Flying Pegasus Ground Attack

6.111 Final Project Proposal

Yini Qi | Tania Yu

1 Overview

Robot Unicorn Attack is an "endless running" flash game in which the user controls the movement of a unicorn through space. The object of the game is to prolong gameplay without falling off the stage, crashing into platforms, or colliding with obstacles. In the original game, movements are controlled by keyboard inputs. This project aims to bring the classic game to life using a camera to detect a player's motion to perform the corresponding actions in the game. Thus, the need for keyboard inputs is removed and the game relies solely on motion tracking. The player flaps her arms up and down to fly a Pegasus sprite, and the speed at which the flapping motion is performed controls its height. Additionally, the player can put her hands together in the center of her body to perform an "attacking" action on the obstacles to destroy them. These movements are predetermined and are the only recognized movements in the game. The player wears a red glove on the right hand and green glove on the left hand to help the camera differentiate between the different hands. The camera data is sent to the game logic in the FPGA on the Nexys 4 board to determine the state of the game and control the movement of the Pegasus. The basic implementation will include sprite images of the Pegasus and obstacles along with sound effects, including the traditional background music of Robot Unicorn Attack (Always by Erasure). The stretch goals include dynamically changing background images and complex obstacle movement.

2 Design

The system consists of four major components: motion recognition, graphics rendering, game logic, and sound effects.

When a player turns on the FPGA, a game opening sequence displays on the monitor. Red and green gloves will be purchased to facilitate hand detection. Since the 6.111 lab is mainly colored blue, red and green would stand out from the background and be easily recognizable in the camera. Once the game starts, the motion recognition module filters in the player's gloves from the video, and determines the coordinates of the center of mass of each glove. Depending on the type of motion detected after a series of frames, the module sends the information of the state (flying or attacking) to the game logic. If the player is flying, the module calculates the speed at

which the player flaps each gloved hand and averages them. If the player is attacking, the module returns the attack signal to the game logic.

All sprite graphics, including ones for the Pegasus and obstacles, will be loaded into memory at the start of the game. During gameplay, these will be loaded from the appropriate memory blocks and painted pixel by pixel on each frame. In the basic implementation, sprites representing ground blocks and any other dynamically generated images in the background will also be pre-loaded in this fashion, but as one of the stretch goals, a large and complex background image will be loaded from SD card for display during the game.

The game logic also controls the interactions between the Pegasus and the game environment. Sound effects will also be added during the attacks and object destruction. Sound effects of attacking and collisions from the original game will be recorded and added. Of course, no Robot Unicorn Game is complete without the theme song "Always" by Erasure, so the song will be played in the background continuously on loop.

