Flex Dance

Caio Araujo, Spandan Sharma, and Tushhar Saha

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—Our system attempts to improve the at-home exercising experience, motivated by recent quarantine mandates. 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. Current alternatives 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. 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 global pandemic, people were unable to exercise in the conventional manner (such as going to the nearest gym) and the indoor options were cumbersome, hard to store, boring, and expensive. This adversely affected the physical, mental, and emotional health of people as feelings of isolation, fear, and loss overwhelmed everyone. 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.

In the game, players follow along a choreography to a song of their choice by stepping on directional arrows laid out on the mat, thus making it engaging and fun. Our game is also user-friendly as it is pre-installed in a Raspberry Pi that the user simply connects to their TV. This requires no setup on their part. Lastly, our game is portable as all the parts (such as Raspberry Pi and Arduino) are small and the game mat is foldable. This makes the whole setup compact and easily storable for any average drawer or cupboard. 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 inflict property damage. Finally, there are options more similar to what we propose such as the arcade game Dance Dance Revolution (DDR) [2,3] and the game series Just Dance [9].

Our main advantage over DDR is that we provide an easily storable and an indoor exercising experience that is cheaper than the $8,495 arcade machine. As for Just Dance, the game series requires a commercial video game console which costs over $200 in addition to the necessary motion-tracking equipment to play the game. The user might not be interested in using this console for purposes 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 13x12x5.5 in^3 and when the mat is expanded, it should fit comfortably in a living room space of 39x39 in^2.

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. Stepping platform size

Relating to the point above, we want this game to be accessible to most people, so our stepping platforms have to be large enough to accommodate them. The American men's average foot size is 10.5 [4], while the women's is 9 [5]. As such, a platform with an area over 11x11 in^2 will serve most people.

E. *False negative rate*

In our case, a false negative means the player stepped on an arrow, but the step was not recognized. We want our game to be enjoyable, so we are aiming for a false negative rate smaller than 1%. We will focus mostly on reducing false negatives over false positives (observing a step when there was no step), since false positives can usually benefit the player and thus are less harmful to the player's experience.

calculations are explained further in depth in *Design Requirements*.

III. Architecture and/or Principle of Operation

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

The Game Mat is responsible for detecting inputs from the 5

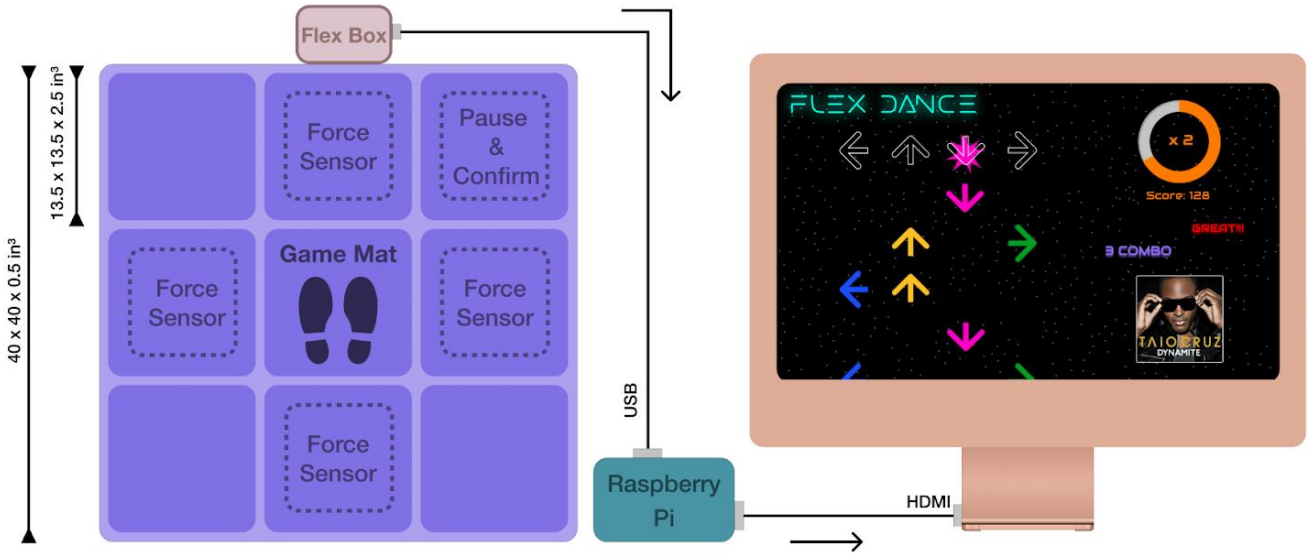

Figure 1: Subsystems overview

F. *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.

G. *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 screen thus should be a maximum of 3 clicks away from the start of the game.

H. *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 force sensitive resistors (FSRs). In the design report, we had 6 FSRs as we thought we'd need separate buttons for 'Pause' and 'Confirm'. However, we decided to combine it into one to fit our on-screen keyboard better and to save costs. 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. The circuit needed to read the FSRs and send it to the Arduino was soldered onto a perfboard and is shown later in Fig 5. In the design report, the circuit used to be underneath the mat. However, changing the majority of the circuit to be located in the perfboard helped us reduce our worries about components and connections underneath the mat. Together, the Arduino and the perfboard are located in the FlexBox, which is a laser cut box designed to house these components and make the system look prettier.

On the other hand, the Raspberry Pi contains everything needed to run the game including the game files, assets, an operating system, and pygame [10]. External software libraries and packages used for implementing the game include 'pygame', 'math', 'pyautogui', 'os', 'time' and 'tkinter'. The mat.ino file is uploaded to our Arduino and makes the Arduino emulate keypresses through stepping on the arrows. The components.py file implements the basic classes (and their subclasses) that make the game: Screen, KeyboardKey, and AnimationHandler.

The song_reader.py file implements a Song class that can read and store information about a song. 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 main.py file runs the main loop of the game, updating and drawing the screen using objects from the previous files. The Saves folder holds save file data, such as high scores for each song.

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 is designed and built by us.

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 Fig. 1, each button on the mat is equipped with 2 FSRs 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 resistors 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. This is necessary to make sure that just holding the button doesn't trigger multiple arrow presses.

