Super FPGA Bros

A novel input for a classic game.

1 Introduction

Our proposed digital system is a hardware implementation of the classic video game, Super Mario Bros that also utilizes a novel user input system. A single player will control the game using a combination of body movements and gestures which will be captured by both a worn pedometer and an external video camera. The user will wear a hat that will enable the camera to identify the user's current stance and facing direction. The combination of walking speed and motion capture data will be used to control the player's in-game character as it traverses the map. To implement this system, we will be using several high level modules that will be responsible for different aspects of the video game. A high level input module will take the combined user data and prepare it for use for the main game logic. A separate output module will handle the generation of video and audio outputs.

2 Input Modules (Kevin)

2.1 Video Capture Logic

The video capture logic takes as input data from an external camera's NTSC signal. The camera will track the position of a marker on the user's body (currently proposed as a reflective ball on a baseball cap). The marker should be sufficiently bright or different from the background to allow the webcam to track it. The image will be divided into several sectors, to determine whether the player is facing to the left or right, and whether the player is jumping, standing, or crouching. To determine whether the person is facing to the left or right, there will be a vertical line in the middle of the frame to split it into left and right sides. If the ball is in the right side of the frame, then the player must be facing to the right and the same holds true with the left side. To determine if the player is jumping, standing, or crouching, there will be two horizontal lines. One line will be above the player's head and the other below the player's head. If the player jumps, the ball wil go above the top line and the game will recognize this as the player jumping. If the player crouches, the ball will go below the bottom line and the game will recognize this as the player crouching. When the player is neither jumping nor crouching, the ball should be between the two lines meaning they are just standing or walking. The lines will be displayed on a screen overlapping the video of the player so that the player can see their position relative to the lines. A separate logic will take this information and compare it to the static background to track the movement of the ball within these sectors, and signals for left, right, jump, crouch, and standing will be output.

Video capture will also be used to determine if the player is standing still, walking, or running. On each arm the player will have a different color marker that will be tracked. The position of the marker on the player's arm will be saved in registers which will be used to calculate the frequency or period of the arm's swinging motion. There will be a threshold value in the logic to determine if the player is merely walking or if they are running. The logic will output signals will be walk, run, or no movement.

If time permits, we will add two additional markers, one for each leg. These markers would be used to track the position of the player's legs. As an alternative to looking at the swinging of the player's arms, we could track the position of the player's legs in order to determine whether they are standing still, walking, or running. The real purpose of adding the two leg markers would be for tracking the movement of all four limbs to be replicated in the game in real time. Instead of using the game sprites, a wire figure or character would be made to replicate the player. The position and motion of both arms and both legs would be used in a separate character drawing module to draw the figure in the game to model the same motions the player is making in real time. This module would now have to also output the coordinates for the markers on each arm and leg since they will be needed by the character drawing module.

3 Game Logic (Douglas)

The main game logic consists of several modules that take input control signals from the video controller and determine how the player character and game world interact. The game logic also outputs a data bus to the graphics subsystem to display the player and game world on the VGA output.

Game FSM Module

The main logic behind the game is provided through a simple FSM. It will take player input commands as well as video module information to determine how the various entities within the game interact.

The In Game World

The world that the player interacts with is comprised of both a static background as well as a dynamic foreground. The dynamic foreground is itself comprised of various tiles that make up the level the player can interact with. Among them are simple platform tiles that can be connected together to provide a surface on which the player character runs and moves, enemy character tiles that present obstacles that the player character must avoid or defeat, and hazard tiles that provide static obstacles for the character to avoid as he traverses the level. To implement this, the location of each requested tile will be stored in a memory array that will be read from at runtime. The coordinates of each object are provided to the game's graphics subsystem, and the actual sprites will be drawn on demand.

Player & Environment Interactions

In navigating the game world, the player character can either move forward (to the right), move backward, or jump. Moving backwards or forwards causes the gameworld to scroll around the player character. Jumping is a little more complicated, as we must take into account momentum provided by forward or backward motion. In general when the player jumps, the character will move in an arc through the air until it makes contact with another object in the game world. If the player reaches the end of the level before the time limit, he can touch a flag that triggers a completion signal. Hit detection is handled by checking the current location of two sprites and checking for overlap. A player can defeat an enemy if he approaches it from the top, otherwise the player loses a life and the level restarts.

If additional time permits, extra levels can be implemented by creating additional memories to store level layout. A submodule can also keep track of the time elapsed until the player completes the level, as well as keep track of score based on certain in-game events (e.g. 100 points scored each time the player successfully defeats an enemy) Finally, a head-to-head competitive mode is proposed in which two players each on separate labkits can race against each other to go through the level. In addition, a translucent "ghost" character can be displayed on the opponent's display, enabling real time feedback as to the relative progress of each player. This can be implemented by designing our game with the ability to render two player characters, and taking opponent data from the second labkit through a custom interface.

4 Output Modules

4.1 Video Output Logic (Douglas)

Our video output will be a sprite based graphics implementation. Initial testing of the game logic will be performed with simple placeholder sprites. The video output logic will be controlled by a direction signal and the pedometer rate inputs. The combination of these two signals will determine the direction of horizontal scrolling for the game’s map. To display images we will use a frame buffer implementation to get around the timing constraints imposed by the 65MHz video clock.

The original NES implementation of Super Mario Bros used a screen resolution of 240x256 pixels. Our frame buffer will use the same resolution for internal logic, but scale each pixel when writing to the VGA output. By representing each internal pixel as 4x3 real pixels, we will be able to scale to a resolution of 1024x720.

4.2 Audio Output Module (Kevin)

The audio output modules will generate sound effects and music to be played on an external speaker. Sound effects will be generated based on event signals provided by the game logic, while a looping background music will be stopped, started, or restarted based on a status signal generated by the game. The background music as well as the different sound effects for jumping, killing an enemy, and dying will be stored in memory. We will use the AC97 codec chip to handle our sound data to generate our analog output audio.