

UCSB
CS190B - F23

Final Project

Team Members:

First and last names	Perm Number
Justin Chung	5768361
Anushka Lodha	8531170
Amey Walimbe	8594210

Report

Problem Overview

Our project is called Dance Till You Pi - a play on the phrase, "Dance till you die". We were inspired to make our own version of "Dance Dance Revolution," or DDR, a game we love playing in arcades. DDR normally involves a dance platform, colorful arrows, and music. A colored light flashes on every beat, and the player(s) must step on the same colored arrow at the right time. This game favors those with good coordination, reaction time, and a musical ear. Dance Till You Pi incorporates all these DDR features into a small design, and is meant to be played with just a single hand. While typical DDR games are large and expensive, our version is much more compact and can be enjoyed from the comfort of your room!

Methods and Solution

In the initial phases of our project, our team aspired to create a comprehensive dance platform within the Internet of Things domain. However, we quickly realized the potential challenges and the ambitious nature of such a goal, especially for us beginners. Thus, we decided to streamline our focus towards developing a scaled-down, finger-playable version of the game.



Our interactive MVP incorporates colorful buttons, an LED light strip, and a speaker, all integrated with a Raspberry Pi. The game involves user interaction with the colorful buttons, synchronization with the LED light strip, and near latency-free audio playback through the speaker. More details about these are listed in the [Materials](#) section below.

Upon completion of the game, a unique scoring system, as described in the [Score Calculation](#) section below, computes the player's performance. The resulting score is then uploaded to a database for display on our leaderboard, as described in further detail in the [Displaying Results](#) section below.

Materials

- Raspberry Pi 3, breadboard, wires
 - Hosted the game logic and connected to all user peripherals
 - Loaned out from 190B staff (thank you!)
- 5V LED light strips
 - One end had a 4 pin connection (5V, R, G, B) to connect to the GPIO pins on the RPi
 - [Purchased from Amazon](#)
- Tactile buttons
 - Red, yellow, green, and blue options
 - Purchased from Amazon
- JBL speaker
 - Output game music (*Blinding Lights* by The Weeknd in our demos)
 - Connected to RPi through 3.5mm audio jack + cable
 - Used Pygame module to receive music from game logic
 - Amey's personal speaker

Circuit Design

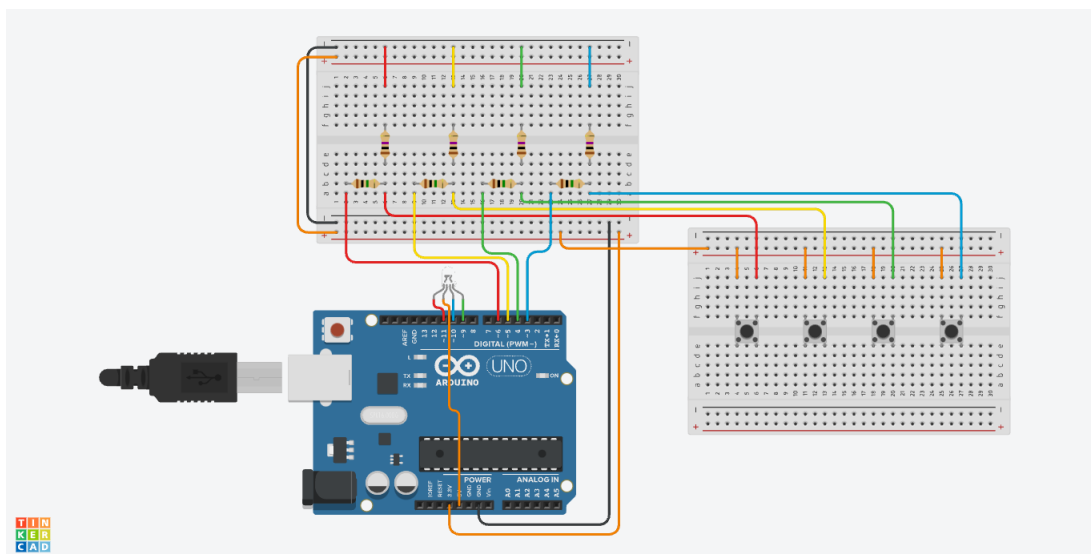


Image 1: Our circuit design connecting our materials listed above

Our design used a Raspberry Pi in place of the Arduino shown in the circuit diagram above, with the wires connecting to GPIO pins on the RPi instead of the digital ports on the Arduino. Not shown in the diagram is the speaker that we connected to the RPi through its 3.5mm audio output port. The buttons on the breadboard to the right represent the inputs for the red, yellow, green, and blue signals (from left to right).