B. Error rate

In order for the best user satisfaction and in line with use case requirements, our game should have very few errors. In particular, we are considering false negatives, where the user steps on the mat button but is not detected. If we have a perfect player playing our game, we want them to be able to ace 1 in 4 games. Since the player cannot "lose" in Flex Dance, "acing" a song means they score every arrow with perfect accuracy, so they get the highest possible score for that song. Considering that an average song lasts for 3 minutes and 200 arrows, our calculations shown below give us an error rate of approximately 1%, and thus we set this value as our goal.

$$P(scoring\ every\ arrow\ correctly) = 0.25$$

$$(1 - error\ rate)^{200} = 0.25$$

$$error\ rate \cong 1\%$$

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 duration. 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 be a maximum of 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 calculations below.

$$1\ \text{session} = 1\ \text{hour}$$

$$1\ \text{week} = 4\ \text{sessions} = 4\ \text{hours}$$

$$1\ \text{year} = 52\ \text{weeks} = 52 \times 4\ \text{hours} = 208\ \text{hours}$$

$$3\ \text{years} = 3 \times 208\ \text{hours} = \mathbf{624\ hours}$$

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. 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.

C. FSRs

To sense feet stepping on the mats, we brainstormed options like mechanisms involving a flip of a switch or making connections between two 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.

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. Therefore, we chose the 2ft long FSRs.

Initially we had one FSR per pressure platform, and each FSR was bent into a Z shape for maximum orientation coverage. However, we soon found out that bending the FSRs eventually broke them, making certain regions not sensitive anymore. Although the Z shape provided great coverage, we had to change it since the durability was awful. After trying a few different layouts using two FSRs per platform, including an "X", a "+", and a "=" shape, we settled for the "=" shape, so two FSRs parallel to each other and cut so they were just 1ft long. This layout proved very durable with reasonable coverage.

D. GUI

For initial game interface, we first designed the grayscale prototypes shown in Fig. 2.

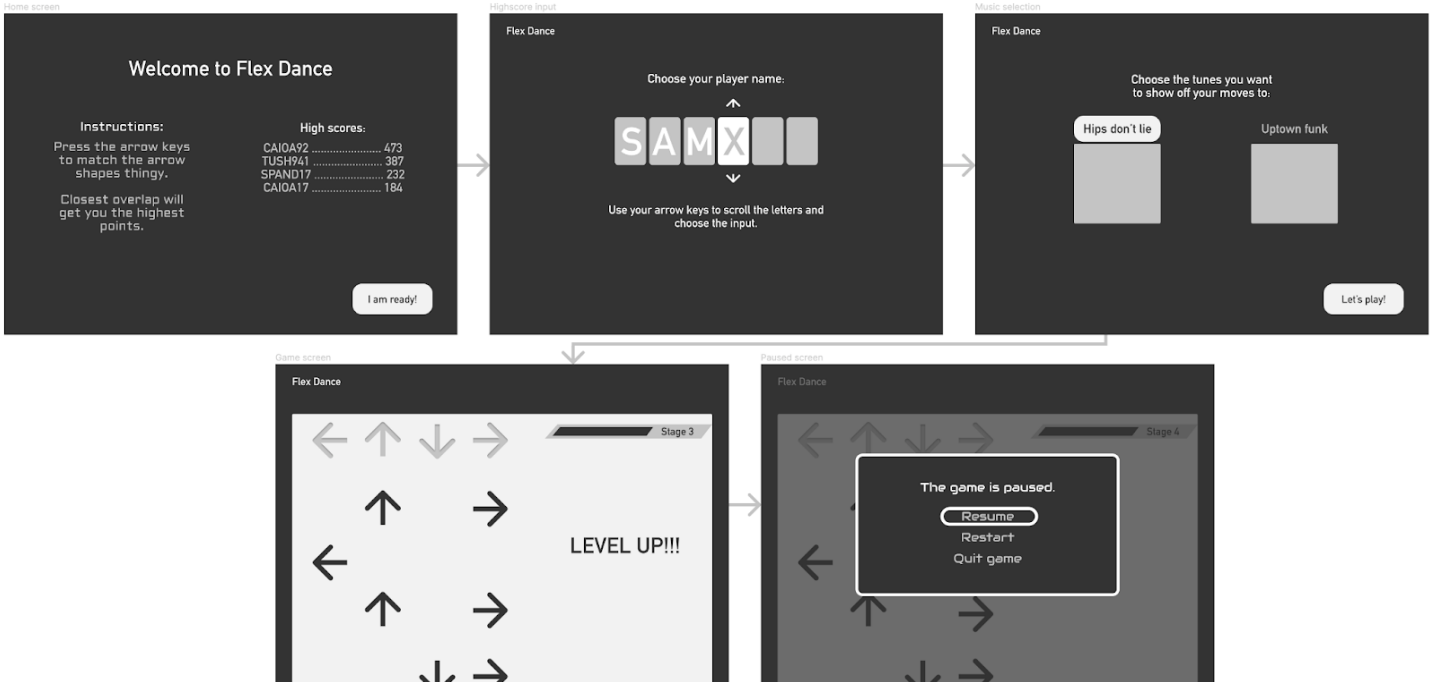

Figure 2: Greyscale GUI prototypes

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 sketches are shown in Fig 4.

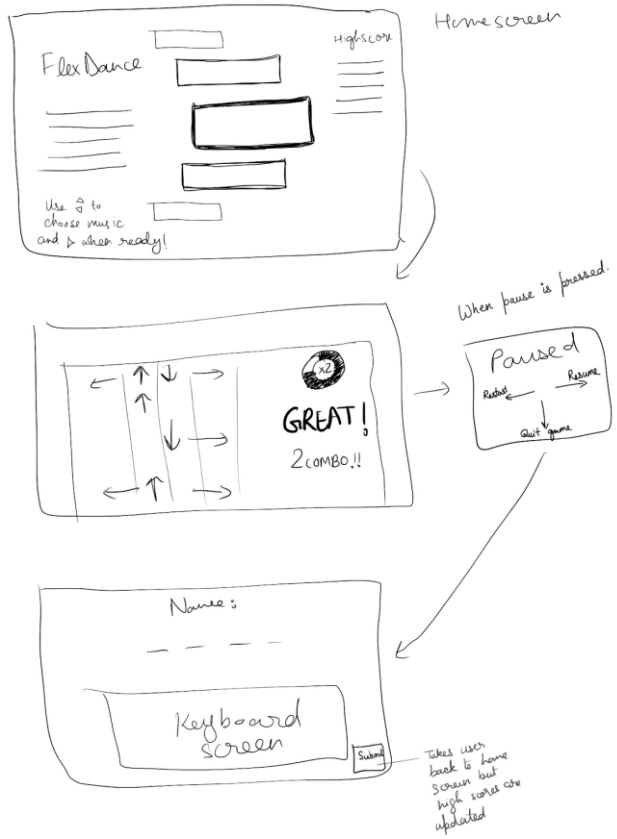

Figure 4: Sketches of the GUI's second iteration

As you can see, 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 the *System Implementation* section.

