PRINCETON UNIVERSITY

Dance Studio:
A Choreography Simulation Tool

Author:
Janet LEE

Advisors:
Dr. Robert DONDERO
Dr. Szymon RUSINKIEWICZ

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF ARTS
DEPARTMENT OF COMPUTER SCIENCE
PRINCETON UNIVERSITY

May 2020

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Janet Lee

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Janet Lee

Acknowledgements

This thesis would not have been possible without the love, care, and support of many individuals, for whom I am extremely grateful.

To Professor Dondero and Professor Rusinkiewicz, thank you for your kindness, support, and guidance this past year. Although the road to completing this thesis was certainly not an easy one, I am grateful for your patience in every Thursday meeting, both in person and on Zoom, and for your continual encouragement that always made this project more of a delight than a burden. Professor Dondero, thank you for believing in me and for always challenging me to produce high-quality work and to relentlessly debug for the sake of the user. Professor Rusinkiewicz, thank you for mentoring me through unfamiliar technologies, and for your continual reality checks and humor that helped me stay on track. Your support exceeded my expectations, and I am so blessed to have received so much wisdom from not one, but two, advisors.

To Six14 and Naacho, the communities that inspired this project, thank you for introducing me to the world of dance. My four years at Princeton would not have been the same without the countless late night rehearsals, Wa runs, freezing photoshoots, and Heaven weeks in Frist. You have shown me the joy of dance and have been families to me on campus, and for that, I am so thankful.

To Phillip, Jessica, Moses, Josh C, Buff, Joane, Jinn, Eunice, Esther, and Amber, thank you for giving your time and energy to test my application. Whenever I felt downcast, your responses to using Dance Studio encouraged me to continue working. Without you, this thesis would not be possible.

To my dear friends, thank you for the most memorable four years at Princeton. To my Spelman 76 cohabitants, it's been a wild ride - I miss our thesis crying parties, oat milk obsessions, and mutual lack of respect for personal space. You have truly made Spelman into a home away from home. To my brothers and sisters in Alabaster Group and Manna Christian Fellowship, I am so grateful to have been able to run this race together, sharpening one another as we pursue Christ. To the Sunday prayer group, I can so clearly see God's abundant love through your selfless desire to pray for me and my family in this final semester, and for that, I give thanks. To my thesis fairies, Marisa, Rachel, Joice, and Lillian: thank you for your snacks, prayers, and words of encouragement that have brought so much delight and joy amidst many trials. To Min Ji and Grace, my Zoom study buddies, thank you for years of friendship that I am sure will continue far beyond Princeton; this is certainly not goodbye.

To my parents, thank you for loving me and for providing for me so sacrificially every single moment of my life. I am so deeply indebted to you, and while I cannot pay back all that you have given to me, I hope to one day love my children the way that you love me. I love you!

Last but certainly not least, thanks be to God my Father. I thank you for all of the people whom I mentioned above, since, after all, you created us. It is through you and for you that I dance, through you and for you that I write this thesis, and through you and for you that I do all things. Thank you for your patience, provision, and unconditional love every single day of my life. I dedicate this thesis to you. *Soli deo Gloria!*

And David danced before the Lord with all his might.
-2 Samuel 6:14a

Abstract

In dance, blocking formations is a critical aspect of choreography; however, there is currently no way to visualize how dancers will look and move onstage in real life. This problem often makes choreography a burdensome and intimidating task and stifles creativity. This paper details the design, implementation, and evaluation of Dance Studio, an online choreography simulation tool that solves this issue by enabling users to visualize and animate their formations with 3D computer graphics. In the application, users can upload music, play the dance back to see transitions between formations in real time, and anticipate collisions between dancers. Most notably, unlike other dance applications, Dance Studio offers users the ability to control dancers' movement by creating custom non-linear paths. Through technical analysis and user evaluation, I demonstrate that Dance Studio successfully provides a medium for choreographers to translate their ideas from their mind, to the screen, to the stage.

Contents

List of Figures

List of Tables

1 Introduction

Dance is an exciting medley of physical strength and artistic expression; for many, the process of choreographing is enjoyable, until one must consider how to position dancers onstage. Otherwise known as "blocking," setting formations can be quite tedious and challenging because it is difficult to visualize how dancers will move onstage in real life. Many college dance routines last an average of 3-5 minutes, containing anywhere from one to twenty or more dancers performing in the same space. While filming oneself dancing is an effective way to remember the dance moves for the routine, there is no simple solution to keeping track of twenty dancers' positions for every second in a routine. The goal of this project is to resolve this issue by assisting users with creating choreography.

To better understand how choreographers currently organize their formations, I surveyed ten student dancers at Princeton. The majority typically keep track of formations by drawing out each position individually with pen and paper, representing individual dancers with circles or their initials. While this is an intuitive and quick way to jot down one's thoughts, the pen and paper method has several limitations. First, because the choreographer must visualize each formation change mentally, it is difficult to anticipate collisions between dancers in between formations. Often, these collisions only become apparent when dancers are actually learning the dance in rehearsals and end up conflicting with each other while transitioning between formations. At best, this scenario wastes precious rehearsal time, but at worst, it can result in dancers' injury. Secondly, modifying the formations requires erasing and redrawing the formations on paper, which is time consuming, possibly discouraging choreographers from freely experimenting with their dancers' positions. Additionally, indicating where each formation is in the music cut proves somewhat cumbersome - the choreographer must write down a particular song lyric, a timestamp in the music, or a specific 8-count to indicate when a formation starts and ends. Finally, losing the physical sheets of paper often results in the dance's formations being lost altogether.

The next most popular method of documenting formations is via Google Slides, with formations being represented by slides with initials on them. One benefit of Google Slides is its ac-

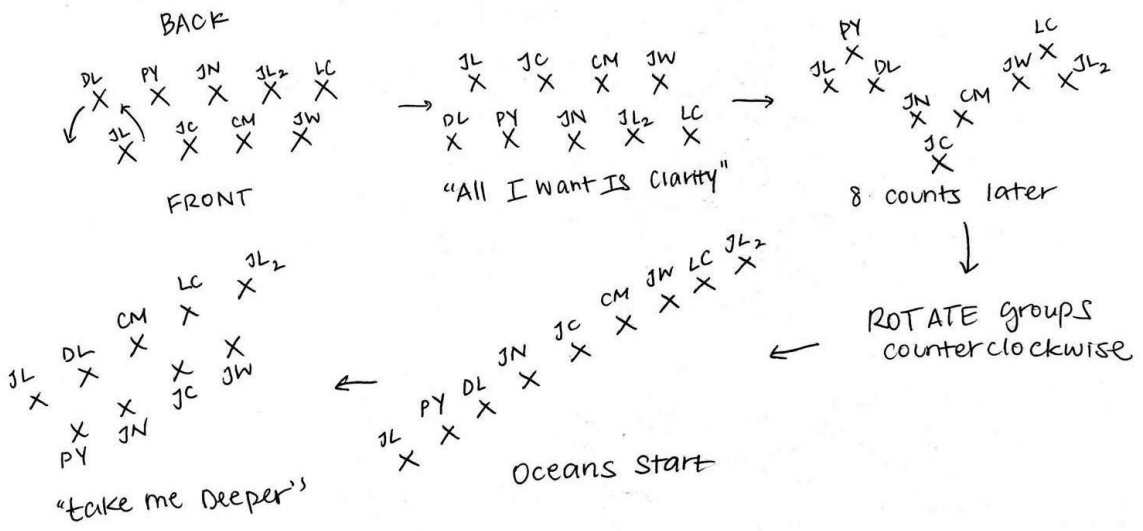


Figure 1: Formations documented on pen and paper.

cessibility; unlike pen and paper, users are able to access the platform from anywhere with an Internet connection, and Google Slides enables multiple users to view and edit the same file simultaneously. Additionally, this digital method is more persistent in that users are less likely to accidentally delete a Google Slides file than they are to lose a piece of paper. Despite these advantages, however, Google Slides still suffers from the same limitations of not allowing the user to visualize transitional movement between formations in time with music.

Lastly, one choreographer told me that the process of documenting formations was too cumbersome altogether, so he simply creates formations on the spot during rehearsals. His feedback coupled with the insights from other dancers demonstrated a clear need for a better way to document formations. As a dancer and choreographer myself, I wanted to develop a solution to this problem to simplify the choreography creation process.

Dance Studio is a choreography documentation and simulation tool that allows choreographers to create, edit, and visualize their formation changes on a 3D model of the dancers and stage, mapped in time to music. By eliminating the limitations and hassle of keeping track of dancers' positions with hand-drawings, Dance Studio encourages choreographers to experiment more with

incorporating creative formations into their routines.

The main features of this tool are: 1) users can easily create custom formations on a 3D graphical interface, 2) users can upload their own audio file for each dance, 3) users can visualize their formation changes animated in real time to music, 4) users can proactively anticipate and work around collisions between dancers by creating non-linear paths between formations, and 5) dances are saved and accessible online at any time from anywhere.

Dance Studio is most suited for dance styles with the following characteristics: 1) having no more than twenty dancers onstage at a given time, and 2) having distinct formations, meaning dancers are not constantly moving, and there is a distinction between stationary positioning and transitional movement. Some examples of dance styles that meet these criteria are hip-hop, lyrical, Bollywood, Bhangra, and Raas.

On the basis of user studies, Dance Studio accomplishes its goals, as demonstrated by all users giving the application positive feedback and preferring to choreograph on the platform to their old method. On average, all users were extremely likely to use the application to choreograph dances in the future.

Figure 2: A screenshot of the Dance Studio user interface [21].

2 Related Work

Several dance formation tools and choreography simulators already exist; however, few are available for free, and of the options that are available publicly, the usability is extremely limited. In this section, I give an overview of other applications that seek to accomplish the same goal.

2.1 Danceapp.us

Danceapp.us is the most similar application to my project, in that it looks like a video editor with a stage and allows the user to map individual dancers onto various places on stage at different points in time, as seen in Figure 3 [4].

