project. Its input will be from the keyboard itself. Its output may be the keyboard data, or it may be a simple number. This module will not need to store anything in memory and it will not involve any intensive calculations. Testing of this module will involve simply outputting the key pressed to the hex display. Daniel Gerber will program the keyboard module.

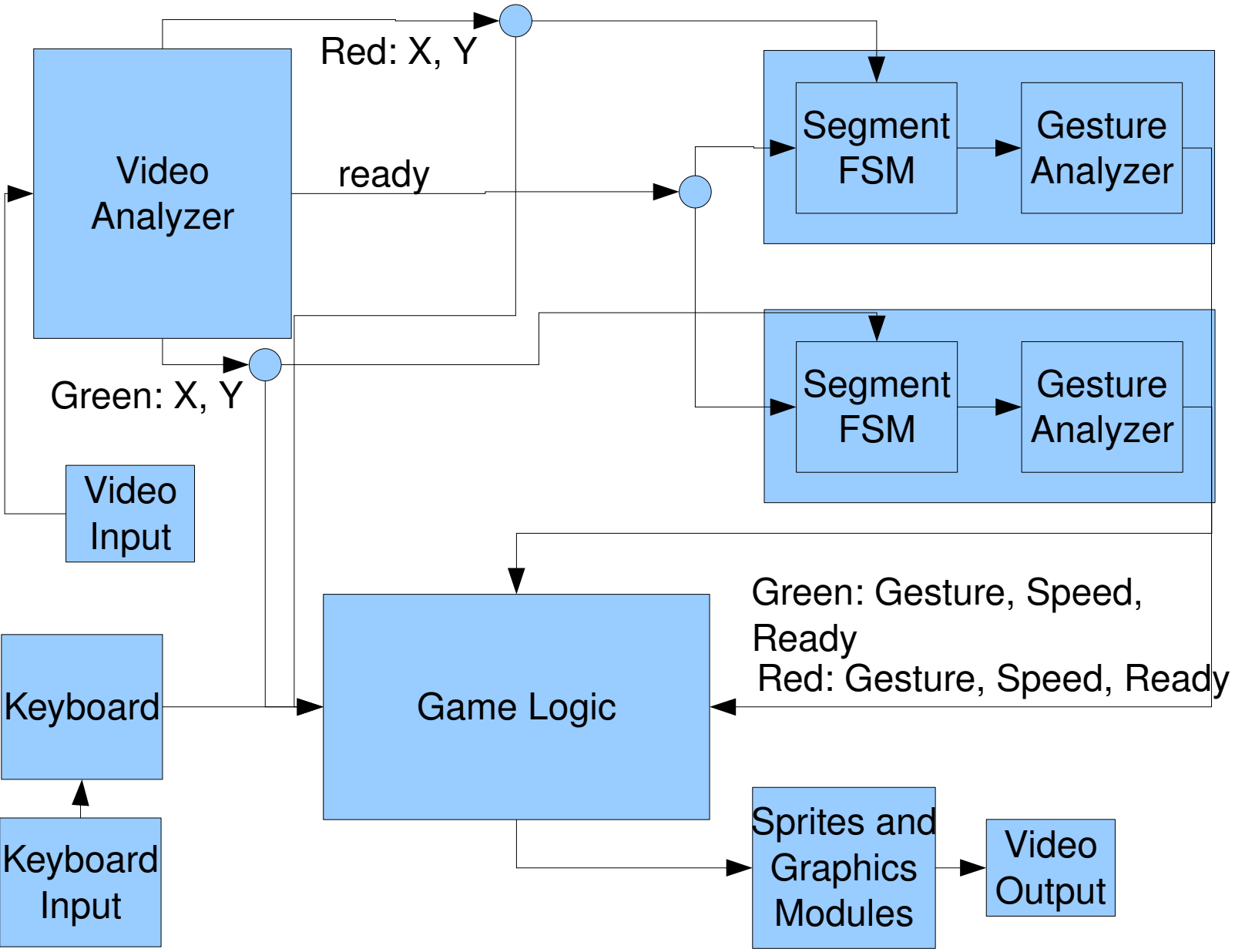
The graphics of the game will be implemented through sprites. We will start with simples sprites so we can start working on the game logic at the same time, but will simultaneously develop more complicated sprites. The current plan is to have the good guy be Kirby and the bad guy be King Dedede. There will also be some sprites that describe the level, and sprites that describe projectiles. We have not fully decided on the content of the game so we don't exactly know how many sprites there will be. We will try to make some fairly generic sprite modules that use pictures stored in memory or simple color blobs. We will likely chain the modules together to determine the ultimate color of each pixel and we will have to pipeline this as a result. Daniel Gerber will program all of the modules that display the sprites.

In addition to the labkit, we will need a red laser pointer (which we think we already have), a green laser pointer (which we will likely need help finding) and some plastic to put on the screen to reflect the laser dots. The plastic should be fairly cheap. If we can find someone to borrow the green laser pointer from that would be best because those tend to be fairly pricey.

In conclusion, our game will be loads of fun to play and to create and it will blow Harley and Rajeev's project out of the water!

The twelve possible basic segments recognized by the gesture recognition FSM




Red: X, Y
Video Analyzer
ready
Segment FSM
Gesture Analyzer
Segment FSM
Gesture Analyzer
Green: X, Y
Video Input
Green: Gesture, Speed, Ready
Red: Gesture, Speed, Ready
Keyboard
Game Logic
Keyboard Input
Sprites and Graphics Modules
Video Output