Flex Dance

Caio Araujo, Spandan Sharma, Tushhar Saha

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*— A system capable of Our system attempts to improve the at-home exercising experience, motivated by recent quarantine mandates. Current options include large gym machines, repetitive bodyweight routines, monotonous dumbbells, and expensive consoles that run exercising games. Our goal, as such, is for our device to be user friendly, easy to store, enjoyable, and comparatively cheap. The platform we provide consists of a pressure-sensitive mat that acts as a controller and a computer game that reads the user's steps on the mat to simulate dancing to a choreography. For maximum enjoyment, users can come up with and share their own choreographies to their favorite songs.**

***Index Terms*— Arduino, dance, design, exercise, force sensitive sensors, game, hardware, pygame, raspberry pi, software, user interface, mat**

I. Introduction

During the COVID-19 pandemic, many exercising spaces such as gyms and parks were closed. With people being required to stay indoors, it became clear that at-home exercising options were limited and cumbersome in some way or another. Exercise is important because it not only strengthens physical health, but also improves mental health, which saw general declines throughout the pandemic due to isolation, fear, and loss. Through this project, we aim to provide an alternative option for casual exercisers who want to stay fit indoors to preserve, or raise, their well-being.

Our platform is composed of two key components: a dance simulator game and a pressure sensitive mat. In the game, players follow along a choreography to a song of their choice by stepping on directional arrows laid out on the mat. This game is pre-installed in a RaspberryPi that the user simply connects to their TV and is then ready to play. We chose this design based on comparisons with other available options which are briefly described below and in further detail later in Related Work.

There are a few different ways someone can exercise at home. For example, one can purchase personal gym machines, but these prove to be expensive, hard to install and/or too large. Cheaper options include dumbbells, door-mounted pull-up bars, and bodyweight routines. However, all of these can be monotonous and can damage property. Finally, there are options more similar to what we propose such as the game series Just Dance. Our main advantage over Just Dance is that Just Dance requires a commercial video game console which costs over $200 in addition to the necessary motion-tracking equipment to play the game, but the user might not be interested in using this console for something other than exercising. Thus, our goal is to provide an easy to store, enjoyable, and comparatively cheap platform for casual at-home exercisers.

II. Use-Case Requirements

Based on our targeted users, the competitive products available in the market, and our goal to leverage technology to create a fun, engaging way of exercising, we have come up with a few use case requirements to guide us in our design process:

A. Affordability

We want our product to be cheap so that users of all financial backgrounds can avail its benefits. Current alternatives (whether metallic machines or mats that require a console to function) start from $ 300. To compete with these options, we want to be able to create our product well under $ 200.

B. Storing

To accommodate our users' diverse living conditions, we want our product to be easily storable. The mat should be foldable and fit in an average drawer of size 13 in x 12 in x 5.5 in and when the mat is expanded, it should fit comfortably in a living room space of 39 in x 39 in.

C. Accessibility

Our game should be able to be simply plugged into a display screen through an HDMI cable and run. This will make it easy for children or older people to play the game at their own ease.

D. Scoring

Players have a subconscious requirement of receiving scores if they get even partial overlap of arrows and do not time their steps correctly in the game. To account for this, we need to use a linear scoring scale that gives points to users based on the amount of accuracy of their stepping.

E. Interface

The interface has a direct point of contact with the user and must fit the following requirements:

1) Engaging: In order to facilitate a pleasing experience for the players, the interface should be stimulating but also not overwhelm the user. None of the information on the screen should be hard to process/absorb.
2) Beginner friendly: Our game should have a beginner friendly interface and it should be easy to start the game. The game screen thus should be a maximum of 3 clicks away from the start of the game.

F. Durability

The game should last a long time with little to no maintenance. Users would subconsciously expect the game to last for a couple of years in their house without needing any

outside intervention. Based on our calculations from research online, it should last approximately 650 sessions. The calculations are explained further in depth in design requirements.

III. Architecture and/or Principle of Operation

Our system comprises of 3 main components in order for it to work. As shown in Fig 1, they are the 'Game Mat', 'Raspberry Pi' and the 'Display Screen'.

The Game Mat is responsible for detecting inputs from the 6 force sensitive resistors (FSR). The relevant circuit has been shown in Fig 3. The mat also has an Arduino attached to it above the top middle square which acts as an ADC. It takes in signals from the FSRs and then, the relevant signals are relayed to the Raspberry Pi.

On the other hand, the Raspberry Pi contains everything needed to run the game including the game files, assets, an operating system and pygame. Fig 2 goes more into depth about the connections between game files. Finally, the display screen shows the graphical user interface and any corresponding sounds. It is important to note that the display screen is something that we expect the user to have. The rest of the system will be designed and built by us.