E. Mat materials

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, wood, and EVA foam. The tarp is rigid and water resistant, thus making our mat more durable. The wood provides some stability when someone is playing the game and distributes pressure to the sensors. The EVA foam cushions the surface for the feet, so the player is comfortable standing on the mat even on a hard floor surface. Fig. 3 shows the two structures we considered for the mat.

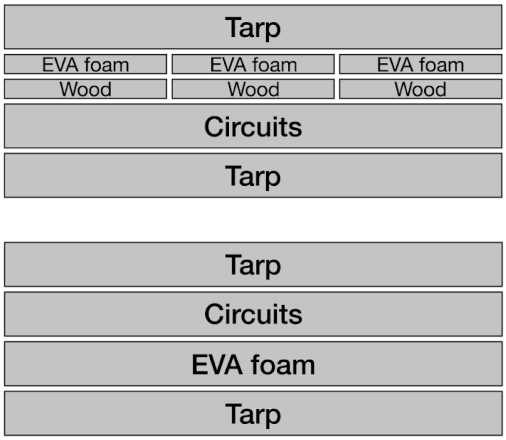

Figure 3: The two mat structures we considered

In our final product, we chose the structure on top, since it provided more protection for the circuits against friction from the tarp sliding on top of it and it was more comfortable to step on since the EVA foam was closer to the top.

F. 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.

G. *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 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, improving 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.

H. *Arduino Leonardo*

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 an 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. Finally, the Arduino Leonardo is one of the few models capable of emulating a keyboard. This particular quality makes it easier for us to build and test the game, since Pygame already has functions that handle key presses.

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. Refer back to Fig. 1 for the whole system. We first discuss the mat which receives the inputs, then the FlexBox which houses a perfboard and the Arduino, and then the Raspberry Pi which runs the software. Then, we explain more abstract components of our system, namely the Graphical User Interface and the Soundtrack File Format.

A. *Game Mat*

The mat is split into two types of components: a square base made of tarp and pressure-sensitive platforms for input. The square base is a single layer of tarp with heavy-duty velcro strips stuck onto it. The velcro strips serve to hold the pressure platforms in place during gameplay, but also allow easy removal of faulty platforms for testing and replacement purposes.

There are a total of five pressure platforms, each corresponding to one of the inputs to the game. Each platform is composed (from top to bottom) of a layer of EVA foam, a 12x12 in^2 masonite plate, two force-sensitive resistive strips, and two I-shaped pieces of wood that "sandwich" the FSRs to the square wooden plate. The platforms are wrapped around with tarp to protect their components and to facilitate painting. Each platform also has velcro strips stuck to its bottom so it can attach to the square tarp base. There is a platform attached to the center of the base as well, but that platform has no FSRs in it and only serves to make players more comfortable when standing on the center of the mat.

Both the square base and the platforms are painted for aesthetic purposes. The square base is painted with white dots and streaks to resemble galaxies, while each platform is painted with its respective input (an arrow or a pause button).

B. *FlexBox*

The FlexBox (Fig. 6) houses a perfboard and the Arduino. The pressure platforms' FSRs are connected to the perfboard, which has female headers and resistors soldered onto it. The perfboard then connects to the analog pins of the Arduino, which reads changes in voltages when the platforms are pressed. If these voltages go above a certain threshold, the Arduino emulates a keyboard key press corresponding to that platform. For example, when the player steps on the up arrow, the Arduino emulates a press on the up-arrow key of a keyboard.

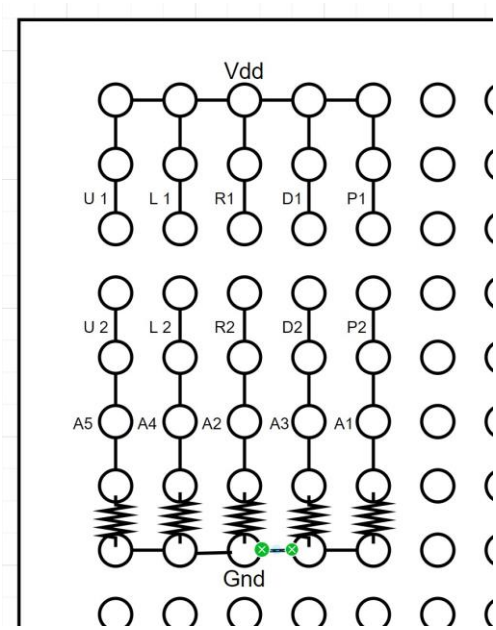

Figure 5: Perfboard pin diagram

Fig. 5 how the perfboard is connected. The nodes U1, L1, R1, D1, and P1 represent one side of the two FSRs for their respective buttons (up, left, right, down, and pause), while the U2, L2, R2, D2, and P2 nodes indicate the other side of the two FSRs for each button (recall each button has two FSRs connected in parallel to each other). Finally, the A1, A2, A3, A4, and A5 nodes connect to the analog pins of the Arduino.

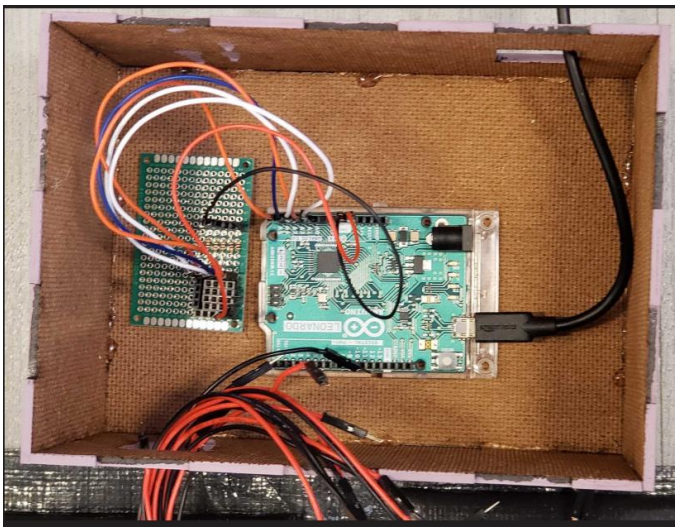

Figure 6: FlexBox with the perfboard and the Arduino

C. Raspberry Pi

The Raspberry Pi runs on the Raspberry Pi OS Lite operating system to assist in running Python code. The RPi is also set up to run the game on boot so that our system does not require a keyboard to run code and the user can simply plug it into a power source and start playing the game.