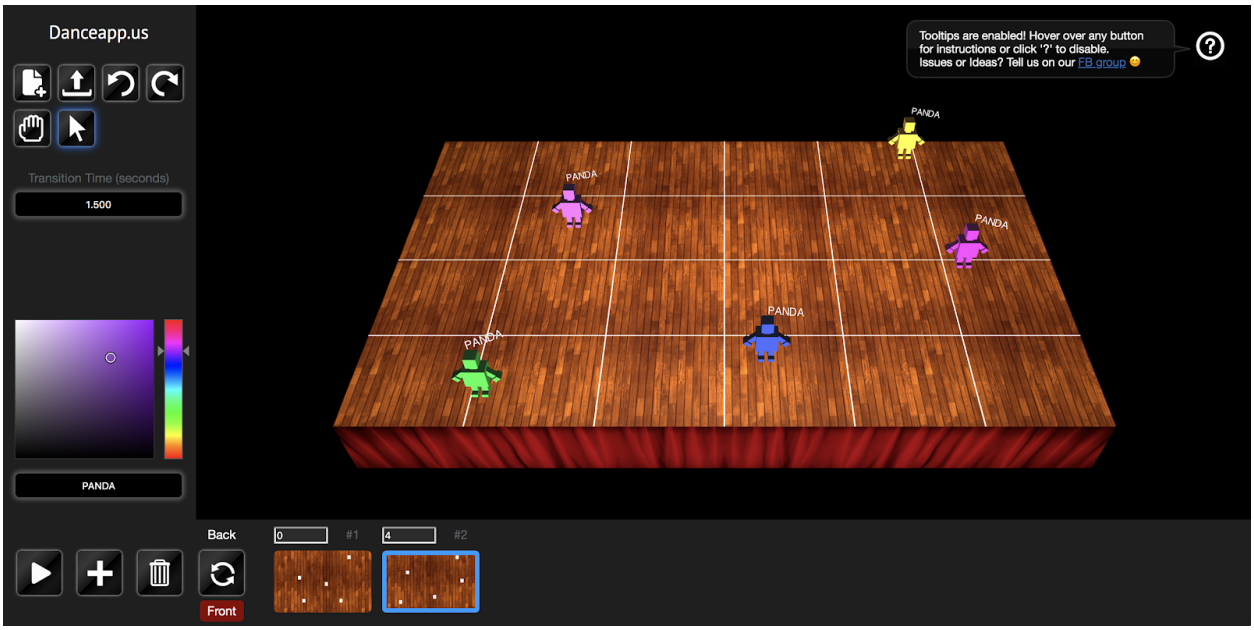


Figure 3: A screenshot of the Danceapp.us user interface [4].

Danceapp.us helps users visualize dances, allowing the user to play back the dance routine with animations that show transitions between formations. However, this application is limited in that it does not allow users to upload music, and it is rather clunky in its appearance. When I tested this application with student choreographers at Princeton University, every single test user was unsure of how to create a new formation to the routine, and it took them several minutes to become accustomed to the user interface. Additionally, the dancers all move in a linear path by default,

so users are unable to create more elaborate dances with non-linear paths. Furthermore, while Danceapp.us is easy to access using any web browser at `http://danceapp.us/`, this application does not allow users to save and retrieve their dances, so practically, it is less persistent than using pen and paper.

2.2 StageKeep

Another app on the market is StageKeep [23]. While its aesthetically pleasing user interface offers more features than Danceapp.us, allowing the user to upload music and visualize their formations with an actual timeline, there are still several limitations. First, the dancers are represented by 2D circles from an aerial view of the stage, rather than in a 3D interface that allows users to change the camera view. Distinguishing between dancers is not very clear because all dancers are circles with the same color, unlike in Danceapp.us, where dancers have customizable colors and names. Additionally, like Danceapp.us, StageKeep can only linearly interpolate between positions for dancers' transitions, meaning that it does not offer users solutions to resolve dancers' collisions.

StageKeep's biggest barrier to entry is its price point - while it is free to download the application, the user must pay a minimum of $9.99/month to gain full usability of the application, with unlimited formations, dancers, custom music, and cloud saving. The most expensive plan is $299.99/month, allowing additional users to share and edit the same dances [23]. Most choreographers, especially novices exploring dance for the first time, are unwilling to commit to paying for a choreography software at such a high price point. As a result, it is not a feasible nor a practical solution to simplify the choreography process.

2.3 Comparison

While both of these applications certainly improve upon certain aspects of using pen and paper to traditionally plan out formations, they both fall short in not offering users enough flexibility or ease of use. Dance Studio, my implementation of a choreography simulation tool, combines the most valuable components of each of these different pieces of related work, while improving upon

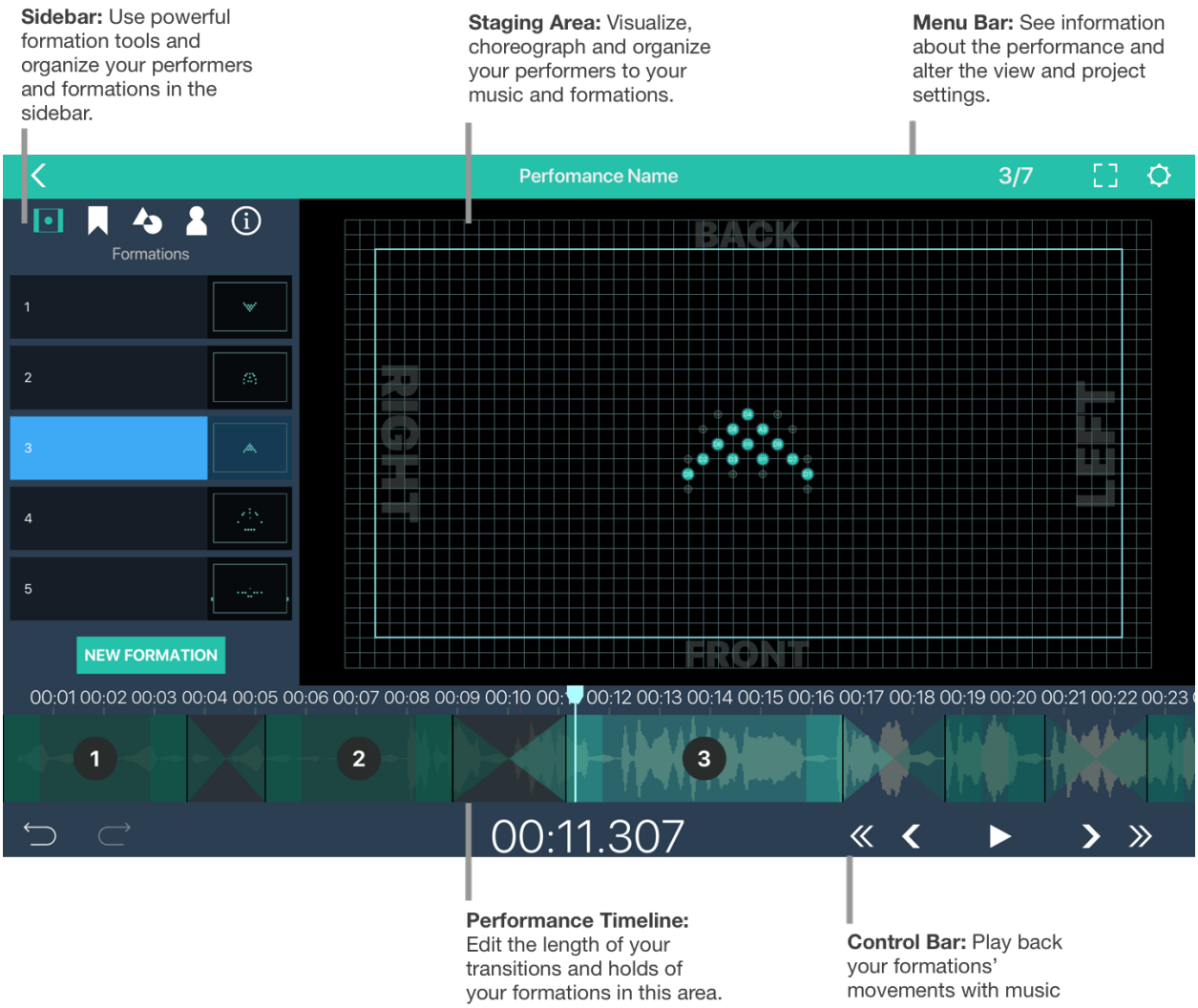


Figure 4: A screenshot of the StageKeep user interface [7].

their shortcomings. The design prioritizes user experience, relying on constant feedback from real choreographers who test the application and relay what was most effective and what could be improved. My project aims to improve upon the existing frameworks in all areas, as detailed in Table 1.

Feature	Means of Formation Planning			
	Pen and Paper	Danceapp.us	StageKeep	Dance Studio
Free	✓	✓	✗	✓
Allows users to play back their routine	✗	✓	✓	✓
Users can upload music	✗	✗	✓	✓
Users can save and retrieve their work	✓	✗	✓	✓
Quality User Interface	✗	✗	✓	✓
Has a 3D rendering of the dance	✗	✓	✗	✓
Provides solutions to proactively resolve collision avoidance	✗	✗	✗	✓

Table 1: A table comparing qualities of different methods of documenting dance formations.

3 Approach

Dance Studio is designed to provide a flexible framework that can accommodate both experienced choreographers and novice dancers. As a result, to understand what features would be important to include for both types of users, I interviewed a variety of dancers at Princeton, some who were more experienced choreographers able to mentally visualize formations, and other more novice members of the dance community who have never choreographed before. In addition to conducting quantitative interviews, I went through mock testing sessions using existing applications on the market (DanceApp.us and StageKeep) to identify users' frustrations. On the basis of interviews, users appear to desire an application with the following characteristics:

- **Users can see animated transitions between formations.** This is the core functionality of the application that distinguishes it from pen and paper, choreographers' current medium of documenting formation changes. Choreographers struggle to visualize formation changes and would greatly benefit from the ability to see a moving, animated version of their dancers.

- **Users can upload music and play the dance in real time with music.** Choreographers generally create formations based on their music, whether it is with 8-counts or with the song lyrics. As a result, hearing the music as one sees the formations is necessary to accurately create formations with the correct time stamps.

- **Users can save and retrieve their work through a persistent web application.** Choreographers expressed that they would like to have the option to access their dances at any time, from any place without the risk of misplacing it like pen and paper, and a system that allows them to revisit previously created dances. Hosting an application on the web offers users this convenience.

- **Users can make a basic dance in under a minute.** The benefits of creating an animated dance are limited by the cost of adjusting to a new user interface. As a result, the application must be easy to use and intuitive to users, much like drawing formations with pen and paper.