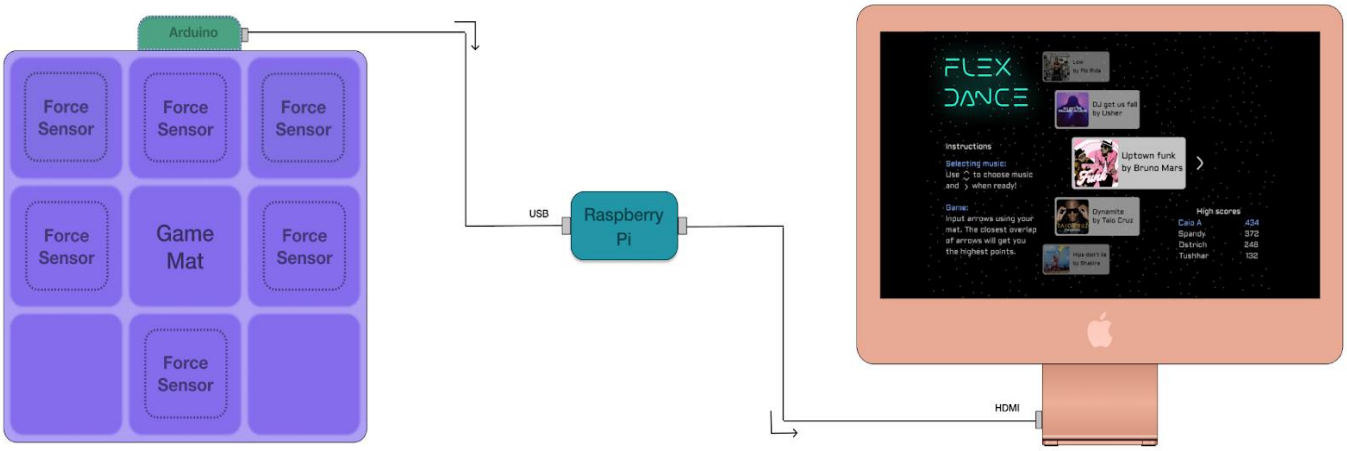

Figure 1: System block diagram

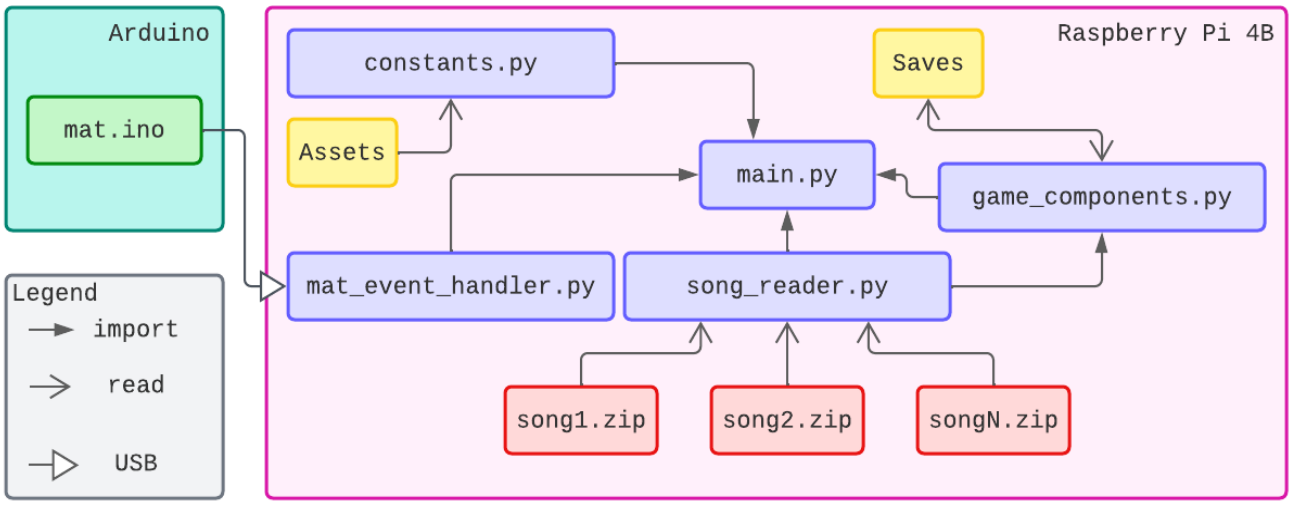

Figure 2: Block diagram for code files

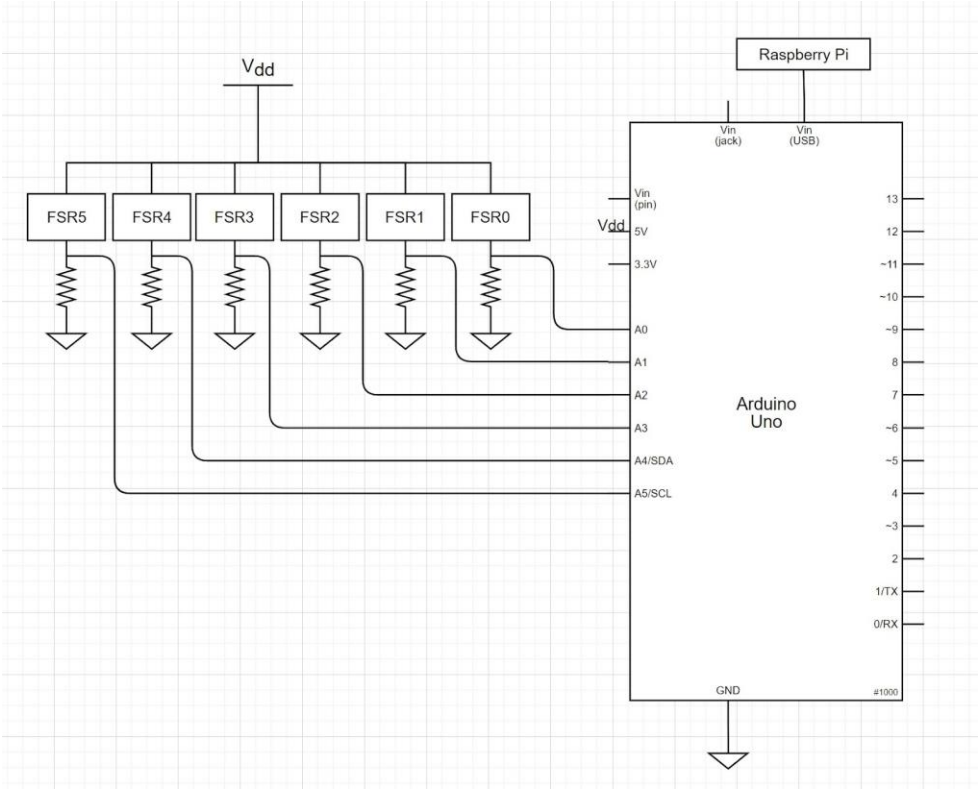

Figure 3: Circuit schematic for mat

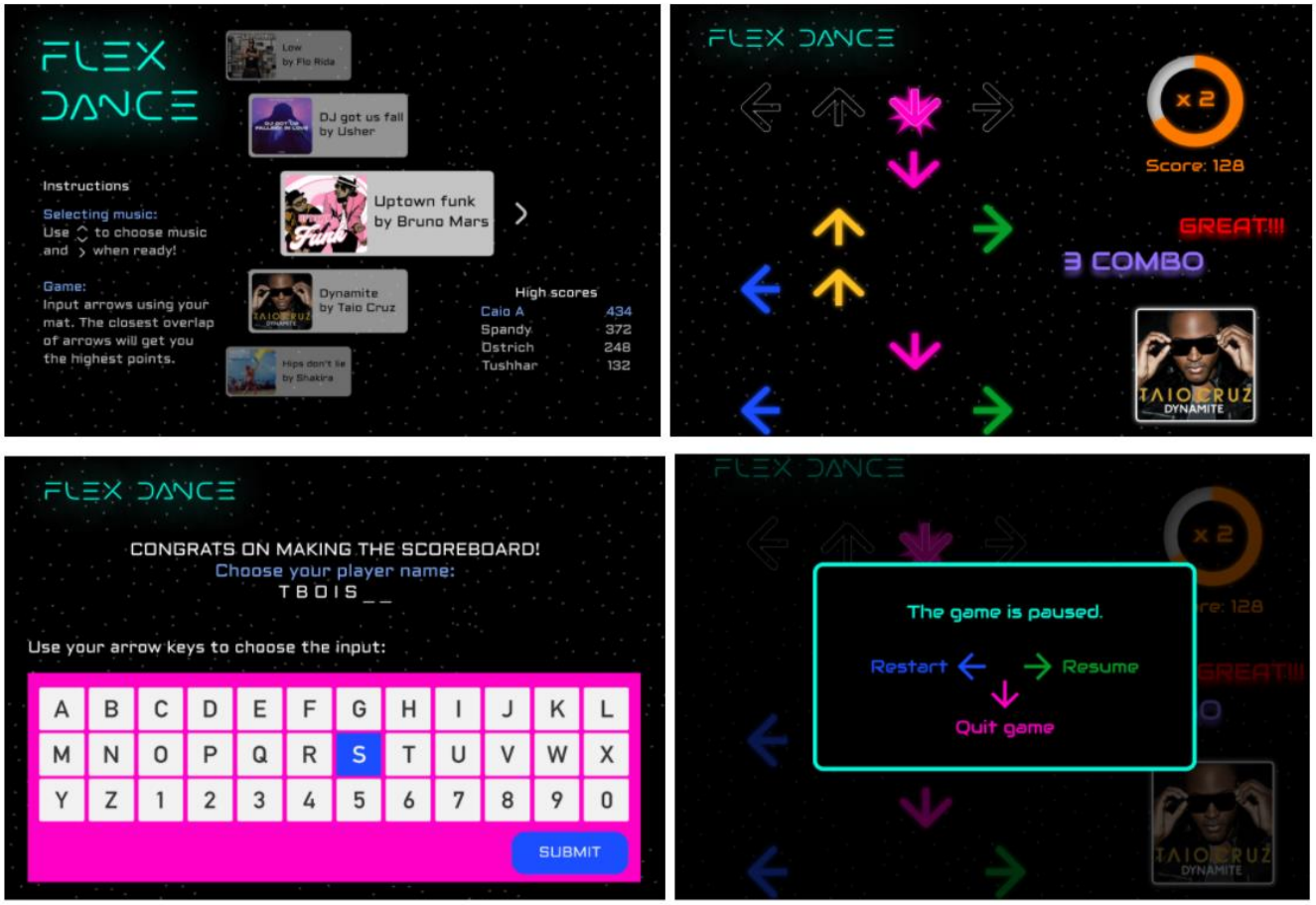

Figure 4: Game User Interface. In clockwise direction: menu screen, game screen, paused screen, score screen

IV. Design Requirements

Based on the goal of providing a smooth at-home exercising experience to our customers and the identified use-case requirements, we have come up with the following design requirements.

A. Force detection:

As can be seen in figure 1, each button on the mat is equipped with a FSR to detect the force produced by the user's feet. This system of detecting force is quite complicated so we have broken it down in subsections.

1) Threshold: Since the force sensors will deal with deliberate as well as accidental triggers by the users' feet, our force sensitive sensors should be able to differentiate between them and detect a force of at least 10 lbs as a threshold. This threshold was determined through our testing of the sensors. We used the sensors to determine the force produced by a resting foot as well as that of a foot putting deliberate pressure.

2) Button coverage: The force of the player's foot, in any orientation, should be able to be detected in any part of the button. Hence, the buttons on the mat should provide 360° coverage of force detection. This is a subconscious requirement of the users and must be satisfied by our design.

3) Pressing vs. Holding: The mat should be able to differentiate between pressing the buttons and holding the buttons for more time. This is necessary for navigation between menu and game screens as well as for increasing the complexity of the game after we are done producing a MVP.

B. Error rate

Our game should have relatively less errors, even in the MVP. If we have a player playing our game and they are a master at it, we want them to be able to ace and win 1 in 4 games. This gives us a 25% chance of scoring every arrow correctly, thus giving us an error rate of approximately 1%. So, the error rate in our scoring system should be less than 1%. These calculations are illustrated in Fig X.