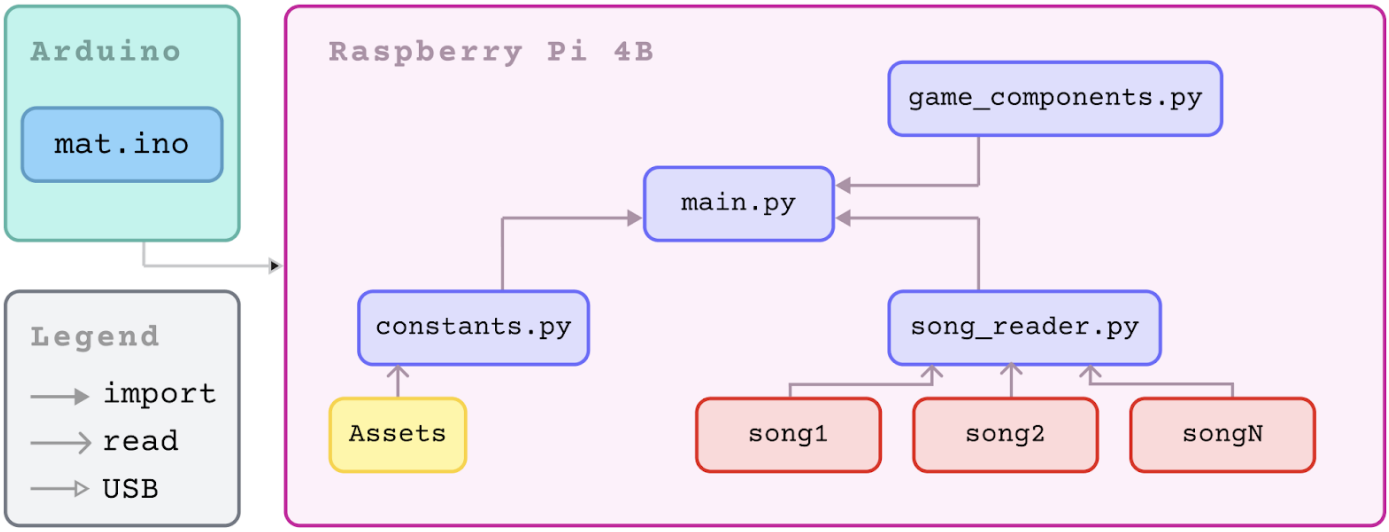

Figure 7: Raspberry Pi files and connections

Furthermore, the Raspberry Pi contains all the files listed in the block diagram in Fig 7. The game_components.py file implements the basic classes (and their subclasses) that make the game: Screen, Arrow, KeyboardKey, and AnimationHandler. A Screen object has its own drawing and updating methods to make it simple to draw and update the game through main.py. An Arrow object represents the arrows that move during the game. The KeyboardKey objects are the keys that appear in the High Score Screen keyboard. Finally, the AnimationHandler objects are queues (First in, first out structures) that store which animations are currently being played. Once an animation is over, it is dequeued and deleted.

The song_reader.py file implements a Song class that reads one of our custom soundtrack files (which are explained in more detail in the next subsection below) and stores relevant information about the song, such as title, artist, album cover, and choreography.

The constants.py file sets all the constants needed for the game, such as FPS, sounds, icons, and colors. The Assets folder stores sounds, icons, songs, and high score boards. Finally, the main.py file runs the main loop of the game, using the update and draw methods of the Screen objects from the previous files.

D. Soundtrack File Format

Each of the games' playable songs is stored in an individual folder for the song. These folders contain the files that are related to this song, including its icon, its publishing information, its MP3 sound, its scoreboard, and its choreography. Out of all these, the most involved file is the choreography file.

Rhythm games must specify a choreography (i.e., a sequence of desired inputs) for each song so that the player can try to hit those inputs at the right moment. In our case, this is a sequence of arrows. 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 is Fig. 10.

The number describes how long each measure lasts and the second number describes how many 'spots' there are per measure. The 'spots' are evenly spread across their measure and an arrow can only appear in one of these 'spots'. In the example above, each measure lasts for 3.5 seconds and there are 8 different times during the measure when an arrow could appear, each time being 3.5 / 8 seconds after the previous time. The arrows are described in order (LEFT, DOWN, UP, RIGHT)

with a 1 indicating that the arrow in that position should be pressed on that 'spot' in the measure. Using this information, 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. The sequence "1111" is special and indicates the choreography is over. We took inspiration from another rhythm game, Stepmania [8], on how to encode these files.

*let-it-go-choreo - Notepad

File Edit Format View Help

```
# Song: Let it go
# Duration of each measure
3.5
# Spots per measure
8
# Measure 1
0010 # UP
0000
0001 # RIGHT
0000
0100 # DOWN
0000
1000 # LEFT
0000
# Measure 2
0110 # DOWN adn UP
0000
0000
0000
1001 # LEFT and RIGHT
0000
0000
0000
# Measure 3
...
# Song is over
1111
```

Figure 10: Example choreography file

E. Graphical User Interface (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 (Fig. 8) 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 using the up and down arrows to decide on which one to play. The high scores corresponding to the song will also be displayed there.