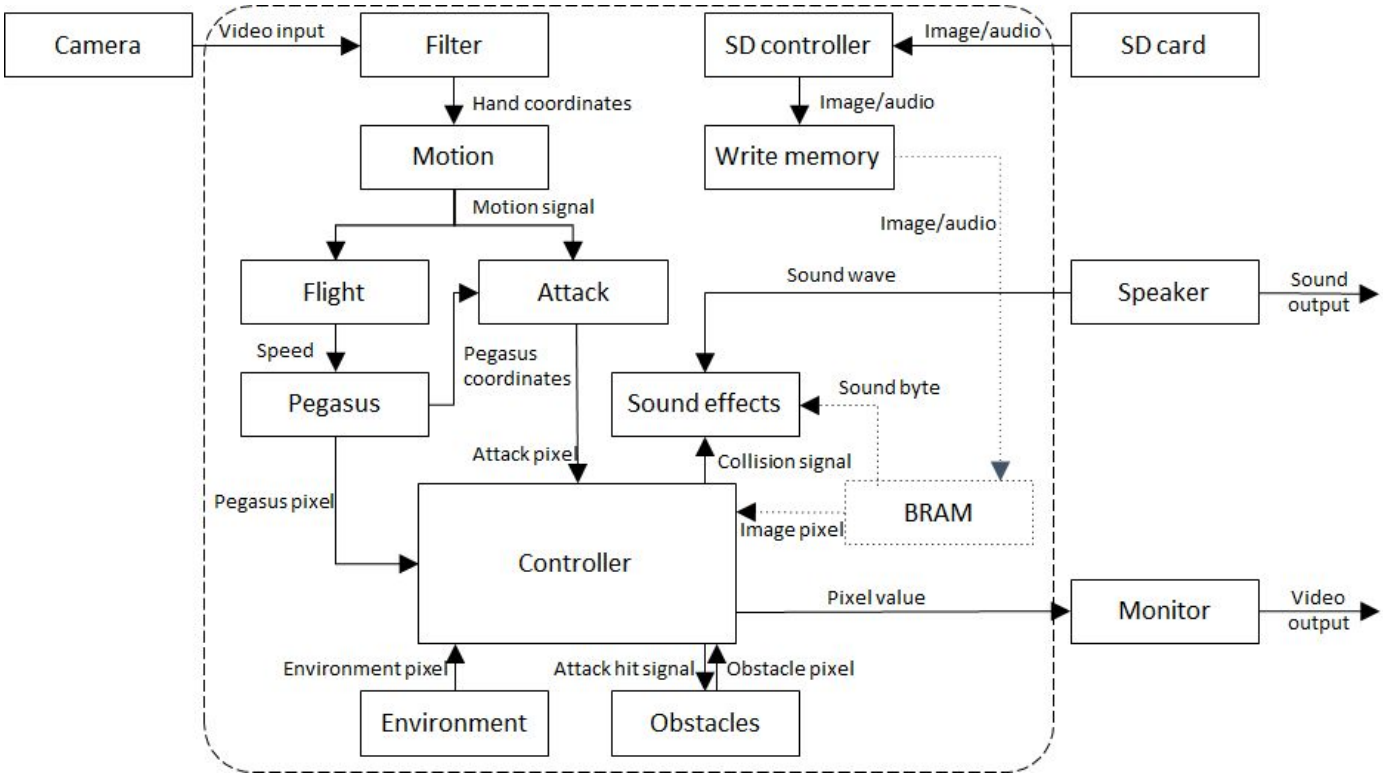

Figure 1: Full system diagram

3 Implementation

3.1 Motion Tracking

The video stream sent in from the camera will be stored one frame at a time in the Nexys 4's BRAM. In order to fit the entire frame into BRAM, the frame will be downscaled from the camera's native 640x480p resolution to 320x240p due to memory constraints. The Nexys 4 has 4860 Kbits of BRAM in total, so there will not be enough space to store the sound, background image, and video frame simultaneously without downscaling. The filter module takes the stored video frame from BRAM and filters the positions of the user's hands based on where the different colored gloves are located. One challenge is being able isolate the glove pixels from each frame and finding the center of mass of each glove. Then, the filter module sends the hand coordinates to the motion module where the type of motion -- either flying or attacking -- is determined. The motion module must store the hand coordinates over many frames in order to determine the speed if the arms are flapping. Another challenge here is calculating the physics behind how speed of hand motions correspond to the actual sprite movement speed. If the motion is flying, then the motion module sends the speed and on signal to the flight module. If there is no flying motion or the motion module calculates a speed below a given threshold, then motion sends a zero signal to flight which will defer to a predetermined falling speed. If the motion is attack, the on signal will be sent to the attack module which coordinates the attack patterns.

3.2 Pegasus

The Pegasus module controls the location and movement of the sprite. Because the background and surrounding objects scroll sideways instead, the Pegasus does not actually move in the x-direction on the screen. The sprite is therefore represented here by a rectangle with predetermined size, with location marked by the top-left corner (the x-coordinate is set and does not change). This module uses the speed output from the flight module to determine the sprite's updated y-coordinate on the screen. The flight speed will be represented as pixels per frame.

3.2.1 Flight

Flight mode is triggered when the player's hands are far apart such that the difference in x coordinates between each glove is greater than some threshold. The flight module calculates the flight speed based on the change in right hand's y coordinate over many frames. Only one hand's coordinate information is necessary because the player is assumed to flap both arms simultaneously, which makes calculating two hands redundant. The choice of calculating the right hand is arbitrary. Since the camera records at 30 frames per second, the speed will be updated every 10 frames, as that is a reasonable estimate of the upper limit of how fast a person can flap her arms. A lower threshold is kept in case the player flaps too slowly, in which case the speed is calculated as zero and the flight module defaults to the falling speed.

3.2.2 Attack

Attack mode is triggered when the player puts her hands together in close enough proximity, such that the difference in x coordinates between each glove is less than some threshold. When the conditions are met, the Pegasus fires a beam directly in front of it, destroying an obstacle if one exists. When attack is triggered, the flying ceases and the Pegasus stays in place for the duration of the attack. The duration is a set number of frames based on the distance from the Pegasus to the obstacle, which determines how long it takes from the attack beam to reach the obstacle. The attack beam will not be triggered again until the player separates her hands and puts them together again.