C. Latency

We anticipate that most of the latency in the game will occur between the Arduino and the Raspberry Pi. This latency should be less than 1/10th of a second or less than 100 ms as that is the minimum time needed for humans to perceive images. This is a design requirement to satisfy the users requirement of having an engaging game.

D. Game interface

Our GUI (Graphical User Interface) will be one of the two crucial points of contact between the game and the user (the other one being the mat) and so the interface must be as flawless as possible. This results in two design requirements:

1) Quick game startup: The game should be easy to start and ready to be played by the user to avoid losing their interest and keeping the game engaging. The game should be pre-installed so all the user has to do is connect the game to a display screen using a HDMI cable and start playing. The process of getting to the game screen from the menu screen should not be a tedious process and thus, the game screen should only be three clicks away from the start of the game.

2) Engaging interface: The interface should be engaging and have few animations to catch the attention of the users. It should also not be overwhelming at the same time. After doing research, we have determined that our interface must follow the 60-30-10 rule of UI and UX design. This rule states that in order to keep a GUI from being overwhelming, three prominent colors in the ratio of 60-30-10 should be used. Thus, there should be smooth animations and the visual content on the screen should be organized in order to avoid confusing the user.

E. Durability

We expect our game to be durable and last approximately 624 sessions without the need of maintenance. If a person ideally exercises 4 times a week and expects a household item to last three years, we conclude that our game should last 624 sessions of usage. This is explained in further detail in the equations below.