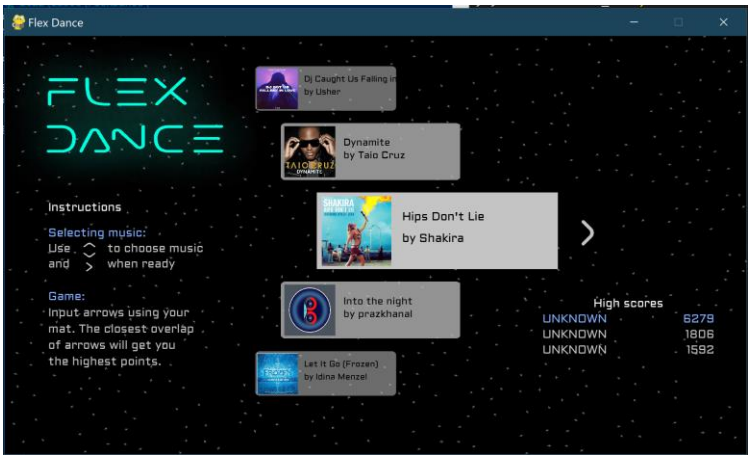

Figure 8: Menu Screen

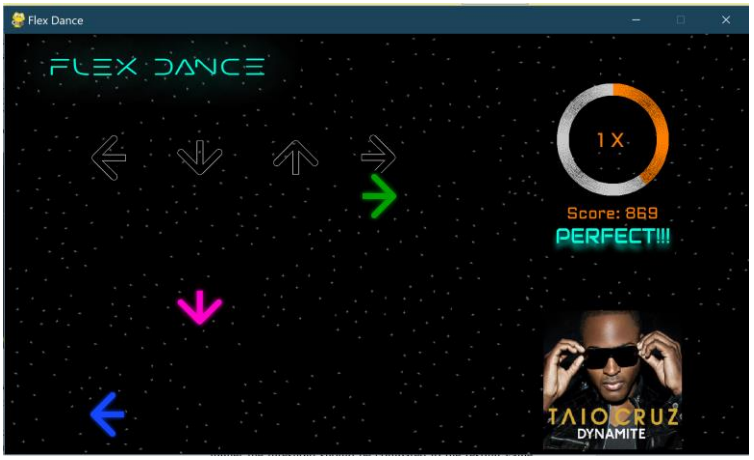

Figure 9: Game Screen

The game screen (Fig. 9) is similar to how real DDR have their arrows come in. They arrive from the bottom and travel upwards until overlapping with the arrow outlines on the top of the screen, at which moment the user needs to hit the correct arrow to earn points. The game also has a score multiplier based on how long of a combo the player has. When the player hits an arrow when it is touching the outline, they receive a base number of points between 1 and 100 depending on how accurate they were. This base number is then multiplied by the player's current score multiplier and the product is added to their total score. Also, their combo count increases by 1. At certain thresholds of combo counts, the player's score multiplier will increase. If the player misses an arrow or steps at the wrong time, their combo count resets to zero and their score multiplier goes back to 1.

Furthermore, the game screen has animations for when the player hits an arrow at the right time and for how accurate they were. Whenever an arrow is stepped on correctly, a little explosion will play behind it. Whenever the player scores points, they'll see either a "Perfect!!!" or "Great!" text pop up on the right side of their screen. Finally, if they miss an arrow, they'll see a "Miss…" on the right side. The animations are implemented using an Animation class that has its own update

and draw methods. The game screen keeps track of two queues of animations, one for arrow explosions and one for score feedback. Whenever an animation begins, it is added to the queue for that specific type of animation and updated/drawn alongside the game screen. When an animation is over, it is dequeued.

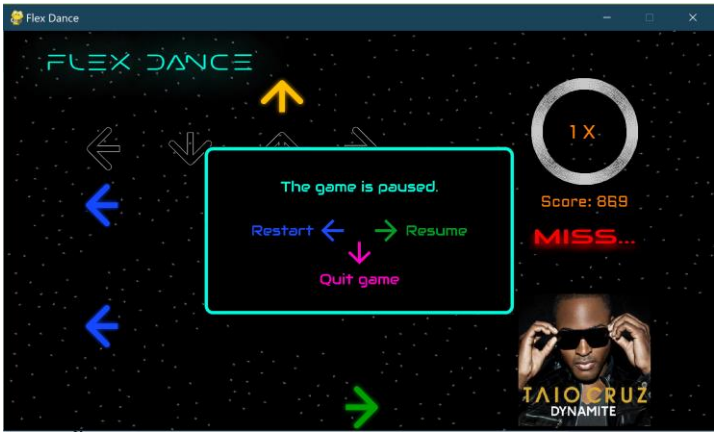

Figure 11: Paused Game Screen

During the game, players can also pause (Fig. 11) by stepping on the pause button. Pressing the appropriate arrows as shown in the pause screen will result in the corresponding actions. Particularly, these are to restart the song, resume the song, and to quit the game.

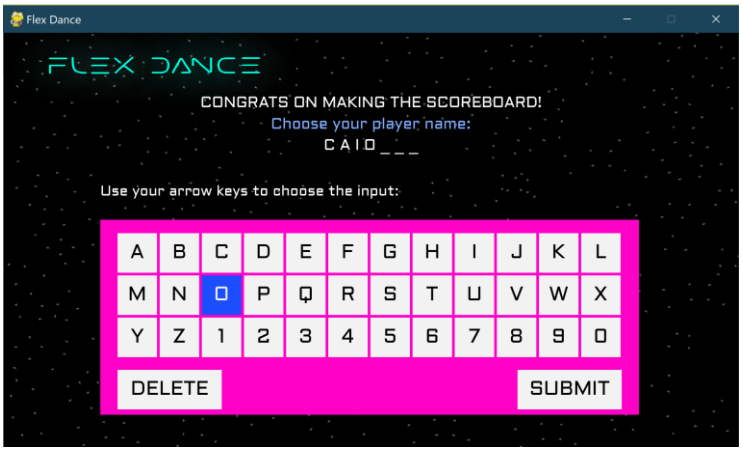

Figure 13: High Score Screen

The last screen is the High Score screen (Fig. 13) for inputting your name if you make it to the leaderboards. The player can input a name up to seven characters long. They can use the arrows to navigate to the correct symbol on the keyboard and step on the pause/confirm button to select that symbol. They can move to the "Delete" key to delete a character and to the "Submit" key to submit their name.

VII. Test, Verification and Validation

A. *Results for Mat Size*

After we finished building our mat, we used a measuring tape to measure how long, wide, and tall the mat was when folded and when unfolded.

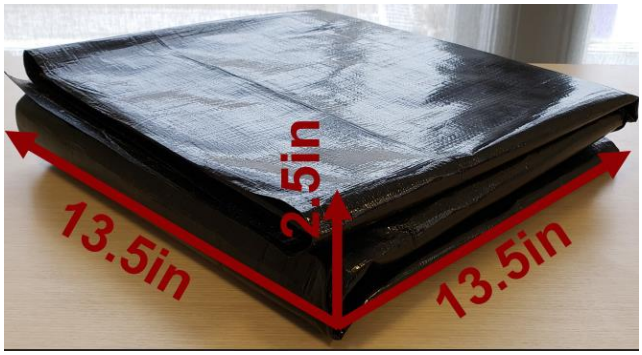

Figure 14: Folded mat and measurements

For the folded mat (Fig. 14), we measured a volume of 13.5x13.5x2.5 in^3, which is a smaller volume than our goal of 13x13x5.5 in^3 but is above our goal in terms of folded width and length. Thus, it might be harder to fit it in a drawer, and we considered this as failing the initial folded size requirement.