- **Users can customize dancers' names and colors.** To avoid confusion between dancers, and for the sake of dancers in the piece learning their positions in the routine, it is imperative to be able to clearly distinguish between dancers. When using pen and paper, choreographers typically identify distinct dancers by their initials for shorthand. However, this becomes problematic when multiple dancers in the same routine have identical initials. As a result, personalization with full names visible for each dancer as well as distinctive colors minimizes confusion between dancers. Furthermore, as an alternative use, choreographers can experiment with costume color schemes by customizing the colors of their dances.

- **Users can not only visualize dancer collisions during transitions between formations, but also create non-linear paths for dancers to avoid collisions.** This feature was by far the most requested by the surveyed potential users. The primary limitation in drawing formations with pen and paper is not that formations are difficult to visualize, but that choreographers are unable to anticipate collisions with 100% accuracy. Typically, dancers realize there is a point of collision during rehearsal and resolve the conflict by explicitly defining their paths. Both may take winded paths to travel to the next position, or one person may delay or accelerate their movement to avoid being in the same space concurrently. The proposed application will replicate this behavior by providing the user the option to create curved paths as well as manipulate the speed of the dancer as she travels along the path. In addition, the option to create non-linear paths also has applications beyond collision avoidance - in many styles of dance (modern, lyrical, traditional South Asian dance, etc.), dancers move in curved paths, sometimes even moving in a circle. This feature would allow choreographers of all styles to visualize their dances.

In addition, some users specified they would like to have the following features:

- **Users can change the camera view.** Choreographers typically plan their formations from an aerial view, but they would like to ensure that their dance is equally impressive from the perspective of an audience member, both in the center of the house and in the seats on the

edges of the auditorium. Additionally, a mobile virtual camera would provide dancers an onstage perspective. This feature necessitates that the application render the dance in three dimensions.

- **Users can change the playback speed of the routine.** Choreographers may want to speed through slow transitions and long motionless sections in their piece. Alternatively, they may need to slow down the dance routine during fast-paced transitions.

The user interface of Dance Studio is inspired by existing tools such as Blender, a 3D modeling software, as well as iMovie, which is a popular video editing tool with a similar GUI, with the screen partitioned into a bottom timeline section, a main video editing section, and a small side panel for additional features as seen in Figure 5 [19, 6]. These applications are commonly used for video editing and 3D animation.

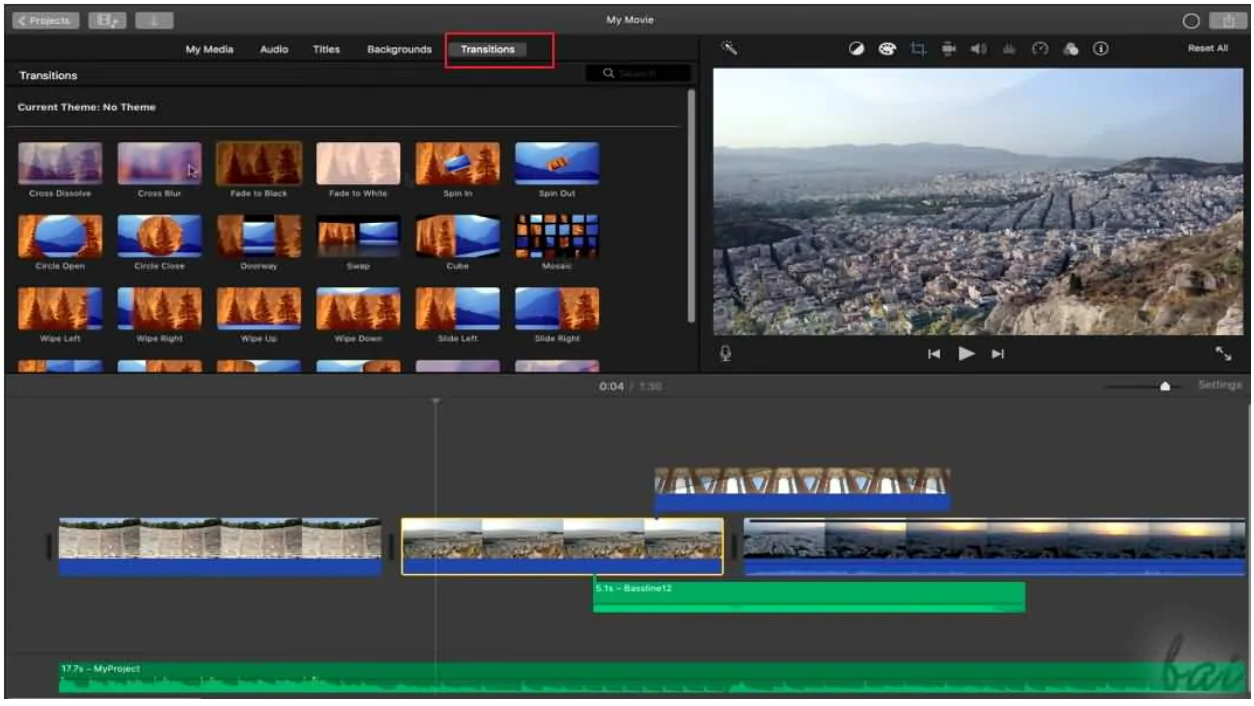


Figure 5: A screenshot of the iMovie user interface [5].

In general, my project relies centrally on a timeline in the editor, allowing the user to have greater control over where in the song to put various key frames representing different dancers' formations. A timeline is extremely beneficial to choreographers in pinpointing exactly where in

the song a formation change occurs. Additionally, having the ability to play the song back from a set point in time is a key component of Dance Studio, allowing choreographers to actually practice the dance repeatedly with music, rather than without sound.

4 Implementation

Dance Studio is a three-tier application that allows users to log in with a Facebook or Google account, create full-length dance routines with any number of dancers, upload music (with a visualization of the audio on the timeline), and save their dances to their account to be accessed at a later time. The user is able to "undo" moves, delete keyframes, and adjust keyframes along the timeline.

4.1 System Architecture

The structure of the application is divided into three tiers. The first tier is the graphical user interface in the form of a website. The second tier is the server processing tier. Its client-facing interface is a RESTful API that accepts GET or POST requests when the client would like to view data currently stored in the database or would like to update data in the database. Lastly, the final tier is the database, which stores relational information about users - namely, their dances and login information.

The application is deployed at `https://dancetigers.herokuapp.com/` via HerokuApp, which was chosen because of my prior experience with the service, its free pricing tier, and its convenient compatibility with Postgresql. HerokuApp's in-house Postgres database service, Heroku PostgreSQL, enables

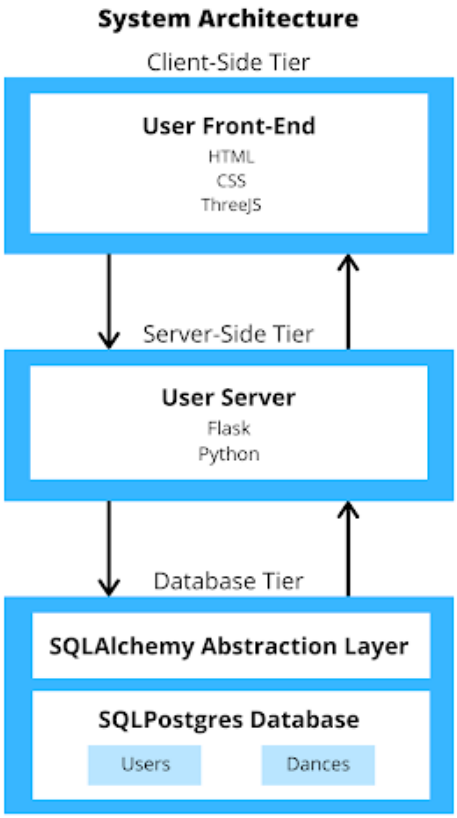


Figure 6: A diagram of Dance Studio's system architecture.

developers to create and manage their application databases that run continuously online [10]. To set up the system, I followed Sahil Diwan's blog post "Making a Flask app using a PostgreSQL database and deploying to Heroku" [13].

4.2 Database

The PostgreSQL database consists of two tables: Users and Dances. The Users table tracks Dance Studio users, based on the information retrieved from Facebook or Google login, while the Dances table contains all dances created by all users of the application. For a detailed database schema, see the Appendix, Part A.

4.3 Server

The back-end server is implemented with Python Flask. Clients interact with the server through a RESTful API. Since the application is structured around the relational nature of the database, GET and POST requests to the server are sufficient to access or modify data without having a stateful connection to the server.

SQLAlchemy is a Flask extension used to facilitate communication with the database [11]. Endpoints that update or retrieve data from the database with SQLAlchemy use a secret key to maintain the integrity and limit the visibility of private data. The server interfaces with Flask-Login, a Flask extension that facilitates user session management, to authenticate users to the server [8]. On receiving requests to update the User table of the database or the Dances table of the database, it makes the appropriate call via SQLAlchemy. It also returns objects representing dances in a JSON format to the client to render a particular user's dances.

4.4 Front-End

The front-end of the website is built with Bootstrap 4, HTML 5, CSS 3, and JavaScript. I chose Bootstrap 4 because of its smooth integration with HTML5 and built-in responsive features. Addi-

tionally, all of the graphics computation is accomplished through Three.js, a JavaScript 3D library that is used to create and display animated computer graphics in web browsers [14]. Three.js relies on WebGL, a JavaScript API, to render its graphics on web browsers without the use of additional plug-ins [16]. The user interface exclusively sends GET and POST requests to the user server.