Let us assume that each session of usage lasts an hour.
1 session = 1 hour
1 week = 4 sessions = 4 hours
1 year = 52 weeks = 52 x 4 hours = 208 hours
3 years = 3 x 208 hours = **624 hours**

V. Design Trade Studies

A. OS for Raspberry Pi

For our operating system, we chose the Raspberry Pi OS Lite (formerly known as Raspbian Lite). This operating system is ideal for a device that runs one single game with limited input since it does not implement a desktop screen with window management capabilities. It also boots fast, has good documentation, and takes up little memory, which leaves more space for our software and song files in the SD card. On top of all that, it runs Python natively, which is the programming language we chose for our software.

Other OS options we considered included the non-lite version of Raspberry Pi OS, which takes longer to boot and has a window manager that would get in the way of our game, and DietPi, which has less documentation and requires apps to be specifically optimized for it. Out of all three, RPi OS Lite was the safest and most coherent option.

B. File format for soundtracks

Rhythm games must specify a sequence of desired inputs (in our case, arrows) for each song so that the player can try to hit those inputs at the right moment, and these sequences are often encoded in a properly formatted file. For Flex Dance, this file is a text file containing a measure-by-measure description of when an input should be pressed in that measure and the time-length of each measure in the song. An example is shown in figure 6.

Each measure has 16 'spaces' in which an arrow can be stepped. The arrows are described in order (LEFT, UP, DOWN,

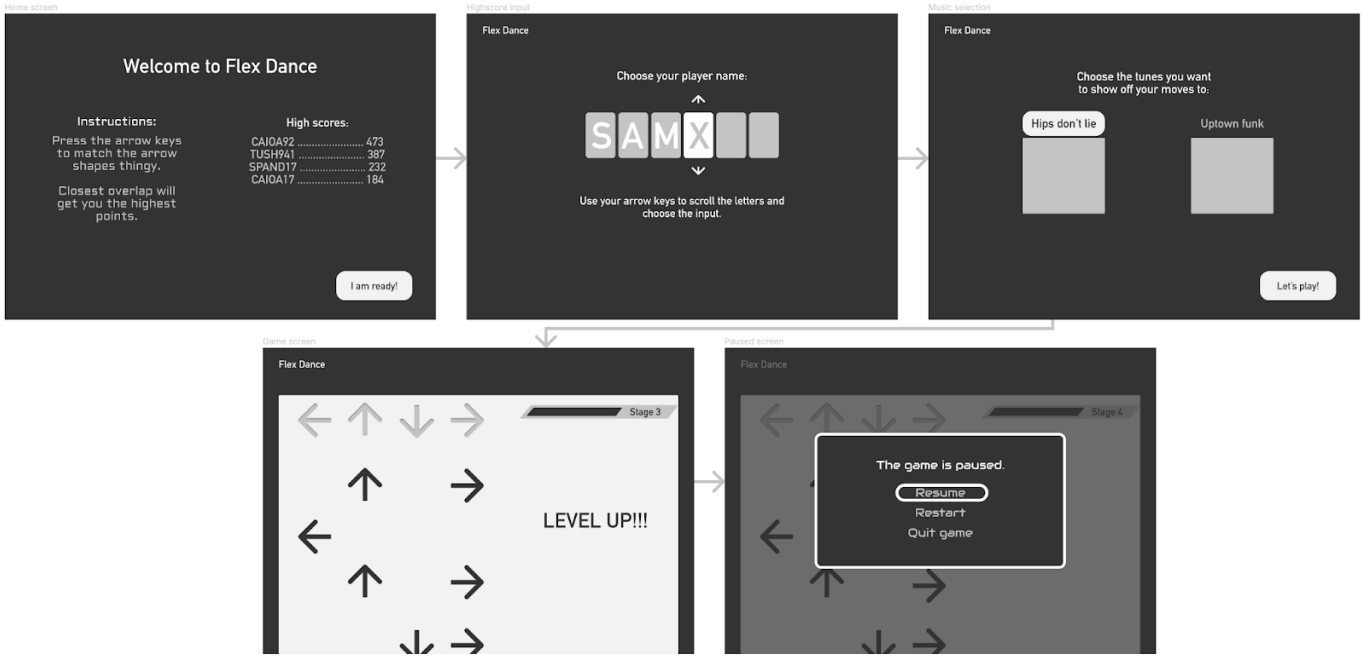

Figure 5: First iteration of GUI

RIGHT) with a 1 indicating that the arrow in that position should be pressed on that moment in the measure. Using this description alongside the measure per minute of the song on the third line, the software can properly predict how early an arrow should appear in the screen in order to match its note in the measure. Anything after a pound symbol is ignored when parsing the file. We took inspiration from another rhythm game, Stepmania, on how to encode these files.

```
File Edit Format View Help
# Song: Uptown Funk
# Measures per minute:
15
# Measure 1
0100 # UP
0000
0000
0000
0010 # DOWN
0000
0010 # DOWN
0000
0100 # UP
0000
0000
0000
0010 # DOWN
0000
0010 # DOWN
0000
# Measure 2
...
```

Figure 6: Text file for arrow sequence description.

C. Pygame vs Unity

We chose to use Pygame for our game engine. Python, and by extension Pygame, interfaces very well with the Raspberry Pi. Also, since it is a widely known library, it is incredibly well-documented. All of our team members are familiar with Python, so we believed that we could learn Pygame on our own through online tutorials and the documentation.

The other contender for game engine was Unity, which is very flexible and also well documented. Unity, however, does not interface easily with the RPi and we were concerned about how long it would take us to learn how to use Unity as well as we know how to use Python. Finally, since we have a better understanding of Python through university courses, it is easier for us to customize our game at a lower level.

D. FSRs

To sense feet stepping on the mats, we brainstormed options like mechanisms involving a flip of a switch or making connections between 2 conducting materials. However, these options would either not be able to handle the weight of a human, be prone to accidental triggers or be awkward to step on. Therefore, we decided to go with force sensitive resistors, which can counter the issues described above and also fulfill our purpose. Moreover, one of our group members also has had experience with FSRs making it the ideal option.

In terms of specifics for our force sensitive resistors, we considered various different options in terms of quantity and coverage. Ultimately, we came down to 2 options. The first option was to use a 2ft long FSR and bend it in a way that it covers a lot of area. The second option was to have multiple smaller FSRs under each square. While both were viable, the second option would end up requiring a lot more wires and connections to the Arduino. Furthermore, they do not have as much coverage as we would ideally like. After some testing with the first option, we found that the 2ft long FSR could be bent (with very little damage) in a Z shape for a lot of coverage and still take proper measurements. Considering the pros and cons of each option, we decided to go with the first one. However, we still have the second option as a backup plan if we run into problems.