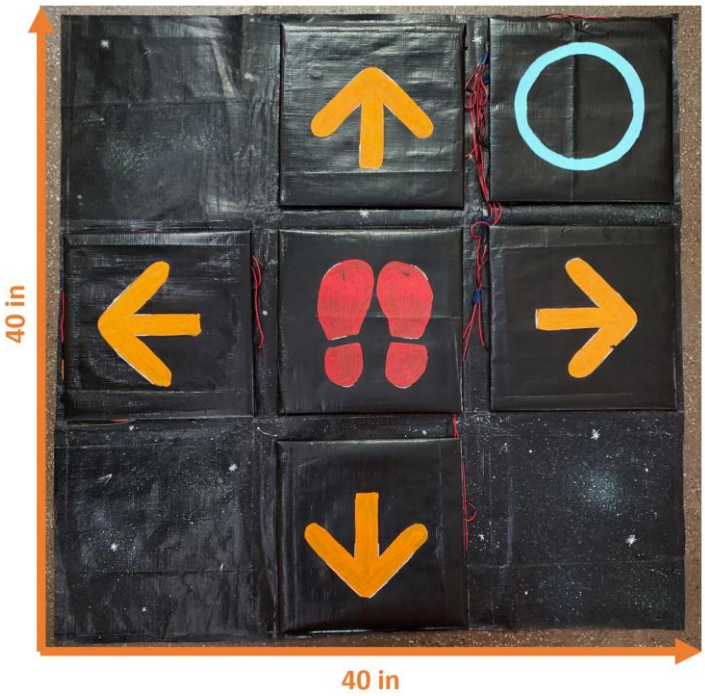

Figure 12: Unfolded mat and measurements

For the unfolded mat (Fig. 12), we measured an area of 40x40 in^2, which is also larger than our goal of 39x29 in^2. Thus, fewer living room spaces can accommodate our mat, and we considered this as failing the initial unfolded size requirement.

Despite these failures however, we do believe that the excess was small enough that it would not pose a considerable risk to the success of our product.

B. Results 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 wanted our mat to consider anything above this 10 lb threshold to be an intentional button press. This involved setting a voltage threshold for the Arduino through its code. We placed a 10-pound weight on top of the FSRs while they were connected to the Arduino and read the voltage value through the serial monitor. This gave us a rough idea of what this threshold should be. At the moment of testing, we set the value a bit higher than what we read through the serial monitor for 10 lbs. This threshold corresponded to approximately 10.2 lbs through simple linear scaling. Since the force value did not need to be exact, we were lenient with the corresponding force value. Instead, we were stricter in terms of the values on the serial monitor. This testing helped us get a better idea of how much higher the threshold should be compared to the resting value, which became valuable information for the implementation of calibration later on.

C. Results for Arrow Button Coverage

In order to measure the coverage of our arrow buttons, we had a simple choreography that was composed of 10 steps on the same arrow. We danced to this choreography 5 times, each time repeatedly stepping on the button following a different cardinal direction. For each time, we registered how many steps were correctly observed by the sensors. Table 1 shows the results.

Table 1: Number of correctly detected arrows out of 10

DIRECTION OF FOOT	# OF CORRECT
North	10
Northeast	10
Northwest	9
East	2
West	7

From these tests we learned that the vertical and diagonal directions were more reliable when stepping on the arrows. Interestingly, the west direction was noticeably better than the east direction, although we expected both to fail similarly. We believe this happened because the curvature of our foot was pressing one of the sensors when trying the west direction, but not when trying the east direction. From these results, we estimate that we have around 270° of coverage, which is only 75% of our goal of 360°. We did not achieve this requirement as well, but we found a way to mitigate it by laying out our sensors in a way that players are more likely to step on them following the vertical or diagonal directions.

D. Results for error rate

After conducting tests for coverage, we were ready to test our error rate. Noticeably, these two requirements are somewhat similar in the sense that they both refer to successfully observed steps. However, we differentiate them by measuring error rate only for steps that our arrows are capable of recognizing, hence why we conducted this test after our coverage test. Essentially, we measure error rate given a degree of coverage.

For our error rate tests, we had a simple choreography with thirty-three repeating arrows. We then played this choreography five times, making sure we always stepped in an orientation that our system was able to recognize according to the coverage test. Table 2 shows the number of missed arrows in each of the five times we played.

Table 2:Number of missed arrows in each trial out of the total 33 arrows.

TRIAL NUMBER	MISSES OUT OF 33
1	1
2	2
3	1
4	2
5	0

From our results, we calculated the false negative rate to be 4% on average. We were not particularly concerned with false positives because those were usually fixed by recalibrating our FSR thresholds.

Our initial goal for error rate was to be below 1%, so our system did not pass this test.

E. Results for latency

In order to test for latency, we first recorded a slow-motion video of someone playing the game and then used video editing software to count how many frames there were between the person applying force to a platform and seeing feedback on the display screen.

We counted one single frame between applying force and observing the feedback. The video in the video editing software was 30 frames per second and 4 times slower than real-time. So, the following equation gives us the delay:

$$Delay = 1 \text{ frame} * \frac{1 \text{ second}_{slow}}{30 \text{ frames}} * \frac{1 \text{ second}_{real}}{4 \text{ seconds}_{slow}} =>$$

$$Delay = \frac{1}{120} \text{seconds} = 8\, ms$$

Our goal was to be under 100 ms, and we certainly succeeded in this test. However, we only had the opportunity to run this test once, and we recognize that there are various factors that could influence the result, such as monitor aliasing, a coincidentally low load in the computer at that moment, human error, etc.

Regardless, we also want to emphasize that we ran playtests with 10 different people, and none of them noticed a delay between stepping on arrows and receiving feedback. As such, we consider that we qualitatively achieved this goal.

F. Results for the scoring scale

As mentioned previously, we ran playtests with 10 different people outside of our group. Each tester danced to one song and then provided us with feedback. After listening to their feedback, we asked “On a scale of 1 to 7, 1 being the lowest, how much do you agree with the score you received?”. Table 3 shows their answers.

Table 3: Ratings from 1 to 7 from play testers

TESTER	RATING	TESTER	RATING
1	6	6	3
2	6	7	4
3	5	8	6
4	7	9	5
5	5	10	6

From the table, we notice that the average rating was 5.3 out of 7, which is equivalent to 75.7% average rating. Our goal for the scoring scale was to produce a user satisfaction of at least 75%, and thus we have succeeded in achieving this goal.

G. Results for cost

When we had all of our subsystems built and properly integrated, we summed up the cost of each material and component used in the final system to find our production cost. Estimating this cost was somewhat tricky since for some materials such as the tarp and pin connectors, we did not use the entirety of what we bought. When we added the cost of the tarp, wood, EVA foam, FSRs, Arduino, connectors, and cables, we had a total of $310.76. This amount does not include the cost of the Raspberry Pi 4 and of the SD card, so the actual cost of producing a single mat would be over $400.00, which is at least double of our initial goal of producing at a cost of $180.00. Our project did not pass this requirement.