4.4.1 Homepage and Login

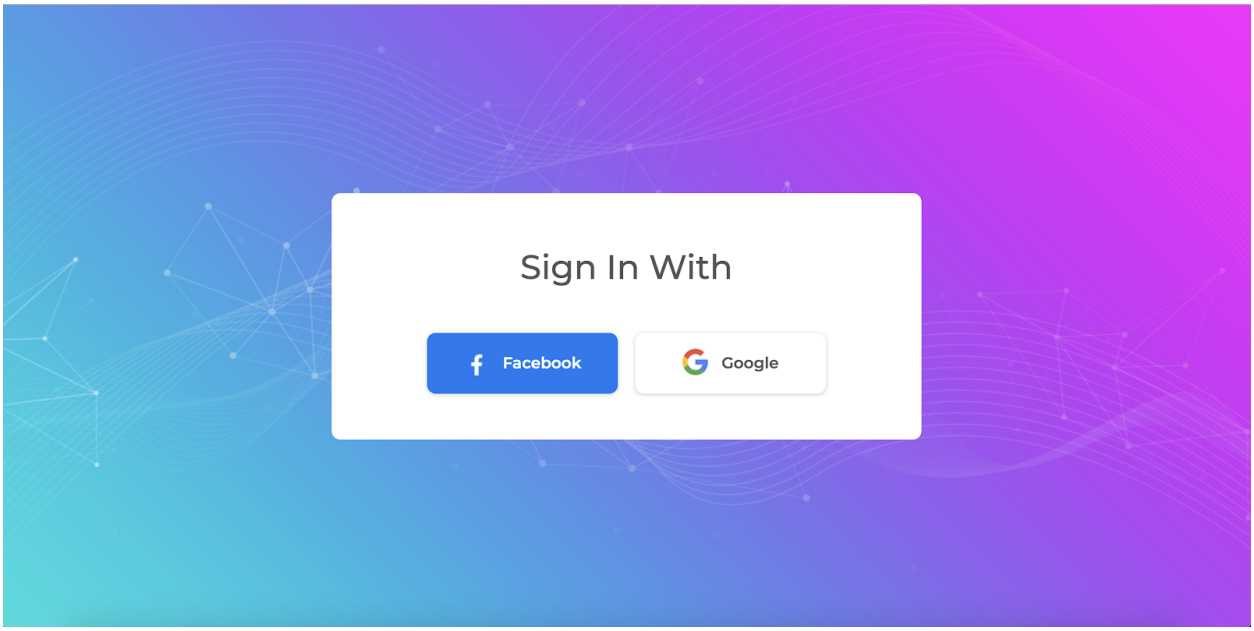


Figure 7: The landing page for Dance Studio [21].

Description: As seen in Figure 7, the landing page offers users the option to sign in with either Facebook or Google; clicking either button will redirect the user to a page created by the platform to authorize their account to be used with Dance Studio. The front-end interface is adapted from a template created by ColorLib [1]. The use of Facebook and Google accounts means that the application is accessible to a wide variety of users, and that secure password management is offloaded onto those platforms.

Implementation: Dance Studio uses Flask-Login to facilitate login and account management, so that users can log in to the application via Facebook and Google. Each user is uniquely identified by their account id, which is retrieved by accessing the Google OAuth API. Once authenticated,

the user remains logged in until the browser is closed or until they logout.

Figure 8: The main interface of Dance Studio [21].

Figure 8 shows the main interface of the application. The following sections provide detailed descriptions of the functionality of each component, as well as how they were implemented.

4.4.2 Stage and Dancers

Description: The stage and dancers as shown in Figure 9 is the main user interface for editing the dance, where the dancers are represented by rectangular prism meshes of different colors, and the stage is a section of a plane made to appear like wood. The dancers' names are shown above the meshes with a text sprite with a black background and white text for readability. In the bottom left corner of the GUI, the total number of keyframes is displayed, and in the other bottom corner, the current time within the dance is displayed. The user can move the dancers one at a time simply by clicking and dragging the mesh; when the user clicks on a dancer, its mesh will become a lighter color to clearly indicate to the user that the dancer is being selected. However, dancers always remain on the y-axis (which is perpendicular to the stage), so they cannot be pulled underneath or far above the stage. In addition, the user can manipulate the camera view by clicking and dragging

Figure 9: The main editing interface, showing a routine with 10 dancers [21].

anywhere else in the GUI, or zoom in and out by scrolling up or down. Furthermore, the user can move the camera position by using two fingers on a touchpad to drag the camera to the new desired position.

Implementation: The entire scene is visible through a camera that is positioned at a predetermined default viewing angle relative to the stage. Several Three.js point light objects illuminate the scene to allow the user to see the objects clearly and vividly. The stage is implemented in Three.js as a mesh created by plane geometry and the mesh phong material, which maps an image of a wooden floor onto the mesh. Dancers are similarly visually represented by meshes created from box buffer geometry and mesh lambert material with an image mapped onto it. The text sprites above the individual dancers display the dancers' names; this was created from adapting the tutorial to create text sprites in Three.js by Johannes Raida [22]. In essence, a canvas is created to be positioned above each dancer mesh with a black background and white text to show the dancer's name.

Dancers are represented as Javascript objects with the following attributes:

- **`keyframePositions`**: an array of keyframes representing the dancer's position, begin-

ning and ending times, and the corresponding transitional paths between keyframes. See Section 4.4.3 for more details.

- **`name`**: the dancer's name, which is set by the user or defaults to "Dancer1", "Dancer2", etc.
- **`mesh`**: a Three.js mesh created from BoxBufferGeometry and MeshLambertMaterial.

The ability to drag and move the dancers is implemented through the technique described in the Script Tutorial "Three.js - Drag and Drop Objects" [17]. Within the "`onDocumentMouseDown`" event listener, a Three.js raycaster object checks if the user's selection intersects with any of the objects in a pre-defined array of "selectable" objects, then updates the position of the selected object by repositioning the object to its intersection point with the y-axis plane.

4.4.3 Timeline

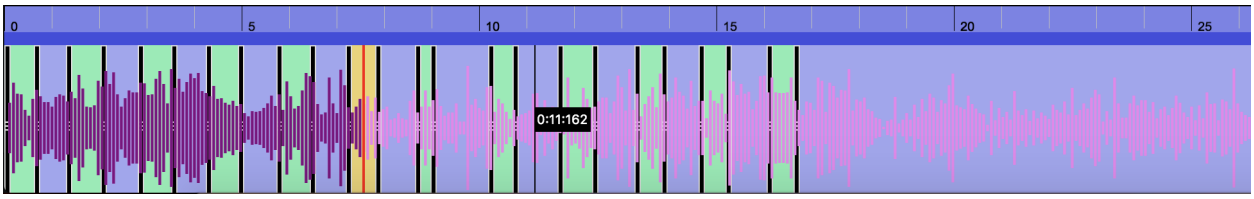


Figure 10: The timeline portion of the user interface [21].

Description: The timeline as shown in Figure 10 is the secondary user interface for editing the dance. The thin red bar indicates the current time within the dance, while the thin black bar with the time signature indicates the time hovered over by the user's cursor. If the user clicks, the red bar will update its position as the application updates the current time. The numeric scale at the top of the timeline delineates the time to the nearest thousandth of a second.

Audio is represented by the pink and purple sound waveforms that vary in height depending on the pitch and volume of the audio. As a result, the user can easily visually identify points in the song to place certain formations. The waveform representing the portion of the audio that has not been played yet is pink, while the remaining portion of the waveform is purple.

Keyframes, otherwise known as distinct formations, are represented visually by green and yellow rectangles that each have a beginning time, end time, and duration. As the user hovers over a keyframe, or if the current time is within a keyframe, it becomes yellow in color. The user can slide the black handles on the border of each keyframe to adjust its beginning or end times, or click and drag the keyframe from its colored middle section to move it in either direction along the timeline, even reordering the keyframes within the routine. Adjusting a keyframe preserves the position of the dancers, only changing the timing of the formation. However, the user cannot move the first keyframe from its position since there must always be a starting position to the dance. To create a new keyframe, the user clicks on the time in the dance where he wants the keyframe to start, then moves one of the dancers onstage to the next position. As a result, a keyframe is automatically created: if the new keyframe is the last one in the sequence, its duration is 2 seconds; otherwise, its duration is half the length of time between its beginning and the beginning of the next keyframe (this is to ensure that in cases where the keyframe is less than 2 seconds before the next keyframe, there is no overlap). If the user adjusts the positions of the dancers in an existing keyframe, it will update itself to have the dancers' new positions. The application automatically loads with one keyframe already created from 0 seconds to 2 seconds; the user cannot delete this keyframe. This is because the dancers have a default position upon loading, and every dance created has a minimum of one formation.

Implementation: The timeline was implemented using wavesurfer.js, a "customizable audio waveform visualization, built on top of Web Audio API and HTML5 Canvas" [15]. Wavesurfer.js was selected due to its easy usability, thorough documentation, and numerous plug-ins that meet the needs of Dance Studio's features. Firstly, wavesurfer.js is built on Web Audio API, so it will accommodate Firefox, Google Chrome, Safari, and Opera, which comprise all commonly used browsers. The core functionality of wavesurfer.js allows the user to upload a music file (either locally or via a URL) and will display a visualization of the sound waveforms, in addition to allowing the user to play or pause the music. Additionally, the wavesurfer.js Timeline plugin easily adds a notched panel with time increments to localize the audio file. Since wavesurfer.js

requires an audio file to show the timeline, if the user does not upload an audio file of their own, Dance Studio automatically loads in a 3 minute long silent audio file.

The keyframes are visually represented by the wavesurfer.js Regions plugin. According to their creator, "regions are visual overlays on [the] waveform that can be used to play and loop portions of audio. Regions can be dragged and resized" [12]. When the user goes to a point in time in the dance where there is no current keyframe, a new keyframe is created. Keyframes can easily be deleted with methods from the Regions plugin API, and event listeners detect when visual representations of keyframes are being dragged or resized to determine how to update the keyframe data structures. Each region has, among other attributes, a beginning, an end, and a color. When the user hovers over or is currently editing a keyframe, its color will change from green to yellow for clarity.

In determining how to represent keyframes in a more abstract way, I had the option to base keyframes off of the timeline, of the wavesurfer.js regions, or of the dancers individually. Ultimately, I decided to represent keyframes based on individual dancers because I was mindful of the issue of adding and removing dancers from the routine. However, in retrospect, a simpler solution would be to store each dancer's position within an object array in each keyframe. In other words, rather than making dancers hierarchically superior to keyframes, it is actually more intuitive to make keyframes the "parent" with dancers' positions as one key-value pair in the keyframe object.

Keyframes are represented as Javascript objects for each dancer object, which has an array of keyframes. Keyframes have the following attributes:

- **`beginning`**: indicates the absolute starting time of the keyframe in seconds.
- **`end`**: indicates the absolute ending time of the keyframe in seconds.
- **`position`**: indicates the dancer mesh's position onstage for the keyframe.
- **`curve`**: a `CatmullRomCurve3` object representing the curve corresponding to the dancer's path from the current keyframe to the next position. The final keyframe's curve is null.

- **`splineHelperObjects`**: draggable, physical meshes that are boxes representing the control points for the Catmull-Rom curve.

- **`positions`**: an array of positions that indicate where the spline helper objects are onstage.