3.3 Graphics

Prior to the start of the game, sprite images will be pre-loaded onto the BRAM. Each sprite will be allocated to a separate block so that each sprite module (i.e. Pegasus, obstacles, environment) will load pixel values from its appropriate block. The corresponding address in memory will be selected by calculating the relative location of the current (x,y) pixel coordinates to the top-left corner coordinates of the sprite.

One challenging stretch goal for graphics rendering is displaying a complex background image that scrolls horizontally to create an impression of sideways motion. Because the image would be large enough to last for 30 seconds of gameplay, it would be 19200x480 pixels, which is much too large to store in BRAM at once. Instead, the whole image will be saved on SD card, and only 640x480 pixels in BRAM at any given point. On each new frame, the leftmost 20 columns of pixels will be removed from BRAM and the next 20 columns loaded in. This creates the effect of shifting the whole background image to the left.

3.4 Sound

The sound module plays background music from the original game and additional effects when obstacles are hit or destroyed. At the start of the game, the background music is loaded from the SD card to CelluarRAM and played continuously on a loop.

3.5 Logic

The controller module detects collisions between Pegasus and objects in the environment, including ground blocks and obstacles above ground. Ground blocks are generated such that there are set gaps between them where the Pegasus sprite can fall through. If the sprite hits the bottom of the screen or the side wall of any ground block, it dies and the game ends. Obstacles are also generated at specified locations, but only one or two at most are displayed at a given time. The obstacles module uses a random number generator to decide which obstacles are

turned “on” or “off”. If an obstacle is on, it is visible on the screen and the Pegasus can either attack and destroy it or collide with it. If it is off, it is as if the obstacle does not exist and has no impact on the gameplay. Destruction of an obstacle also triggers the controller to signal the sound effects module.

3.6 Stretch Goals

1. The obstacles will move around the environment, making them more difficult to dodge and destroy
2. Import actual graphics from the Robot Unicorn Attack game to make the environment more beautiful, including dynamically updating backgrounds
3. Add coins which the player can fly over to gain points
4. Incorporate different characters with varying stats and attacks

4 Timeline and Testing

The timeline is divided as follows in Figure 2, with blue representing Kelly, yellow for Tania, and green for both. In terms of work distribution, Tania will begin by working on the graphics rendering of the Pegasus and environment. Kelly will be in charge of filtering the motion from the camera. The one who finishes first can start working on the physics calculations for the wing speed. After these two parts are integrated, we will work on the game logic module and divide the work between environment effects and sound effects. Each member is responsible for testing her individual modules so that they work as standalone components. Each module will have test benches if it is testable by simulation.

After the initial integration of the motion and Pegasus modules, the system will be tested to ensure minimal functionality. This is expected to be the most troublesome stage as individual modules may have to be modified to fit with the other modules.

Finally, the team will focus on intensive debugging for the one week after the system is completely assembled. Stretch goals will be implemented as time allows.

Task \ Week	11/1	11/8	11/15	11/22	11/29
Motion tracking					
Basic graphics					
Flight physics					
Integration and testing of basics					
Attacking obstacles					
Collision detection/death					
Integration and testing					
More graphics					
Sound effects and music					
Final integration and testing					
Stretch goals					

Figure 2: Gantt chart of the proposed schedule

5 Resources

The materials needed are two different colored gloves, a camera, SD card, Nexys 4, and speakers. For testing purposes, more colors of gloves will be ordered in case the red and green are not easily recognizable. The camera, SD card and Nexys 4 are available in the lab, and the gloves can be easily purchased for $15-20 each.

6 Conclusion

Our project aims to create a version of the popular Robot Unicorn Attack game that can be played with only hand motions, eliminating the need for keyboard inputs. We aim to support two types of motions, flying by flapping one's arms, and attacking by putting one's arms close together. Finally, we would like to incorporate sound effects to enhance the player's experience.

We expect the most challenging component to be motion tracking. Interfacing with the camera will take a significant portion of time as we have to configure it and figure out how to filter the inputs so that only the relevant hand images are stored.

The Flying Pegasus Ground Attack game is a project which will be fun and interactive. We have played the online game extensively, and we look forward to bringing it to life with FPGA and motion tracking technologies.