However, in a real-world scenario in which Flex Dance was produced in bulk, we would have a smaller cost for a few reasons. First, our mat needed ten 1-foot long FSRs, but we could only purchase 2-foot long FSRs. We ended up cutting them in half, and essentially wasted half of the money spent on these FSRs. Should these mats be produced in bulk, it would be possible to place bulk orders of 1-foot long FSRs, which should reduce the cost by around $100.00. Second, large materials like tarp, wood, and EVA foam could be purchased in bulk and receive discounts. Third, we could create cheaper, custom ADCs to replace the Arduino and build dedicated hardware to run the game instead of using the RPi. As such, it is possible that the production cost of Flex Dance is actually close to $180.00 depending on the manufacturing scale.

VIII. Project Management

A. Schedule

Since the design report, the new schedule (Fig. 15) should reflect major changes in the buffer period. In particular, our schedule to start construction of the mat got delayed by a little more than a week. We used the buffer to account for this extra time. As we worked on the mat, we also realized that constructing everything needed for the mat and combining the layers would require more than 3 weeks. The buffer helped us do that. Lastly, we introduced testing during the last two weeks. This also aided us in finding reliability issues and fixing it with calibration of the FSRs. Any extra tasks we did are also listed in the schedule attached.

B. Team Member Responsibilities

All team members contributed heavily to get ready for the demo and meet the deadlines. Some of the specific tasks can be found listed in the schedule. Specifically, the area of expertise along with the specific responsibilities are given below:

1) *Spandan Sharma:*
   She was the presenter for the Proposal presentation and is handling the software systems and UI design. She worked on creating the menu screen in Pygame, designing a user-friendly interface, painting the mat and Flexbox, and creating the Final Video.
2) *Tushhar Saha:*
   He was the presenter of the Design Review presentation and is handling circuits of the game. He worked on the FSRs, their calibration, the Arduino code, designing and soldering the perfboard, and the making of the Flexbox.
3) *Caio Araujo:*
   He was the presenter of the Final presentation and is handling the software systems. He was responsible for creating the game screen in Pygame, high score screen, pause screen, arrow tracks for the songs, and setting up the Raspberry Pi.

Everyone worked together to make the physical mat and combine the various layers. Every team member was involved with user testing as well. For the other deliverables like the posters or presentations, it was generally a team effort from everyone.

C. Bill of Materials and Budget

Our bill of materials is attached in the document in Fig. 16 and reflects the components we ended up not using (highlighted blue), as well as new components we added since the design report (highlighted green).

D. Risk Management (used to be Risk Mitigation Plans in Design Document)

There were several problems we encountered as we started building the mat. This section will talk about the two major issues that we ran into and how we solved them.

As mentioned before in the design tradeoff section, we ran into issues of durability with the Z shapes for our FSRs. We found that after a couple of uses, the parts after the folds did not respond. Therefore, we tested our various configurations and got the most satisfactory results with 2 parallel FSRs per plate. Dealing with this issue was important as it is what determined how well our game responded to steps. We ended up losing a bit of coverage, but we got much more durability out of it. Furthermore, using 2 FSRs per plate would mean increasing our costs. As a result, we decided to cut down on 1 button and have 5 instead. This would help cut down costs and save some money in the budget in case we needed to make any other changes.

The next major risk we ran into was unreliable responses from the FSRs after stepping on the arrows a couple of times. Our playtesting with users helped us notice this issue, and we were able to dedicate the last week of buffer to fixing it. We noticed that any thresholds we set based on resting state needed

to be changed at a regular interval in order to get more reliable results. This was because the resting state values of the FSRs changed after stepping on it. This may be because of the plates sticking a bit more to the FSRs. Whatever the reason was, it was an unforeseen circumstance. We diagnosed the problem and dealt with it by adding calibration of the FSRs based on their current resting state values. Calibration helped set the thresholds based on these values and was made to run whenever the game was switched on. We also added a button to manually trigger the calibration procedure so that we can press it between games. This helped us get much more reliable results.

IX. Ethical Issues

Generally, a foldable mat designed for dancing is not something we expect to harbor serious ethical problems. That being said, some of the ethical issues we considered was how this product could affect users in ways they might not expect it to.

As engineers of this mat, we need to account for the potential safety issues with the mat including possible slippage while dancing that could result in accidents. While we made sure to use materials that won't slip as much, there is always a slight risk that remains. Generally, we hope that the user will be careful while dancing on the mat. Additionally, we would add some fastening mechanisms if we had a little more time to work on the mat. Potential options include velcro that can be attached to all four corners of the mat and the floor.

Designing a mat that can be used in small living places like an apartment implies that this mat won't just be affecting the user but also the people around them. Our mat requires users to step on the arrows quite firmly, and it also works best with speakers. As a result, it can lead to lots of loud noises like thuds and sound from the game. We haven't addressed this issue in our design, but we do recognize it to be relevant upon seeing people play with it during user testing. We wouldn't want the users' neighbors to be disturbed and complain about the sounds. In another iteration of the design, this is something that can be reduced by using materials in the mat that absorb sound and distribute it away from the floor. Introducing methods like Bluetooth to listen to the music through earbuds would also reduce noise emitted.

Lastly, our game is not suitable for people with walking disabilities or who lack fast reflexes. We tried to be as inclusive as possible so that even old people or children can play. However, there are only so many users we can cater to. Ideally, we would like our game to be more accessible to different people.

X. Related Work

Our game is heavily inspired by a classic arcade game called Dance Dance Revolution (DDR). In DDR, players follow dance steps by stepping on directional arrow buttons (up, down, left, and right) according to the rhythm of whichever song they chose to dance. These buttons are laid out on a metal platform. Although DDR is a good exercising tool, its purpose leans more towards entertainment rather than fitness. Also, a DDR machine is too expensive and too large to be considered as an at-home exercising alternative. As such, although we based our game on DDR, we do not believe that DDR is one of our direct competitors.

There are, however, other at-home exercising options that are alternatives to 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 that 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 DDR and 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 extravagant 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.

XI. 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. Our journey of creating this game was filled with milestones and obstacles which we will briefly summarize below.

The first challenge that we overcame was latency. To reduce latency, we meticulously coded our software to minimize unnecessary simultaneous actions (such as clearing data between each screen after drawing images, reading USB input, and playing music) which can all slow down the software, especially in a small device like the Raspberry Pi. As talked about in our testing, our users were satisfied with the responsiveness of the mat and our own qualitative testing also yielded good results.

The second challenge was coverage and durability. After trying multiple options such as the "Z", "+", and "x" shape, we settled on using two parallel force sensitive resistors that were both durable and provided practical coverage of a user's step. Though the buttons provided 270° out of 360°-degree coverage, our usability testing proved that the users were still mostly satisfied with our game.

Our last critical challenge was staying within our desired budget. The main components that were expensive were the Raspberry Pi and the force sensitive resistors. Though our overall cost came out to be more than we aimed for, we now know that through bulk ordering and using different alternatives for our game components (such as using an old Raspberry Pi or using different sensors) could greatly reduce our cost of the product.