E. GUI

For initial game interface, we designed the screens in grayscale following the design process learned in our previous classes (Fig 5).

For the home screen, we found that it was too text heavy. For the high score screen, we used a lottery slot machine design to allow users to select their player names. However, this interface design would require the player to keep pressing arrow buttons on the mat which would be tedious if the player wanted to use letters such as S or Y. So, we decided against it. For our music selection screen, we thought it was too boring and decided against it. For the game screen, we liked the arrows placement and kept the design as it was similar to the original Dance Dance Revolution game. For the score, in the game screen, we had designed a progress bar to show how much of the song was left. However, we thought that was redundant. Lastly, for our screen when the game was paused, we used the conventional screen. But to choose options like "Restart", the user would have to click at least twice: once for moving towards the option and twice to confirm the selection.

Based on our observations, we performed a second iteration. We first sketched the ideas on paper to make sure we all agreed upon it. The second design looked like this (Fig 7):

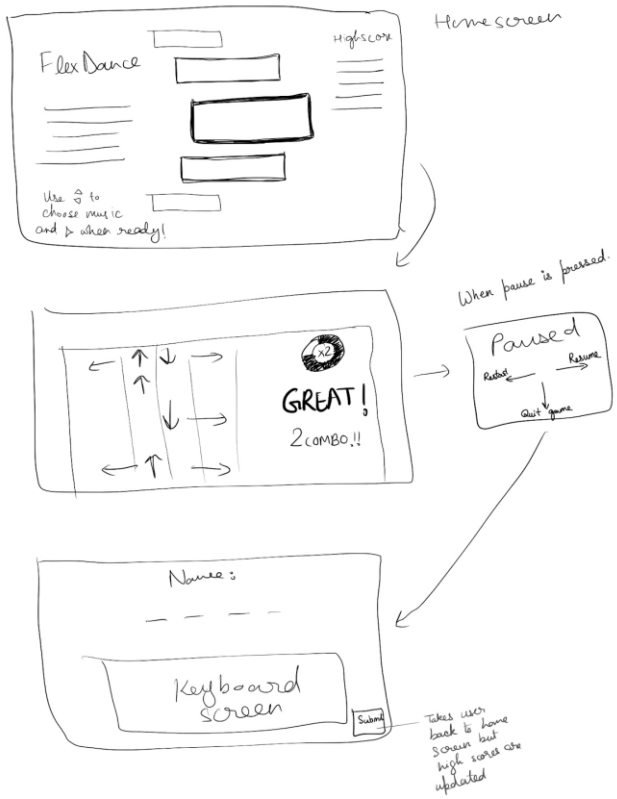

Figure 7: Second Iteration of GUI

As seen above, we combined music selection and game screens in order to reduce the screens needed to reach the game screen. The music selection is designed to have smooth animations which makes it engaging yet not overwhelming. The high score screen was arranged to be after the game screen and could only be landed upon if the player scored high enough points. The game screen was left mostly intact with the only change being in the progress bar. We decided to have a score multiplier to give incentive to the user to continue playing competitively. The score multiplier would collect points based on how many arrows the player is able to score correctly. Another change we made was in the "Pause" pop-up box. We designed it to take inputs directly corresponding to the arrows on the mat. This minimizes the number of clicks and makes the interface more efficient. Lastly, the high score input screen has been designed to have a keyboard input which makes it easier to input names rather than having to click or hold the buttons for a long time. Based on these changes, we have our final interface design that can be seen in figure 4.

F. Mat Material

For the material of the mat, we considered several options. First was a normal fabric but we thought it would not be rigid enough and the FSRs could sense it. Then we considered Styrofoam but that could break easily. We also considered wood but that would increase the weight of the product thus, making it unfit to be lifted by young children if they wanted to. Finally, after much deliberation, we settled upon using a mix of tarp and EVA foam. The tarp is rigid and water resistant thus, making our mat more durable. The EVA foam would provide cushion surface for the feet so the player is comfortable standing on the mat even on a hard floor surface. The possible orientation of the layers that we are considering currently are given below:

Figure 8: Different arrangement of layers of tarp and EVA foam for game mat

G. HDMI

Since we are developing a computer game, we needed to decide what kind of display we would use. HDMI prevailed over other options such as DVI since it is more convenient. Most, if not all, modern televisions have at least one HDMI input, and we expect our users to play Flex Dance in a larger space such as their living room on their TV. Also, the RPi supports HDMI out of the box with no need for any adapter. Lastly, HDMI transports both video and audio, which means we will not have to worry about providing a speaker interface since the user's own display will manage that.

H. Raspberry Pi 4B

One of Flex Dance's most important features is ease of storage, so whichever device is running our software has to be compact. The Raspberry Pi line of microcomputers shines in this aspect. A Raspberry Pi is powerful enough to run the code that we need to run and it also fits inside any drawer that the user has in their home. Some of our group members also have experience working with RPi's. We chose the Raspberry Pi 4B in particular because it is the most recent model, overpowering the Raspberry Pi 3B+ in every major aspect, and it has USB 3.0 ports which permit faster transfer of data between the mat and the RPi, which will improve our speed, even if by just a little bit. Finally, since the RPi files are stored in a micro SD card, it is easy to change the files in it through our laptops.

I. Arduino

We needed an ADC to convert analog voltages produced by our FSRs into digital USB signals that the software can read. We chose to use a headerless Arduino Leonardo for a few reasons. All of our group members are familiar with Arduino code and building circuits using Arduinos. As such, this reduces the time we need to invest in learning how to use this microcontroller. Also, the Arduino Leonardo interfaces with computers through a micro-USB to USB cable and can send information through that cable. USB Since we can program the Arduino, it allows us to do some state processing in the Arduino itself which lightens the load on RPi. For example, the Arduino can check which button state (pressed, idle, held, etc.) the voltages read represent and just send the state to the RPi instead of the raw voltage.

VI. System Implementation

This section will go into depth about how each of the different subsystems work and how they are connected to the other subsystems. The system as a whole is shown in Fig 1. We will start from the mat which receives the inputs and follow through to the Raspberry Pi. Then, we will finally talk about the Graphical User Interface.