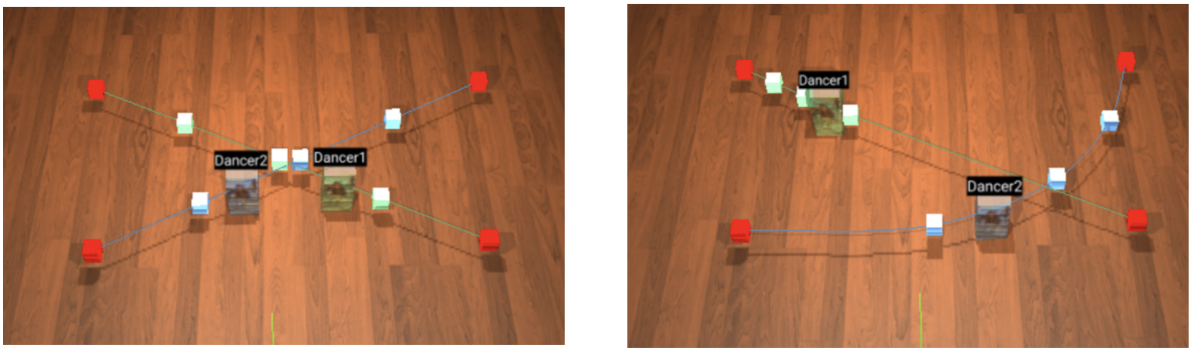
The array of keyframes for each dancer is always sorted in chronological order, so as soon as a keyframe is dragged or updated, it is repositioned within the array as well. If a wavesurfer.js region is updated, Dance Studio checks if the keyframe's new position overlaps with other keyframes, and if it does, the user is notified that overlapping keyframes are not allowed, and the visual representation of the keyframe will return to its previous state. One challenge in checking for overlapping keyframes is that the Regions API method does not return the region's original beginning and ending times, so it is difficult matching the updated region to the keyframe that it visually represents as it is stored for each dancer. To work around this issue, I assign each wavesurfer.js region a specific `id` that corresponds to its keyframe's beginning time. Then, when a region is updated, while its beginning and ending times may have changed, its `id` is still the beginning time of the keyframe, so I check the `id` with the beginning time to identify the match. Once a match is found, I verify that the new beginning and ending times do not conflict with any of the other existing keyframes.

4.4.4 Updating the dance in real time

One interesting design decision was to limit server-client communication as much as possible, since most rendering and computation can occur entirely on the client side. Since the user is constantly updating the positions of the dancers, and the Three.js animation loop must continually render animation frames, the real time updating occurs within the client. Time is computed with respect to the wavesurfer.js object. In an `update()` loop that is continuously called while rendering the dance, I use the `getCurrentTime()` method from the wavesurfer.js object to determine the time. Then, to properly update each dancers' position, I iterate through the array of dancers. For each dancer, I iterate through their keyframes to determine if the current time is within a keyframe or between two keyframes. While this loop is proportional to the number of dancers multiplied by

the number of keyframes, the computation is so efficient that there is no delay in runtime. In the first case where the time is within a keyframe, each dancer's mesh position is the same position as defined in the keyframe. In the case that the time is between two keyframes, k_1 and k_2, I compute the time difference ρ between the beginning of k_2 and the end of k_1, then subtract k_1's ending time from the current time to get a value δ. I finally determine the dancers' progress between the two keyframes by dividing ρ by δ, which results in a fraction. Between every two positions, there is a Catmull-Rom Curve that delineates the path the dancer takes during the transition time (see Section 4.4.5). For each dancer, I use the Three.js method `curve.getPoint(frac: Float, dancerPosition: Vector)` to compute the point that is a fraction of the way along the curve and assign the point value to the dancer position to update the mesh in real time. Lastly, if the time is past the last keyframe's ending time, the dancers simply remain in the last keyframe's position.

4.4.5 Editing Paths



(a) Default straight paths, resulting in a collision [21].

(b) Paths after user adjusts them to avoid the collision [21].

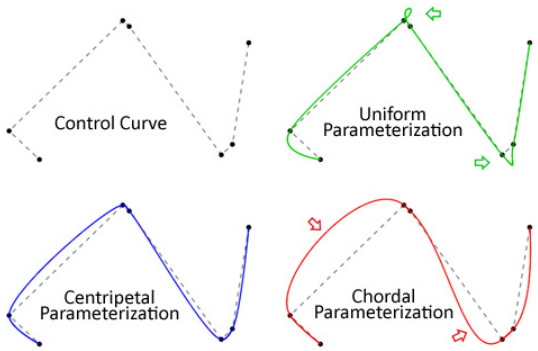
Figure 11: User path editing. In (a), the two dancers collide, but in (b), the custom paths prevent them from being in the same place at the same time.

Description: By default, Dance Studio linearly interpolates between dancers' positions to define their paths between keyframes. However, if there are collisions, or if the user desires to achieve a particular movement style, the user can create custom non-linear paths for their dancers through the Edit Paths feature. The user must check the "Edit paths" checkbox on the left sidebar to reveal

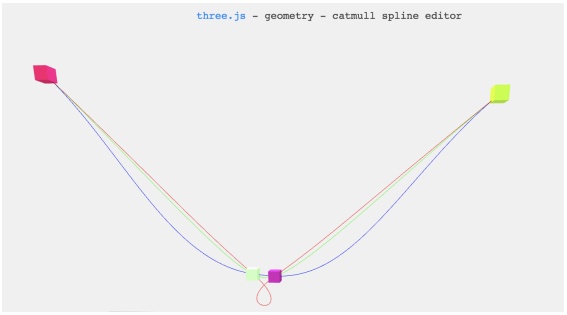
this feature, which enables the user to see visualizations of the paths dancers take during their transitions between keyframes (see Figure 11). The paths are represented by a thin curveable line in the dancer's color and five block-shaped boxes, three of which are in the corresponding dancer's color, and two on the end in a red hue. This color difference indicates to the user that they are only able to edit the three middle blocks, because if they wanted to edit the endpoints, they can do so in the keyframe instead. Furthermore, while the user is editing a path, the user can see a more transparent version of the dancer's mesh along the path as a guide, although the mesh is not draggable. The remaining three boxes share the same color as the corresponding dancer to avoid confusion with other dancers in the case that there are many paths onstage. To edit a dancer's path, the user can click and drag the three block meshes in the middle one at a time to move them to their new desired position. The curve automatically updates in response to the user's movements. Furthermore, having three blocks allows the user to manipulate the dancers' speed; there is an equal amount of time between each block, so dancers move relatively faster when blocks are farther apart, and relatively slower when blocks are close together.

Implementation: The paths were implemented using centripetal Catmull-Rom splines, which are "piecewise polynomials that are used to pass through several control points" [3]. These splines were selected as opposed to other implementations because they are already built into the Three.js library, the control points would allow the user to easily edit the path (rather than an alternate implementation of having the user draw the path with a mouse or trackpad), and the control points allow users to manipulate the dancers' speeds. Additionally, Catmull-Rom splines interpolate between control points, meaning they pass through every single point, as opposed to approximating splines that are estimates between the points. For Dance Studio, the former is more likely to be intuitive for beginners since the curves closely follow the control points in a predictable manner. For the Three.js `CatmullRomCurve3` object, there are three distinct types of curves which all differentiate themselves in how the curve is computed. I selected centripetal Catmull-Rom splines in favor of chordal or uniform splines because the curve more closely follows the control points and does not result in excessive looping (see Figure 12). Each representation of a curve is composed of

the `CatmullRomCurve3` object, an array of five splineHelperObjects to display the draggable blocks along the path, and an array of five positions that correspond to each of the blocks.



(a) Different types of Catmull-Rom splines [9].



(b) Distinct Catmull-Rom splines represented in Three.js, where the red curve is uniform, the green curve is centripetal, and the blue curve is chordal [2].

Figure 12: Catmull-Rom splines with the same control points in 2D and 3D.

When a keyframe is added, a straight Catmull-Rom curve is created using simple linear interpolation between the two positions. As soon as a dancer's position is updated, the curve is deleted and recreated with the new positions. Additionally, if keyframes are rearranged in order, the curves for the affected keyframes are removed and recreated. However, if a keyframe is dragged but the keyframe order is preserved, the paths remain unchanged. By default, the curves are all created with meshes set to have `visibility = false`, which is then changed if the user selects the "Edit Paths" checkbox. To determine when to reveal a path, in the continuous update() loop, paths are set to visible only if the current time corresponds to the time appropriate for the path to be shown, and if the "Edit Paths" feature is on. Conversely, paths are set to being invisible at all other times.

To allow the user to edit the paths by dragging the boxes, the same raycaster that was used to check for dancer meshes also checks for curve blocks, since the user can only select one item at a time (either a dancer mesh or a block along a path). A separate array keeps track of all selectable blocks by constantly adding and removing relevant blocks depending on the current time of the dance and the status of the "Edit Paths" checkbox; this is to avoid having the user unintentionally drag invisible blocks out of position.