Score Calculation

Initially, we created an **accuracy-based scoring system**. The player would receive 1 point for pressing the correct button in the correct time interval, and 0 points if not. The score would be the points received divided by the total points possible (essentially, the total number of light flashes displayed). However, a perfect score was too easily achievable, so we decided to take the player's speed and reaction time into account for score calculation.

Next, we implemented a **weighted reaction time-based scoring system**. This approach assigned 0.8 points for each correctly matching first input for each light flash and incorporated an additional component, a fraction of 0.2 points, scaled to the reaction time for that input. The cumulative score was then divided by the total number of light flashes to determine the final score.

In the subsequent phase of our project, we **transitioned to a rhythm-centric scoring model**. In this revised system, players receive higher scores for accurately pressing buttons in sync with the beat of the song. The scoring calculation involves awarding a point for each light flash matched with a correct first button input. Additionally, points are deducted based on the time difference between the downbeat and the button input. The resulting score is then normalized to a scale of 100, representing a percentage, by dividing the total points by the total number of light flashes.

Inspired by point systems in games like *osu! mania*, our approach distinguishes itself by emphasizing the importance of precision in button selection over multiple inputs. Notably, the system places significant weight on the accuracy of the first correct button press, disregarding subsequent inputs for a particular light flash. This design choice underscores the significance of selecting the correct colored button over the mere act of pressing any button.

Music

We used the **pygame** module to add music functionality to our project. Pygame is a free, open-resource python library that many game developers use to add rich graphics and sound. For our purposes, we downloaded an mp3 file of the song we wanted (*Blinding Lights* by The Weeknd), and then used library functions to have the music play on our JBL speaker which was hooked up to our Pi.

Displaying Results

Your Score

Name	Score
j	90.394

Top 5 Items (Sorted by Score)

Rank	Name	Score
1	j	92.6
2	j	90.394
3	AL	90.37
4	Amey	89.873
5	zz	89.724

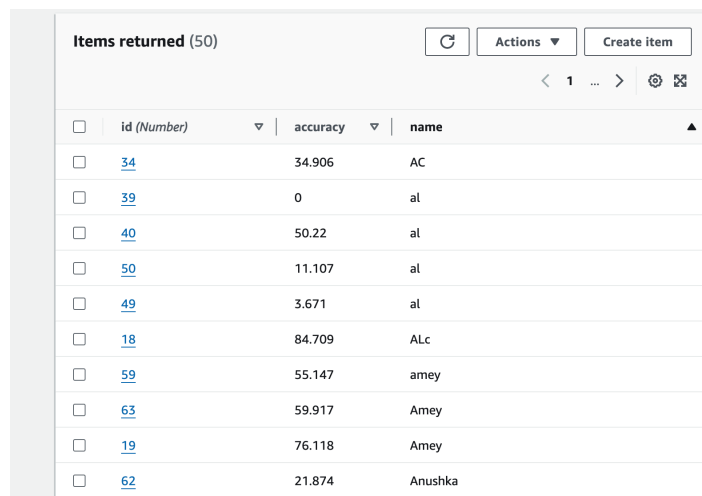
Image 2: Our Display Page

After playing the game, we pushed the player's score via the **python boto3 library** to store the results in an AWS DynamoDB table. This was convenient, because we could easily add columns when we wanted to (for example, to include "player name" as a column, rather than just "player id" and "player score").

Locally, we created a separate python app.py program, along with an HTML file. Together, they would work to pull data from the AWS DynamoDB table to display the results.

Obviously, we wanted to show the most recent score. Since we set up the id value to increment by one after every game, we could easily access the most recent score by grabbing the largest id value.

We also wanted to add some competitiveness to the game, so we decided to display a leaderboard of the top 5 high scores along with the corresponding player's name (the player name was retrieved by user input before the game began). We thought about having a larger leaderboard, but decided against it as it would take up a bunch of space on the screen.



Items returned (50)		
id (Number)	accuracy	name
34	34.906	AC
39	0	al
40	50.22	al
50	11.107	al
49	3.671	al
18	84.709	ALc
59	55.147	amey
63	59.917	Amey
19	76.118	Amey
62	21.874	Anushka

Image 3: Contents of our dynamoDB scores table

Results and Findings

Results

- We successfully created a finger-style version of DDR!
 - We were able to integrate our Raspberry Pi with LED strips and tactile buttons
- Each of us learned many new skills. To name a few...
 - We played around with the breadboard, and learned how resistors can limit current going to circuit components which can't handle higher current levels.
 - This was the first time making an IoT project for each of us. We were able to apply our findings from Lab4 to store our game results in a DynamoDB database, then make queries on this database to display the five highest scores.