In conclusion, our project was a fun and learning experience, both academically and personally. We learned how to effectively communicate, how to meet deadlines, and lastly, how to enjoy ourselves while reinventing a product to fit popular needs.

Glossary of Acronyms

DDR – Dance Dance Revolution
RPi – Raspberry Pi
FSR – Force Sensitive Resistor
ADC – Analog to Digital Converter
FPS – Frames Per Second
GUI – Graphical User Interface
UI – User Interface
UX – User Experience
EVA – Ethylene-Vinyl Acetate copolymer
DVI – Digital Visual Interface

References

[1] Babich, N. (2022). 5 Simple Tips On Using Color In Your Design. Medium.
[2] Dance Dance Revolution Arcade game for sale- Vintage Arca. Vintage Arcade Superstore. (2022).
[3] Dance Dance Revolution for Wii, PS2, PS3, Xbox 360, & PC. Ddrgame.com. (2022).
[4] Healthline, *Average shoe size for men*, healthline.com (2019)
[5] Healthline, *Average shoe size for women*, healthline.com (2019)
[6] 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.
[7] NextFab, 'Making a USB Game Controller', https://www.instructables.com/Making-a-USB-Game-Controller/
[8] News - StepMania. Stepmania.com. (2022).
[9] Nintendo Switch Gray Joy-Con Console Set, Bundle With Just Dance 2022, Walmart.com. (2022).
[10] Pete Shinners, et al. (2011). PyGame - Python Game Development. Retrieved from http://www.pygame.org

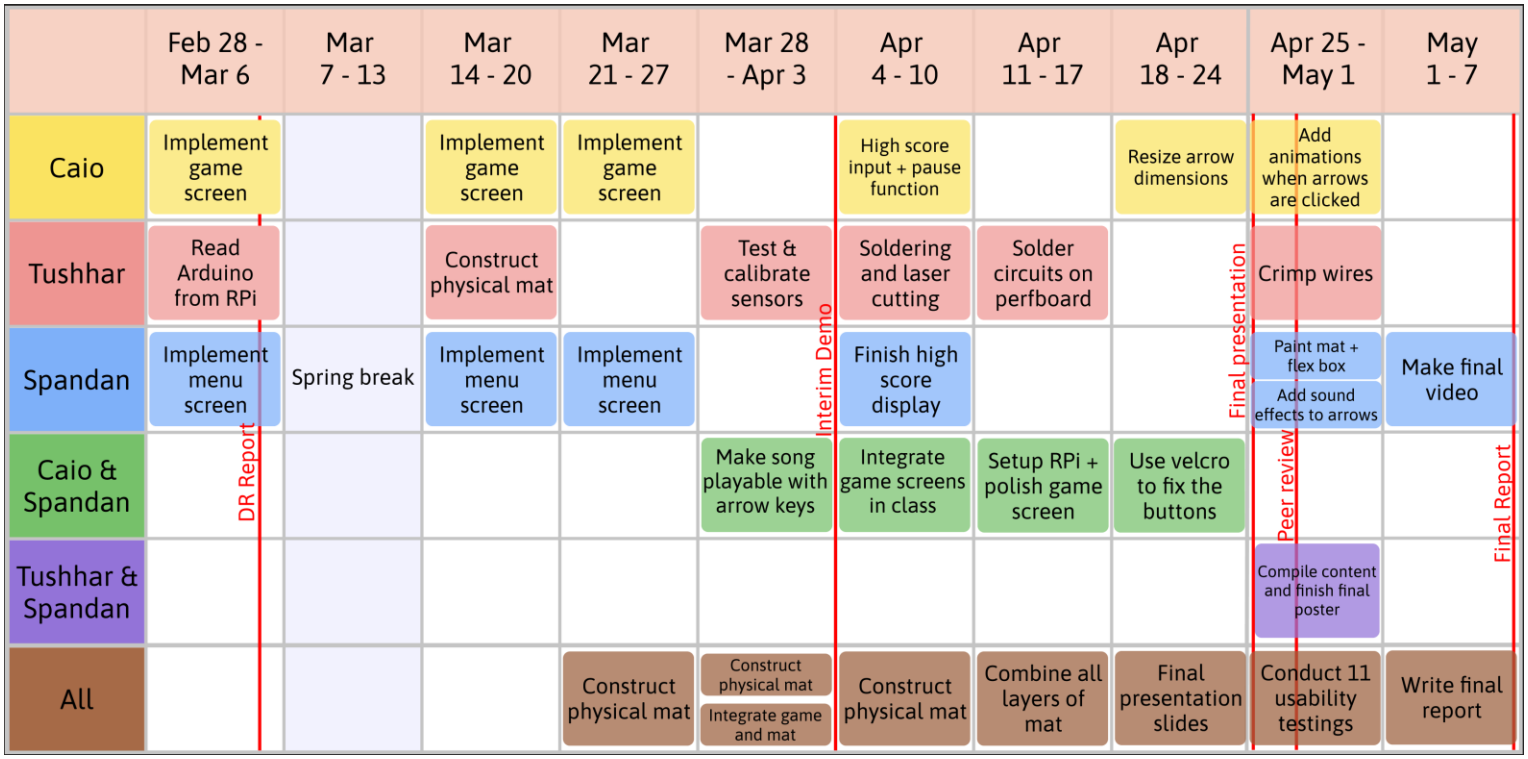

Figure 15: Final schedule

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	Headered Arduino was better suited for our needs
Arduino Leonardo with headers	Arduino	1	$24.14	$24.14	Arduino Store	Used for testing before assembly of mat
2 ft Force Sensitive Resistor	Interlink Electronics	12	$17.96	$228.48	Adafruit	Will act as the force sensors to be stepped on
Perfboards	ELEGOO	1	$12.99	$12.99	Amazon	Will have the circuit for the mat soldered
Male and Female Connectors	Any	1	$8.98	$8.98	Amazon	MF Pin connectors for connections to perfboards
Housing for connectors	Any	1	$9.50	$9.50	Amazon	Housing for connectors above
Masonite Boards 1/8"	Any	11	$1.50	$16.50	Techspark	For plates and box
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	Paint was too wet. We used normal paint from techspark.
Thread	Unique Craft	1	$12.99	$12.99	amazon	We ended up using velcros to attach the plate.
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.
Velcro	Amazon Basics	2	$8.69	$17.38	Amazon	Velcro pieces to create pockets to easily store the buttons in the tarp
			TOTAL	$423.03		
			Amount Left	$176.97		
Components not used:						
Components added after Design Report						

Figure 16: Final BOM