4.4.6 Sidebar GUI

Description: This sidebar not only reveals important information to users about the current dance they are editing, but also allows users to edit the current dance and to access other dances. First, the top light blue bar contains the user's profile picture (taken from Facebook or Google), a greeting with the user's name, and a power button that logs the user out. Moving downward, the dance name is displayed, with a help button on its right that opens a user tutorial modal, followed by a button that opens the "Edit Dance" modal (see 4.4.7) and a button that opens the "Edit My Dances" modal (see 4.4.8). Next, a small canvas displays an aerial view of the dance, which updates in real time with the user's edits. Having both the overhead view and the main perspective view always visible helps users ensure their formations are lined up properly without forcing them to constantly realign their main camera to the aerial view. Below the canvas, a "Save Dance" button allows users to manually save their dance, and the current "saved" status of the dance is displayed on the right. If the dance is saved in the database, the message "Saved!" appears in white. If the dance is currently in the process of saving, the message "Saving your changes. . . " appears in yellow. Lastly, if there was an error in saving the dance, the message turns red and becomes "Error: Unable to save changes. Try again.", to prompt the user to click the button until the dance is saved

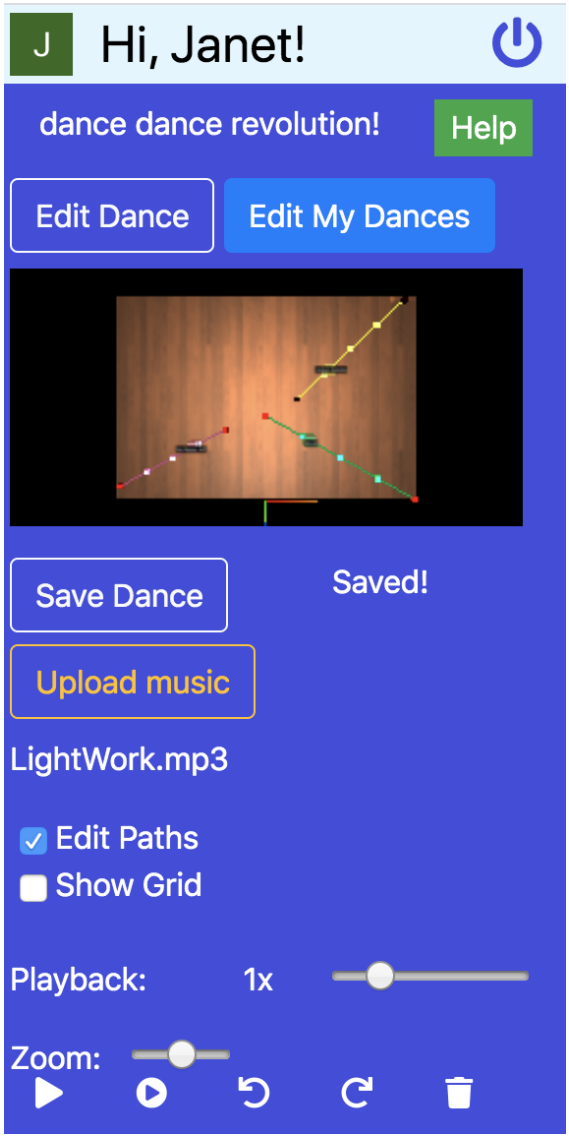


Figure 13: The sidebar user interface [21].

properly. These messages update in real time as the dance is continuously autosaved. Next, the "Upload Music" button allows users to add an audio file from their computer's local file system to their dance, displaying the name of the file below in white. Below this, two checkboxes offer the user options to toggle the "Edit Paths" and "Show Grid" features. A playback slider gives users the option to manipulate the dance's playback speed from 0.05× to 5× the normal speed, in increments of 0.05. Next, users can zoom in and out of the timeline by moving the corresponding slider. Finally, directly above the timeline, five icons provide control over the timeline, with buttons to play/pause the dance, play the dance from the beginning, undo the previous change, redo the previous undo, and delete the current keyframe. When the user hovers above each of these icons, a tooltip appears above them to explain the functionality. To play and pause the dance, the user can also press the space bar.

Implementation: The sidebar is implemented using Bootstrap 4 buttons, sliders, and containers, and the aerial view is created with a renderer in Three.js. The canvas renders the same scene as in the main view from the perspective of a different camera, which cannot be adjusted by the user.

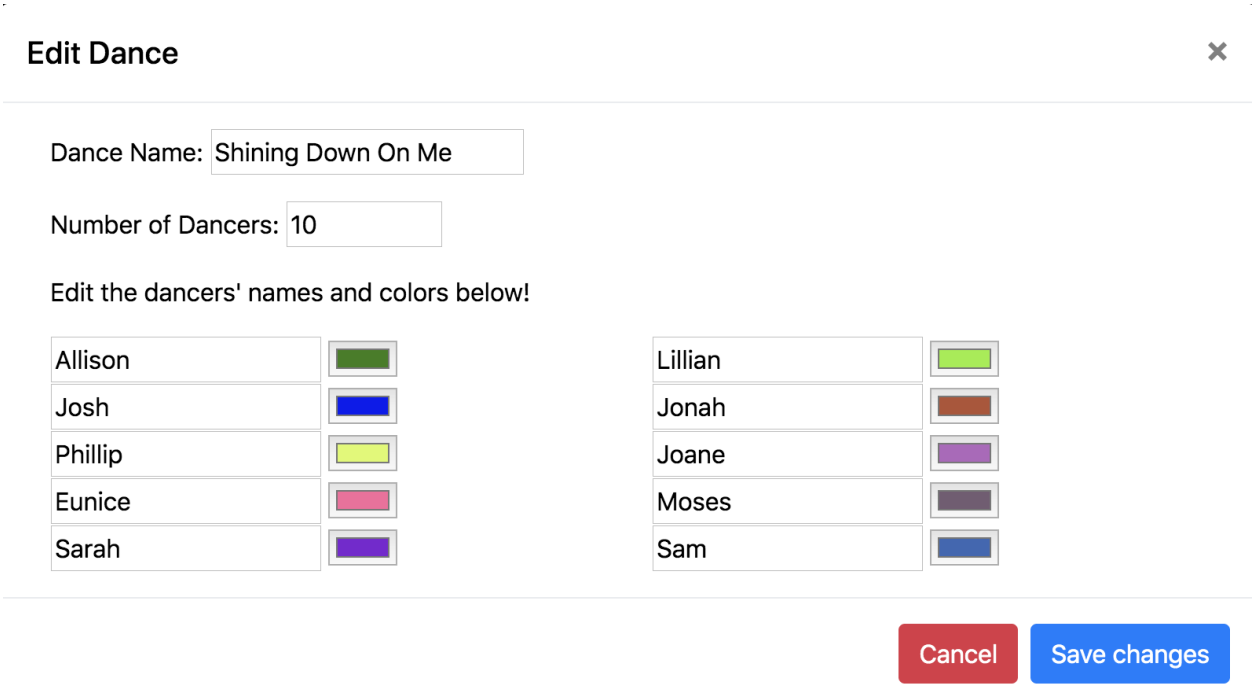


Figure 14: The modal users can use to edit the current dance [21].

4.4.7 Edit Dance Modal

Description: As shown in Figure 14, this modal allows users to edit the current dance's name and dancers. In addition to adding new dancers, users can also modify existing dancers' colors and names.

Implementation: This modal is implemented using the Bootstrap 4 modal component, in addition to an HTML 5 form that presents the dance's information and updates the dance according to the user's changes. This modal is identical to what the user sees when creating a new dance for the first time.

4.4.8 Welcome/Edit My Dances Modal

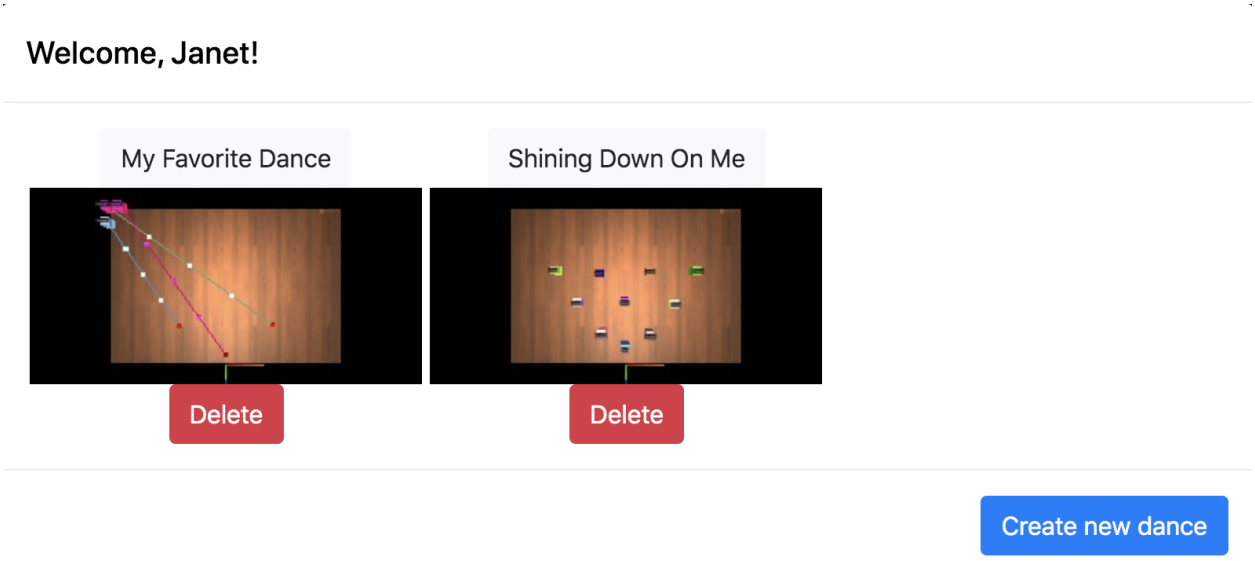


Figure 15: Welcome Modal / Edit My Dances Modal for a user who has created two dances [21].

Description: As shown in Figures 15 and 16, these modals are accessible by automatically appearing when the user logs into the application and if the user clicks the "Edit My Dances" button from the sidebar. If the user has already created dances in Dance Studio, this modal displays each dance's name and an aerial view of the stage from the most recently saved version of the routine. The user can edit existing dances by clicking on the name or the image, or delete a dance by pressing the red button below its image. Additionally, the user can create a new dance by selecting

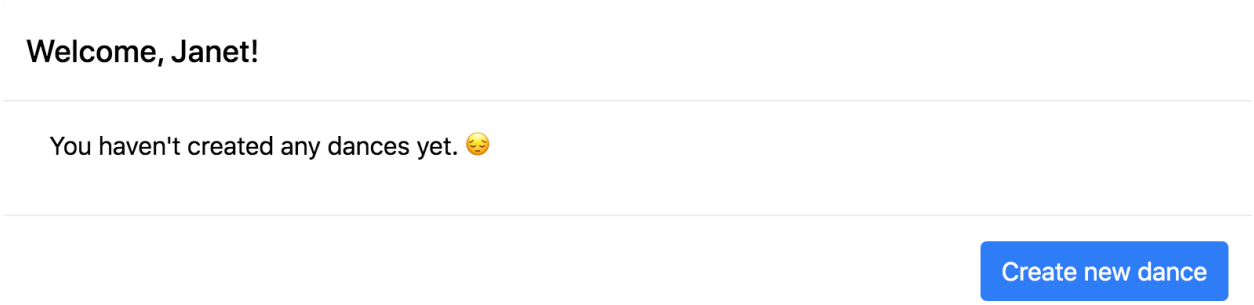


Figure 16: Welcome Modal for new users [21].

the blue button in the bottom right corner.