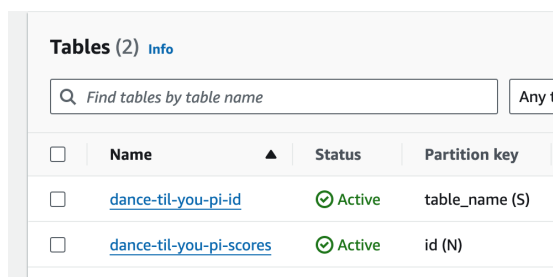
Findings

- Our team found out that monitoring the timestamps of the lights flashing and the buttons flashing actually worked. The latency of the buttons and the lights were not too high, thus we could accurately calculate how many flashing lights (and/or song beats) the player missed, and by how much time.
- We realized we should've better understood how we would use a project part before we purchased it. For example, we should've determined whether the LED strips were compatible with GPIO pins and/or USB cables before we bought them. As Shereen actually mentioned to us, we should've found code snippets or tutorials online prior to buying anything, just to make sure it was even possible to control those strips through the Pi.
- It was often difficult for us to determine whether our issues were due to faulty software or hardware. Many times our code worked just fine, but we had a buggy Raspberry Pi that kept disconnecting from power. Other times, our code caused disconnection from the Pi user. Simply gaining more practice working with hardware would probably help us distinguish the types of errors more quickly.
- We learned that we don't necessarily need huge code changes to add huge value to our product. For our initial method of scoring, we only assessed whether the player clicked the correct button in the correct interval; thus, any player who had a 100% accuracy would get a score of 100. After a while of testing this, all three of us became experts at this and found the game boring. Pretty quickly, we were able to adjust our code to also take into account the players' reaction times; with testing accuracy *and* speed, our game became much more interesting, and the competitive nature of our game was restored.

Challenges

- **Connecting the LED strip.** We weren't sure if we could control the LED light strip we purchased with the Raspberry Pi. At first, we tried connecting using the USB port. Pretty soon, we realized we'd have to switch over to using GPIO pins. After some debugging, we were able to figure out how to make the lights flash RYGB colors at random intervals.

- **Using the breadboard and buttons.** We didn't realize we needed resistors, so our Pi kept shutting down every time we ran our Python script.
- **Adding music to our project.** During the early research stages, we found that making the lights flash to music without manually adjusting it would be quite difficult. Thus, when we chose to add Blinding Lights by the Weeknd to our game, we had to figure out how to make the flashing lights and button presses fall on alternating beats of the song. While syncing the music, lights, and buttons was tedious, it definitely paid off.
- **Pushing a unique ID to our dynamoDb table.** We found out that dynamodb does not inherently support auto-generated keys/ids, so we needed to figure out a workaround in order to push our ID-score pair to the table. Our solution was to create a new table that stores just the ID value. Then, we increment the value by 1 each time the program runs. When the score is finally calculated, we pull from the ID table to get the newly incremented value, which we can then use to push our id-score pair to the scores table



Tables (2) Info

Find tables by table name Any t

<input type="checkbox"/>	Name	Status	Partition key
<input type="checkbox"/>	dance-til-you-pi-id	Active	table_name (S)
<input type="checkbox"/>	dance-til-you-pi-scores	Active	id (N)

Image 4: The tables we created for this project



Items returned (1)

Actions Create item

< 1 > ⚙️ ✖️

<input type="checkbox"/>	table_name (String)	id
<input type="checkbox"/>	dance-til-you-pi-scores	64

Image 5: The contents of our ID table

Future Work

Because of the limited time we had to design and implement this game, we couldn't add all the features we had envisioned. Here are some ideas we had for a more developed version of the project:

- We were initially inspired to go the DDR route after watching the video of the table-sized "Whac-a-mole" game that Professor Krintz added to one of her introduction-to-IoT lectures. The buttons of our MVP were pretty small and only able to be pressed with a finger; ideally we'd want to have bigger buttons, like the Whac-a-mole game. This way, players could hit the buttons with multiple fingers/ their entire palm, allowing for a more arcade-styled, yet still portable, game.

- The original DDR game allows multiple players to dance simultaneously! If we had more time and resources, we would have liked to implement a multiplayer game where the two players could compete against each other for the higher score.
- Our MVP hides the hardware from the player with a cardboard box. While sufficient for the purposes of a prototype, we'd want a more aesthetically pleasing case for the hardware for a more developed version of the game.