A novel implementation of a portable Dance Dance Revolution game

Gabriel Ha, Daniel Kim
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139

Introduction

Dance Dance Revolution (DDR) is a pioneering music video game focused on rhythm and dance. Released in 1999 in North America, DDR has become a well-known staple of arcades and home gaming systems. Players stand on a dance platform or pad with four arrows (pointed front, right, back, and left). The goal is to hit these arrows with their feet according to visual and musical cues. Players are judged by how well they time their steps to the patterns displayed on the video screen and receive a score at the end of the song. Because of the popularity and appeal of such games, we are interested in designing a new implementation that is novel in its adaptability and portability.

This project offers two major advantages over the original. First, the game removes the necessity of the DDR pad and uses cameras to detect the player's foot movement on the ground instead. This provides a freer user experience and will hopefully eliminate the frustrations of the DDR pad not recognizing intended movements by the player. Second, the game allows the player to upload and dance to any song. The game features two levels of difficulty: Easy and Standard. Standard DDR rules, graphics, and scoring are emulated in the project, including bonus points for player's step-time accuracy, and additional bonus points for consecutive good step-timing.

Functionality

The game can be broken down into four basic tasks: music and arrow encoding, motion processing, user interface, and game logic.

As our implementation is intended to be versatile in song choice and difficulty, our functionality requires music and arrow encoding from an outside source. As such, the user will be able to read in any mp3 file, and the music processor will be able to output a series of dance commands that are unique to each song. Thus, our implementation will utilize amplitude and frequency analysis of the inputted song to output appropriate dance patterns for the user's game.

Additionally, the game must be able to understand the motions of the game player to ascertain whether the player has actually performed the correct move. As such, we will design a motion processing module that can take the input from two web cameras and analyze the location of the game player's feet. This will then be put into the game logic to score the player.

The user interface allows the game player to make choices in the game and set parameters. With the user interface, the user should be able to choose the difficulty level, start the game, calibrate the system, and reset the game.

The game logic will detail all the features and parts of the game, from the start screen to the completion and scoring of the game. This part of the game will take inputs from the music and arrow encoding, the motion processing, and user interface to output the appropriate actions and results based on the game's rules.

Implementation

The block diagram associated with this proposal is composed of 9 modules and/or component representations. We will discuss implementation details of these modules to a depth we believe to be sufficient for our proposal. Non-clocked components are indicated by the abbreviation NC in parentheses. The person who will oversee and complete most of the work for the module is also indicated in parentheses by last name.

Self-Explanatory Components (Kim)

mp3 uploader

This module takes in the mp3 input, mostly likely via USB, and processes the file into useable information.

ZBT (RAM) (NC)

The ZBT stores the music from the mp3 uploader and the associated arrows that are processed in the Music-to-Arrows Algorithm Processing module.

Video Output

The video output module takes in a number of inputs from the Game Logic and Synchronizer module and colors the pixels appropriately to be displayed on the monitor.

Audio Output

The audio output module takes in the music from the Game Logic and Synchonizer module to be synced with the dance arrows.

Speakers (NC)

The speakers play music and receive input from the Audio Output module.

Module Components

Player Motion Processing (Ha)

This module takes inputs from the external cameras to determine which arrow directions the player is intending to communicate by his steps. The module outputs four bits for each arrow direction (up, down, left, and right) indicating a 1 if that direction was stepped on, and a 0 otherwise.

Determining whether or not a player has made a step is facilitated by some sort of marking on the player's foot, possibly colored tape. The two cameras are placed diagonally with respect to the player and the virtual grid, as illustrated below:

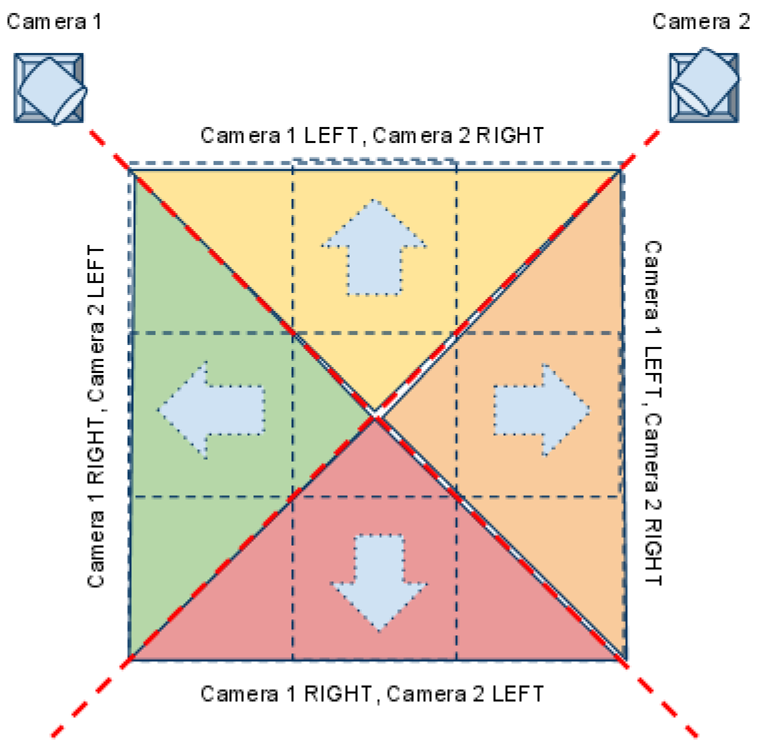
Camera 1
Camera 2
Camera 1 LEFT, Camera 2 RIGHT
Camera 1 RIGHT, Camera 2 LEFT
Camera 1 LEFT, Camera 2 RIGHT
Camera 1 RIGHT, Camera 2 LEFT

Figure 1: Camera setup to facilitate a virtual DDR pad. Note that the lines in the diagram are dotted, indicating that this grid is not actually seen on the ground.

It can be seen that each direction represents a distinct combination of LEFT or RIGHT of the two cameras. A step is then determined by the presence of a foot 1) on the ground, and 2) "enough" to the right or left of each camera to determine the foot's position as being not in the center of the virtual grid. There will be a calibration process, accessible from the game menu, that allows the module to determine the cameras' distance thresholds from "center" and from the ground.

If a person's foot remains on the ground in an arrow area, it is not to be counted again as a step; the person must lift his foot and place it down again.

Music-to-Arrows Algorithm Processing (Ha)

The music to arrows algorithm processing module takes in data from the mp3 uploader and analyzes the sound to assign a sequence of arrows to the music. On the most basic level, this algorithm will look at amplitude and use threshold values to determine whether the peak should be considered a "significant" peak. If the peak is considered significant, an arrow will be assigned to that peak and stored into the ZBT RAM with the sound at that peak. On a higher level, this algorithm will also look at the frequency domain and analyze high frequency and low frequency components to determine key music lines, such as the melody and the bass line. Using this analysis, the algorithm can then assign certain arrows to the different frequency components.

Debouncer (NC) (Kim)

The debouncer module takes asynchronous inputs from the user interface (such as the "Enter" button and the directional buttons) and turns them into synchronous outputs for use by the game logic module.

Game Logic and Synchronizer (Kim)

The game logic module takes inputs from the player motion processing module, the music-to-arrows algorithm processing module, the ZBT RAM, and the user controls and synthesizes them according to the game rules. The visual outputs are then sent to the display while the audio outputs are sent to speakers. The game logic can be in three main states: game set up, game play, and game ended.

Game Set Up

In this state, the game has not yet started, and the different components have not yet been put into a game-ready state. As such, the user must first input an mp3 file to be analyzed by the music-to-arrows algorithm processing module and put into the ZBT RAM. In this process, the user must also indicate whether the level of difficulty should be standard or difficult. Additionally, the cameras must be calibrated to the user's desired steps. After these setup tasks have been completed, the game can then be played.

Game Play

In game play, the module follows the typical game play of Dance Dance Revolution. The module reads the music and corresponding arrows from the ZBT RAM and displays them on the monitor. These arrows are read in advance, so that they can be displayed moving up the screen. When the arrows reach a certain point on the screen, the user must step in the direction of the arrow. The motion processing module reads that movement and sends the step direction to this module, where the timing and step direction are compared to the desired values to determine accuracy and scoring. Additional functionality may include special scoring for a series of well-timed steps and accuracy. Game play mode ends when the song has been completed, or when the game player resets the game.

Game End

When the music has finished, the game is over. This module then outputs the score to the screen and freezes game play. A new game can be started by either restarting the current song, or resetting the game entirely to input a new song.