A. Game Mat

The layout of the mat from the top layer to the bottom layer will be a layer of tarp, a layer of circuits, a layer of eva foam and finally, another layer of tarp. The arrows, buttons and borders will be painted on the tarp clearly to show where the user should step. The square mat is divided into 3 x 3 squares with the force sensitive resistors (FSR) being present on 6 of the 9 squares (check Fig 1 for locations). These FSRs correspond to the arrows that users need to step on to play, as well as the pause and confirm buttons to navigate through the user interface.

The FSRs are connected to resistors and the Arduino through stranded wires that have been soldered. These wires are also soldered to the headerless Arduino. The corresponding circuit diagram is shown in Fig 3. The Arduino has spikes in the analog values it measures from each FSR. If it crosses a certain threshold, it will detect the step as a hit. Following this, it will send an array of 6 values to the Raspberry Pi, which can then be used in the game.

B. Raspberry Pi

The Raspberry Pi will have an Operating System installed so that it is easier to run Python and Pygame in it. It will also be set to run the game on boot up. This will allow the user to simply plug it into a power source and then start playing the game.

Furthermore, the Raspberry Pi consists of all the files listed in the block diagram in Fig 2. The mat_event_handler.py file takes in the array sent by the Arduino and triggers a pygame event when necessary. The game_components.py file implements the basic classes (and their subclasses) that make the game: Screen, Button, and Arrow. A Screen object has its own drawing and updating methods to make it simple to draw and update the game through main.py. A Button object has a method that does something (change a screen, update the current screen, update a high score, etc.) according to which input is pressed when the button is selected. The song_reader.py file implements a class to read our custom track files (a zipped archive containing an MP3 file with the song and a text file describing the arrows for that song). An Arrow object represents the arrows that move during the game. The constants.py file sets all the constants needed for the game, such as FPS, sounds, icons, and colors. Sounds and icons are stored in the Assets folder. The Saves folder holds save file data, such as high scores for each song. Finally, the main.py file runs the main loop of the game, updating and drawing the screen using objects from the previous files.

C. Graphical User Interface (GUI)

Fig 4 illustrates our initial design for the GUI. As mentioned in the Design Trade Studies Section, the different game screens were inspired from games similar to Dance Dance Revolution.

The menu screen has clear instructions on how to play the game. Navigating through everything should also be quite intuitive. The user can scroll through the songs in this screen to decide on which one to play. The high scores corresponding to the song will also be displayed there.

The game screen will be similar to how real DDR have their arrows come in. They will arrive from the bottom and the user needs to hit the correct arrow when the moving arrow on the screen is close to the stagnant one. We will also implement game features such as combos and score multipliers. Both of them will go hand in hand i.e. the score multipliers will increase with higher combos. The pause screen can be reached simply with the pause button. Pressing the appropriate arrows as shown in the pause screen will result in the corresponding actions. Particularly, these are ‘restart’, ‘resume’ and ‘quit game’.

The last screen will be a screen for inputting your name if you make it to the leaderboards. For now, there are 7 spaces to input letters or numbers. The arrows will be used to navigate to the correct symbol on the keyboard, and the confirm button will be used to confirm the selection of each letter.

VII. Test, Verification and Validation

A. Tests for ease of storage

When we finish building our mat, we will use a ruler (or measuring tape, whichever is more convenient at the time) to measure its width, length, and height, both when the mat is folded and when it is unfolded. When the mat is folded, it will pass this test if it occupies a space smaller than 13in by 12in by 5.5in, which we estimated to be a lower bound on the size of a drawer, where we expect our user to store the mat while not in use. When the mat is unfolded, it will pass this test if it occupies a surface area of 39in by 39in, which we estimated to be the minimum area in a living room necessary to play the game comfortably and safely.

B. Tests for force detection threshold

From hands-on testing with a force gauge, we estimated that the weight of a resting foot is approximately 10 pounds. We want our mat to consider anything above this 10 lb threshold to be an intentional button press. This will involve setting a digital voltage threshold for the Arduino through its code. We will place a 10 pound weight on top of the FSRs while they are

connected to the Arduino and read the voltage value through the serial monitor. This should give us a rough idea of what this threshold should be and we can set that value for our detection threshold. Finally, we will test it using our own resting feet and adjust the threshold accordingly to whether we ourselves would expect the current weight put on our feet to trigger the sensors.

C. Tests for arrow coverage

Flex Dance can be a fast paced game, and thus we do not expect players to always be completely precise on which part of the arrow squares they step during the choreography. As such, it is important that our individual squares have good coverage, being able to detect feet in many different positions and orientations. For this requirement, we will test each square by randomly stepping on it on different points and registering which and how many steps it actually detected. Our goal is to detect an average of over 95% of these steps for a good user experience. If we notice that a particular orientation is not working well, we can work on detecting that specific orientation better in order to reach our target.

D. Tests for latency

In a similar vein to the previous requirement, high latency will also hurt the user experience if it is noticeable. Whether our mat can detect steps or not will not matter if these steps are registered with too much delay. We believe the longest latency in our platform will come from the software and not the hardware. As such, we are assuming that the time between stepping on the mat and sending that through USB to the RPi is negligible. In order to measure software latency, we will use the Python function time_ns() to register the times, in nanoseconds, (a) when the RPi first reads the USB signal that the arrow was stepped on and (b) when the game is done computing whether that arrow was a hit or not, since this is the feedback that the user will receive from that step. Our goal is to achieve $t_b - t_a < 100ms$, which is the minimum time for humans to perceive duration and, by extension, delay.

E. Tests for error rate

With games in general, it can be very irritating when the player's input is read incorrectly. For Flex Dance, we aim to have an input error rate of less than 1%, meaning that if the user steps on an arrow, that step will be correctly registered over 99% of the time. We chose this target following the calculation below, in which we assume that a perfect player will score perfectly 25% of the time in a track with 200 arrows in total and eta represents the error rate:

$$P(scoring\ every\ arrow) = 0.25$$
$$\Leftrightarrow (1-\eta)^{200} = 0.25$$
$$\Leftrightarrow \eta \cong 0.001$$

Equation 1: Error rate calculation

Noticeably, this requirement is similar to our coverage requirement, but the coverage requirement involves tinkering with the layout of the sensor and the materials around it to make sure we can detect steps in different regions of the arrow squares. When we test for error rate, we will have already verified that we meet the goal for coverage. In order to measure this error rate, we will have one person dance to a song on the mat while another person watches the player's steps to mark down whether they should have been detected. Then we can compare the player's score, or how many times the game detected their steps, versus the watcher's notes.