Implementation: These modals are implemented using Bootstrap 4's modal component. Dance Studio makes a GET call to the database to retrieve the user's dances, and if there are none, then the Welcome Modal appears. On the other hand, if the user already has dances, each dance's information is tracked so that if the user chooses a dance, the stage is loaded with the proper number of dancers, keyframes, paths, and audio.

5 Evaluation

To evaluate the performance of Dance Studio, I revisit its original goal: Does it allow users to more easily plan their formations for their dances? In this section, I detail how I first assess Dance Studio on a technical basis, using Nielsen's Heuristics, and next, how I conducted user evaluations to understand how well Dance Studio is received by real people.

5.1 Technical Analysis

The technical expectation of Dance Studio is for the system to be bug-free and to effectively communicate to users how to use the application, offering an intuitive and smooth user experience. To test my system throughout the building process, I often went through cognitive walkthroughs of Dance Studio, iterating through as many possible use scenarios as I could imagine. Whenever I added a new feature, I repeated the same walkthrough to ensure the update did not introduce any new bugs.

Furthermore, before I conducted user studies, I used Jakob Nielsen's heuristic evaluation as a guideline to estimate how well my application would perform with real users. In Nielsen's approach, ten heuristics for user interface design encapsulate qualities of a system with quality human computer interaction [20]. Table 2 details examples of how, based on my expert evaluation, Dance Studio meets the needs of its users through each heuristic.

However, when I conducted my user studies, I discovered a gap between my analysis and actual user behavior. Several features that I believed to be self-explanatory were not actually intuitive for users. As a result, I iterated on Dance Studio and continued to refine my understanding of user behavior.

5.2 User Evaluation

To evaluate how well users respond to Dance Studio, rather than collecting succinct feedback from hundreds of users, I conducted long-form, qualitative evaluations with ten people. Given the

Stage of Heuristic Evaluation	Implementation in Dance Studio
Visibility of system status	The user can always see the status of their dance, indicating whether or not it is saved to the database.
Match between system and real world	Vocabulary that is familiar to the user is used in every aspect of the application, with plain terms allowing them to save their work, play the routine, delete formations, etc.
User control and freedom	System allows users to "undo" and "redo" their most recent changes, to delete keyframes, and to alter the speed of the music playback.
Consistency and standards	Consistent vocabulary is used to describe the same ideas, and a standard color convention of green indicating success and red indicating failure is used for the dance's saved status.
Error Prevention	Tooltips indicate each button's functionality to prevent button misuse, and autosaving ensures the user has the most recent version of the dance available.
Recognition rather than recall	Users can view their keyframes by playing the dance or going immediately to the time they are interested in. They do not need to memorize any information themselves, as everything they need is displayed on the screen.
Flexibility/efficiency of use	Creating a basic dance (as in the User Task List, see the Appendix, Part B) takes less than 30 seconds. Both new and experienced users can efficiently create dances according to their needs; the "Edit Paths" option allows more experienced users to create more refined routines.
Aesthetic and minimal design	The site is minimalistic and only contains necessary features; the bright color scheme is pleasing to the eye.
Helps users recognize, diagnose, and recover from errors	When the dance cannot be saved, the user sees an error message, which prompts them to press the "Save" button to try again.
Help and documentation	A help button is always available for the user to see how to use the system, and each icon has a tooltip to explain its function (eg: "Play from beginning", "Undo", "Redo", etc.).

Table 2: Nielsen's Heuristics as implemented in Dance Studio.

limitations of the scope of this project, and the importance for Dance Studio to be user friendly, I greatly valued having more detailed qualitative feedback and the advantage of being able to watch user behavior as it happens live. I selected my users by reaching out to a segment of the target user demographic, my peers in the dance community at Princeton. Despite the small sample size, people with a range of dance experience were represented in the test group, which included some people completely unfamiliar to dance and others who have been choreographing for years. Additionally, the users were equally split in using Mac and PC machines.

The evaluation itself consisted of completing a user task list to use Dance Studio to create three dances of increasing complexity, and then to optionally make an original dance. The task list prompts the user to follow a list of instructions to produce the dances, however, specific application features are not specified, so the user must independently translate the broader instruction to Dance Studio. For example, one instruction is to move the dancers to the front of the stage; the user must determine how to move the dancers by clicking and dragging the meshes across the stage. Observing users going through the task list revealed where users were confused about how to implement a certain action due to poorly defined features. Additionally, by observing user behavior in real time, I could see how, when confronted with a task they did not know how to complete, people sometimes responded by misusing other features or performing other actions. The full task list is available in the Appendix, Part B.

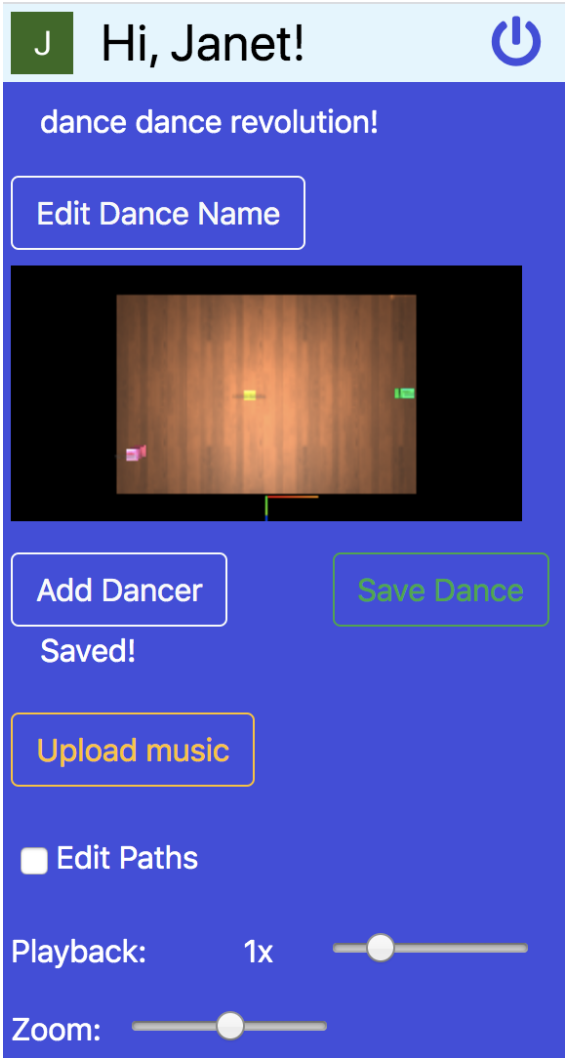
My pilot study was conducted with Professor Dondero individually. At the time, the application user interface was not very user friendly; the user was unable to complete several tasks without my verbal aid due to an unclear user interface. For example, the current time was not clearly identifiable, since its marker was a black line on a timeline with other black lines, so I updated the marker to be red and thicker in width. In addition, several bugs were identified in this round of evaluation, at which point, I ensured the system was bug-free before continuing with further evaluation. Furthermore, I updated the task list to clarify confusing or overly ambiguous instructions.

After I iterated on the feedback suggested in the first round of evaluation, ten more users evaluated the system using the updated task list. On average, when asked how likely they were

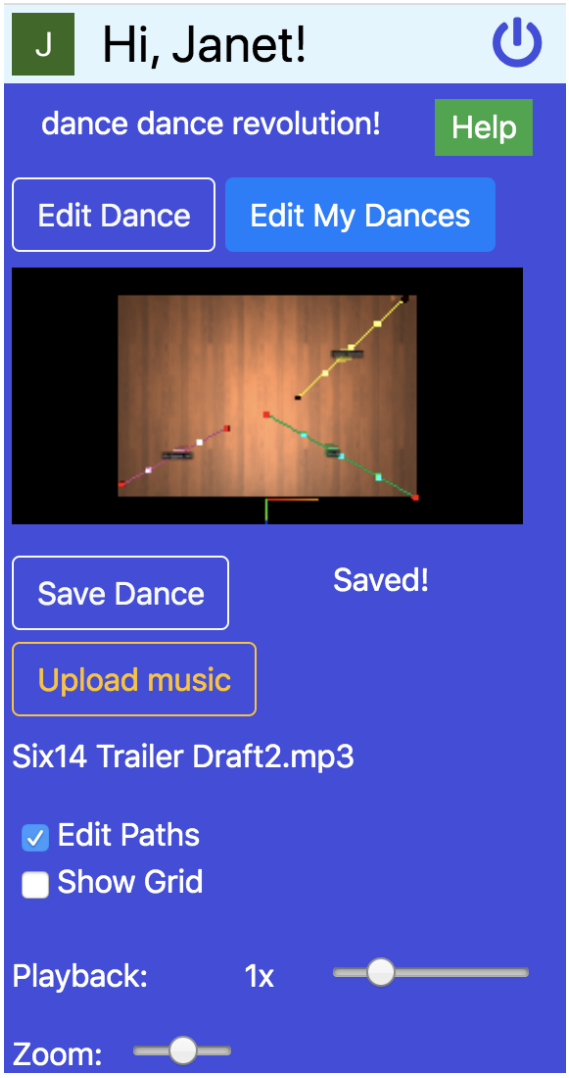
to use Dance Studio to choreograph their next dance, users gave a rating of 8/10. Their favorite feature was the core functionality of the application; many users said it solved many of the issues they faced when attempting to use pen and paper to keep track of formations. They enjoyed the ease of moving dancers on stage, the generally intuitive nature of the interface, and the ability to upload music and visualize it on the timeline. Another highly praised feature was the edit paths functionality; all users agreed that the option to create custom paths took the application to the next level, and it was "very satisfying to see the changes happen so quickly in real time" (Yoon, 2020). Most users opted to create their own choreography because they naturally desired to experiment more with the application and with possible formations, indicating that it encourages user creativity. Overall, there was no noticeable difference in user behavior between experienced dancers and novices.

As mentioned above, while generally intuitive, the system at the time of user testing did not completely fulfill Nielsen's Heuristics with actual users. The main constructive feedback relates to buttons on the sidebar GUI not being clear. For example, users were confused about how to make a new dance; in the version they evaluated, a user must click on their own name in the top tab to access their dances and to create new routines. However, while this was intuitive to me, for most users, this was not self-evident. As a result, I added a blue "Edit My Dances" button above the aerial rendering of the screen. Additionally, the button to allow users to save their dance was green, making it blend in too much with the background; to resolve this issue, I changed the button to be white to stand out more clearly.