F. Tests for scoring scale

The main purpose of our linear scoring scale is to be forgiving towards users when they cannot hit an arrow at the exact time. As such, we want to test it through user interactions. We will have some people play the game when it is in MVP state and ask them whether they feel that their final score was an honest reflection of how well they played. This feature will be verified once we achieve 75% or more positive responses. According to different news articles and research, around 75% of people believe that at-home exercising will remain as a trend, even post-pandemic. As such, we want to make sure our product caters to at least this same percentage of people. Should we fail to meet this goal, we will rescale the score awarding system to be more forgiving (i.e. give more points for less precise steps) and run these tests again until we reach our target.

G. Tests for cost

Finally, we want our product to stand out from other at-home alternatives by being comparatively cheap. We are aiming for a final purchase cost of at most $200.00, which should be less expensive than the majority of other available options. Larger options such as gym machines cost at least $500.00, and most are much more expensive than that, falling in the $1,000.00 to $3,000.00 range. The game-like alternatives like Just Dance require a video game console and motion tracking hardware are also above our target. The most cost-effective Just Dance experience is provided through the Xbox 360 ($150.00 without the HDMI output) with the Kinect ($20.00) and a copy of Just Dance ($10.00), which amounts to a total of $180.00, but without the benefit of Flex Dance's simple interface dedicated solely to our software and no other services. In order to reach our purchase cost goal, we will sum up the cost of every material used in our final product and see if the building cost is less than $180.00, which in turn allows us to sell our product for $200.00.

VIII. Project Management

A. Schedule and division of labor

All team members are handling specific tasks for a couple of weeks. Afterwards, all team members will come together to integrate the different components of the game. The area of expertise along with the specific components are given below:

1) Tushhar Saha: He was the presenter of the Design Review presentation and is handling circuits of the game. He is currently working with FSRs to create buttons for the mat.
2) Caio Augusto Araujo: He will be the presenter of the Final presentation and is currently handling the software systems. He is working on creating the game screen in Pygame and arrow tracks for the two songs needed for the MVP.
3) Spandan Sharma: She was the presenter for the Proposal presentation and is currently handling the software systems and

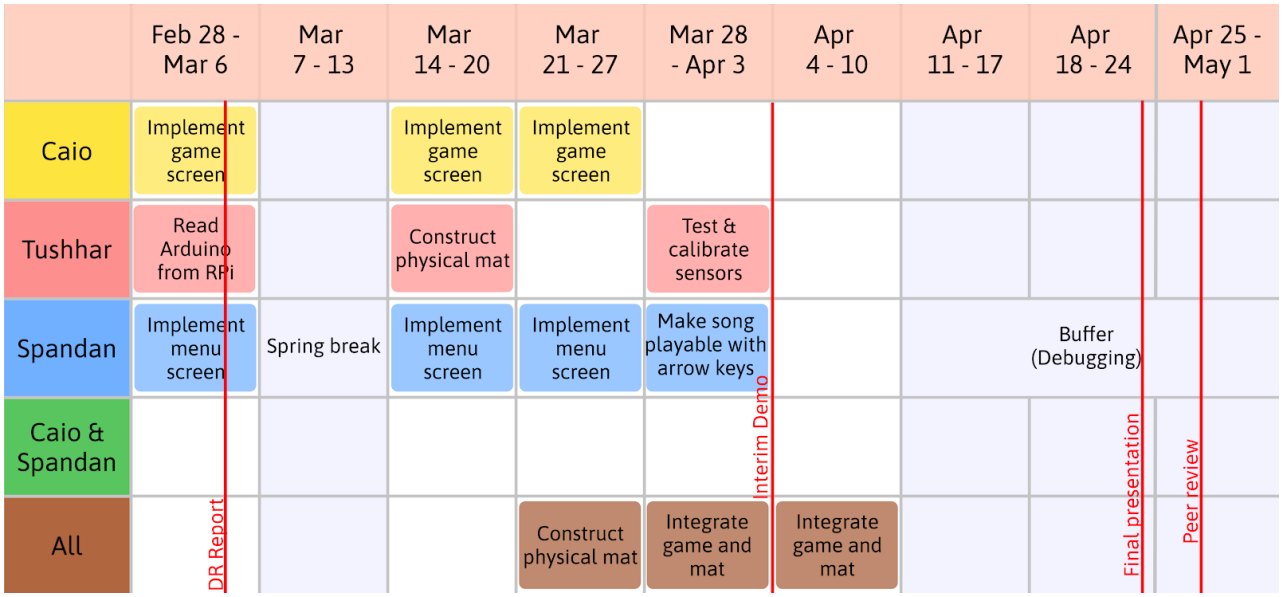

Figure 9: Schedule and Division of Labor

UI design. She is working on creating the menu screen in Pygame and ensuring the interface design is user-friendly.
Everyone will be working together to integrate the game interface and the mat.
4) The schedule and team member responsibilities are given in the chart below. The chart has been updated and the timeline is starting from the submission week of the design report.

B. Bill of Materials and Budget

Our bill of materials is shown in figure 10 at the end of the document, and we will have $375.24 left in our budget after we order the rest of the parts.

C. Risk Mitigation Plans

Based on the testing and progress since the design review presentation, our team has come up with a few mitigation plans. If our Arduino is unable to detect the signals from the force sensors when the player steps on the buttons, we will either decrease the threshold or scale the readings. The modification of the threshold also applies to account for the error rate.

If in our testing, it is revealed that the users are unhappy and think the scoring is unfair, we will be more forgiving with the scores. If they have any suggestions for the improvement of the interface design, such as the screen is too “text-heavy”, we will try to modify our Pygame algorithm to implement their suggestions.

If our game has a lot of latency, we will decrease the usage of external assets such as images in icons. Rather than using our customized images of arrows, we will use Pygame functions to draw them.

IX. Related Work

There are a few other alternatives for at-home exercising other than Flex Dance. The one we believe is the most competitive towards us is the game series Just Dance. Just Dance uses motion tracking software supported by a video game console (usually an iteration of the PlayStation or the Xbox) to capture players’ abilities to follow a choreography shown on the screen. This game shares many of the appeals of Flex Dance: it occupies little space, only requires the console to be installed, and also provides a fun workout through dancing. The advantages of Just Dance are that it does not require a mat since it uses a camera to track motion and it allows for more diverse choreographies since it tracks the players entire body and not just whether they step on a specific place. Just Dance, however, proves to be a poor alternative for a casual exerciser who cannot justify buying a video game console solely for exercising, as the entire setup would cost at least $300.00 (console + motion tracking hardware + Just Dance software). Also, for someone who is not used to console menu interfaces, they can be overwhelming at first since the console offers much more than Just Dance. Flex Dance aims to provide an intuitive and simple interface without the multiple other apps that consoles throw at their users.

Another digital competitor is Stepmania, a free DDR-like computer game that allows users to create and share their own choreographies to songs. Unlike Just Dance, and similar to Flex Dance, Stepmania uses a directional mat, so it has the same limitations in terms of choreography as our platform. Although Stepmania is a heavily supported software with a large community, it still requires a reasonable computer to run it, which once again might prove unjustifiable to buy for the casual exerciser who does not need an overly powerful computer. Also, Flex Dance is intended to be played on a TV, and not everyone has their desktop or laptop connected to a TV in their living room.

Finally, there are the non-digital alternatives. The most preferred ones are usually small exercise tools, such as dumbbells, kettlebells, and elastic bands. These options are on the cheaper side and are quite easy to store. The main caveat with them is that they are usually not particularly enjoyable (aside from the mental reward of exercising), especially for beginners. On the other side of the spectrum, there are home gyms and wall-mounted exercise machines. These are usually quite expensive and hard to assemble and/or install. They are usually quite flexible, which is a clear advantage, but in most cases are too glamorous for the casual exerciser.

With all these options in mind, Flex Dance finds a niche with casual exercisers who are looking for budget options to enjoyably exercise at home.

X. Summary

Our design provides an enjoyable, comparatively cheap, easy to store, and beginner-friendly alternative to people who want to exercise at home. Our platform consists of a foldable, pressure sensitive mat that acts as a controller for a rhythm game running on a Raspberry Pi which can connect to any HDMI display the user already possesses, such as a modern TV. The player follows a sequence of directional arrows shown on the screen by stepping on the corresponding arrow on the mat. This formula has been tested and proven by existing games such as Dance Dance Revolution and we have converted it into a convenient household product. Now, any casual exerciser who wants to stay fit from the comfort of their home, be it because of quarantine or just inclement weather, can have a small device that assists on their workouts.

At this moment, we anticipate three major challenges, namely minimizing latency between stepping on an arrow and seeing feedback on the screen, providing good coverage for each different button on the mat, and staying under our allocated maximum purchase cost of $200.00. For the latency challenge, we recognize that our game will be doing many things at the same time, such as drawing images, reading USB input, and playing music, which can all slow down the software, especially in a small device like the Raspberry Pi. It will be very frustrating for our users if at any point our latency is long enough that the player notices a delay between them stepping on an arrow and being awarded points.

Second, if our mat has poor coverage of the arrow squares, minimizing latency will be irrelevant since we will not detect steps anyway. Once again, this creates a very disappointing experience for anyone using our device. The force-sensitive resistors that we are using can only bend so much and in so many directions, so if we fail to provide good coverage using these sensors, we will have to reevaluate our approach.

Lastly, although we have managed to stay under the budget that we projected for our entire device so far, we have not accounted for the price of the Raspberry Pi and the SD card, since these were retrieved from the ECE inventory. Once we account for the market price of these two parts and possibly other parts that we need down the line, it is likely that the cost of materials used to build the final product will stop any profit of selling this platform for $200.00.

References

[1] NextFab, ‘Making a USB Game Controller’, https://www.instructables.com/Making-a-USB-Game-Controller/

[2] Last Minute Engineers, ‘Interfacing Force Sensing Resistor (FSR) with Arduino’, https://lastminuteengineers.com/fsr-arduino-tutorial/#:~:text=Force%20Sensing%20Resistors%20are%20also,pressure%2C%20squeeze%2C%20and%20weight.

Bill of Materials						
Part/Model Number	**Manufacturer**	**Quantity**	**Cost / component**	**Total cost**	**Source**	**Short Description**
Arduino Leonardo w/o headers	Arduino	1	$23.40	$23.40	Arduino Store	Wires will be soldered on to the headerless points
2 ft Force Sensitive Resistor	Interlink Electronics	6	$19.95	$119.70	Adafruit	Will act as the force sensors to be stepped on
Raspberry Pi 4B	Raspberry Pi	1	$0.00	$0.00	ECE Inventory	A mini computer to run our game that can connect to the Arduino through USB and any HDMI display.
32 GB SD Card	(check tomorrow)	1	$0.00	$0.00	ECE Inventory	A memory card to store the OS for the Raspberry Pi, along with our game files.
Black tarp	Harpster Tarps	1	$17.99	$17.99	amazon	Material for the mat
Evafoam	The Foamory	2	$13.99	$27.98	amazon	need the 2mm one
Neon Paint	Neon nights	1	$25.00	$14.44	Amazon	To paint the arrows and additional buttons on the mat
Thread	Unique Craft	1	$12.99	$12.99	amazon	need the black one
Resistors	Any	10	$0.00	$0.00	ECE Inventory	The resistor values will be decided according to thresholds we want
100 ft Stranded Wires	Any	1	$0.00	$0.00	ECE Inventory	Wires to be soldered to resistors, FSRs and Arduino
Mini HDMI to HDMI cable	(check tomorrow)	1	$0.00	$0.00	ECE Inventory	A cable to connect the Raspberry Pi to any HDMI display.
10-feet USB A to Micro USB (for Arduino)	Amazon Basics	1	$8.26	$8.26	Amazon	A long cable to connect the Arduino from the mat to the Raspberry Pi with a considerable distance so the user does not stay too close to the TV.
			TOTAL	$224.76		

Figure 10: Bill of Materials