Additionally, in the main graphics interface, dancer names were hard to read because at the time, the text sprites above each dancer had no black background, which I later added due to this feedback. While dragging dancers was straightforward, some users also found it difficult to physically drag the "Edit paths" blocks due to a lack of responsiveness of the system. Furthermore, several users reported they wished the stage had a grid to allow them to align dancers more accurately; I added this grid in the next version of the app. A significant minority of Mac users faced issues accidentally prompting the browser to go back to the previous page when they were scrolling



(a) Sidebar before user study [21].



(b) Sidebar after changes were made based on user feedback [21].

Figure 17: A before and after of the sidebar user interface based on implementing user feedback. The key differences are the new ”Help” button, the new “Edit My Dances” button, the recoloring and repositioning of the “Save Dance” button, the removal of the “Add Dancer” button in favor of allowing users to do so in the “Edit Dance” modal. The newer version also enables users to toggle displaying a grid onstage.

through the timeline, before they had an opportunity to save their dance. Additionally, with PC users and people with shorter screens, there were occasionally issues with the content of the sidebar overflowing into the timeline; I resolved this issue by making the sidebar scrollable. While the system allows users to use the spacebar to play and pause the dance, several users desired even more keyboard shortcuts to speed up the workflow.

6 Limitations and Future Work

6.1 Known Limitations

Occasionally, the database is inconsistent between the local environment and Heroku's PostgreSQL. As a result, while the database is successful in reliably saving work on the local version of the application, it occasionally times out and is unable to make GET or POST requests to the Heroku PostgreSQL database. Originally, the system autosaves every time a dance is changed (updating a keyframe, moving a dancer, editing a path, uploading music, changing the name, etc.). However, since autosave was only successful approximately 80% of the time, I modified Dance Studio to have a manual save button that prompts the user to click save as often as possible, and then to try again if an error message is displayed. When testing this method with users, some people forgot to save their work frequently, and all users generally preferred having an autosave as well. To compromise between these two methods and ensure that users have access to the most recent version of their work, dances are autosaved, and there is also a save button to allow the user to manually store their work if the autosave fails. This way, if the user is not able to save their work frequently, a relatively recent version of their dance is still available due to autosave.

Additionally, sometimes the initial modal does not load due to a 401 server authorization error, likely due to an inconsistency with Flask-Login and database authentication. As a result, the user must click on the "Edit My Dances" button to access the modal.

The functionality of the "Undo/Redo" feature is inconsistent; it can be seemingly unresponsive, then overly aggressive in undoing changes. While dancers may be added or modified, the ability to delete dancers has yet to be implemented. When a new dancer is added, there occasionally is an error in rendering the paths and position for the new dancer. While the limitations mentioned do not inhibit the core functionality of the application, they are next highest in priority to resolve to improve Dance Studio's usability.

6.2 Future Work

In the future, I hope to focus on shareability and ease of use, since the main suggestions from users revolved around these two principles. To upgrade Dance Studio from a single-user application to become more collaborative, I hope to implement features such as the ability to export to a video, a share feature that allows different users to view and edit the same dance within Dance Studio, and a printer-friendly exporting option that displays the dance as images of each formation with timestamps. Furthermore, to increase accessibility, Dance Studio can be further adapted for mobile screens and other non-desktop devices. To improve upon the application's ease of use, potential future features include the abilities to move multiple dancers at a time by highlighting a section of the stage, to copy and paste formations, to access a library of existing formations for the given number of dancers, and to create copies of dances to encourage further experimentation.

In addition to adding the proposed features to increase accessibility and the application's usability, in the long term, I hope to add on to the core functionality of the application to reflect a fuller dance performance experience. One option is to incorporate custom lighting options, which is achievable using Three.js, on a separate timeline to allow users to experiment with lighting their performance. This would also require the dancer meshes to appear more human-like to allow users to better assess how their lighting looks on real people. Additionally, a further advancement is to enable these humanoid meshes to have motion capabilities to simulate entire dance sequences virtually, using previously defined dance moves or by allowing users to film themselves dancing with technologies such as OpenPose, a system which uses computer vision to detect human body keypoints in single images [18].

6.3 Future Applications

Dance Studio also has applications beyond just the dance community. The application's core functionality can be used to map out blocking for theatrical plays, musical theater productions, a capella performances, cheerleading competitions, and ice skating shows. This blocking tends to be more simplistic than in dance choreography, so little modification to the application is needed.

The movement of individuals on a surface is also necessarily mapped in sports plays; instead of dancers, the main meshes can represent individual players, while the "Edit Paths" functionality in particular can allow coaches to experiment with and visualize movement in football, basketball, soccer, and other sports. Players on opposing teams can be differentiated by their colors, and each "dance" can represent a different play. One modification is to include a ball and potentially a basket, goal, or end zone.

7 Conclusion

In sum, Dance Studio achieves its goal of simplifying the choreography process by allowing users to easily visualize and plan their dance formations. Based on feedback from users, Dance Studio offers aspiring choreographers a simple solution to clearly block their dances and anticipate and avoid collisions between dancers. While the user experience can still be improved, users are pleased with the application and are extremely likely to use it in the future. Dance Studio empowers all users, regardless of their experience, to succeed in transforming their creative vision into a reality.

References

[1] [Online]. Available: https://colorlib.com/etc/lf/Login_v5/index.html

[2] [Online]. Available: https://threejs.org/examples/webgl_geometry_spline_editor.html

[3] "Computer graphics: 15-462." [Online]. Available: http://www.cs.cmu.edu/~jkh/462_s07/

[4] "Creating a new formation?" [Online]. Available: http://danceapp.us/

[5] "How to use imovie to edit videos and make a movie." [Online]. Available: https://filmora.wondershare.com/imovie/how-to-use-imovie.html

[6] "imovie." [Online]. Available: https://www.apple.com/imovie/

[7] "Interface overview." [Online]. Available: stagekeep-docs.helpscoutdocs.com/article/8-interface-overview

[8] "Login." [Online]. Available: https://flask-login.readthedocs.io/en/latest/

[9] "Parameterization of catmull-rom curves - cem yuksel." [Online]. Available: http://www.cemyuksel.com/research/catmullrom_param/

[10] "Postgres - sql database service." [Online]. Available: https://www.heroku.com/postgres

[11] "The python sql toolkit and object relational mapper." [Online]. Available: https://www.sqlalchemy.org/

[12] "Regions plugin." [Online]. Available: https://wavesurfer-js.org/plugins/regions.html

[13] "Sahil diwan - local diner." [Online]. Available: http://blog.sahildiwan.com/posts/flask-and-postgresql-app-deployed-on-heroku/

[14] "three.jsr116." [Online]. Available: https://threejs.org/

[15] "wavesurfer.js." [Online]. Available: https://wavesurfer-js.org/

[16] "Webgl - opengl es for the web," Jul 2011. [Online]. Available: https://www.khronos.org/webgl/

[17] Andrey, "Three.js – drag and drop objects," Jun 2015. [Online]. Available: https://www.script-tutorials.com/webgl-with-three-js-lesson-10/

[18] CMU-Perceptual-Computing-Lab, "Cmu-perceptual-computing-lab/openpose," Apr 2020. [Online]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose

[19] B. Foundation, "Home of the blender project - free and open 3d creation software." [Online]. Available: https://www.blender.org/

[20] W. L. in Research-Based User Experience, "10 heuristics for user interface design: Article by jakob nielsen." [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/

[21] J. Lee, "Dance studio." [Online]. Available: http://dancetigers.herokuapp.com/

[22] J. Raida, "Software." [Online]. Available: http://www.johannes-raida.de/tutorials/three.js/tutorial13/tutorial13.htm

[23] StageKeep, "The formation app for every performer." [Online]. Available: https://stagekeep.com/

A Database Schema

A.1 User Table

Key	Description
`id`	The primary key for this table. The id is the user's Facebook or Google account unique user id, depending on how they logged into the application.
`name`	The user's first and last name.
`email`	The user's email address.
`profile_pic`	The url to the user's profile picture.

A.2 Dance Table

Key	Description
`id`	The unique dance id, the primary key for this table.
`user_id`	The user's id (the "id" field in the User table) for the corresponding user to whom this dance belongs.
`user_email`	The user's email address.
`dance_name`	The name of the dance, as specified by the user.
`dancers`	The dancer objects for the dance, stored as a string. See the dancer class below.
`keyframes`	The array of keyframe times for the dance, stored as a string.
`number_of_keyframes`	The number of keyframes, or formations, in the dance.
`audioFileName`	The local file name for the dance's audio.
`audioURL`	The URL for the audio file for the dance.

The **`dancer`** object is primarily dealt with on the client side via Javascript in the file Dancer.js. The object has the following attributes:

Key	Description
`name`	The name of the dancer, as specified by the user.
`mesh`	The mesh object that is displayed onstage in the Three.js renderer.
`keyframePositions`	An array of objects that specify the dancer's position at each given keyframe time.

The **`keyframePositions`** object is comprised of the following keys:

Key	Description
`beginning`	The absolute starting time of the keyframe in seconds.
`ending`	The absolute ending time of the keyframe in seconds.
`position`	The dancer mesh's position onstage for the keyframe.
`curve`	A `CatmullRomCurve3` object representing the curve corresponding to the dancer's path from the current keyframe to the next position. The final keyframe's curve is null.
`splineHelperObjects`	An array of 5 draggable, physical meshes that are boxes representing the control points for the Catmull-Rom curve.
`positions`	An array of 5 positions that indicate where each of the spline helper objects are onstage.

B User Task List

Welcome! Thank you for helping me in my senior thesis. Please follow the instructions to the best of your ability. The application is hosted online at `https://dancetigers.herokuapp.com/`. Once you get to the landing page, choose to sign in with Google.

B.1 Basic Use

To introduce you to the basic features of the app, we will make a very simple dance.

1. Make a dance called "Dance 1" with two dancers in it. Name the dancers "Blue" and "Red" and color them according to their names.

2. Make the two dancers stand in the back right corner (from the audience's perspective) for the first position.

3. Create a new position starting at time 15 seconds, where both dancers are standing next to each other in the front.

4. Play the dance from the beginning to watch what you just created. Once you have seen both positions, stop the playback. To increase the speed of playback if it is too slow, you can adjust the playback speed up to 5x if you like.

5. Extend the initial position so the dancers are stationary from 0 to 10 seconds.

6. Play the dance again.

7. Save your changes.

B.2 Intermediate Use

Now we will make a more complicated dance.

1. Make a new dance called “Dance 2” with 3 people in it (names and colors can be whatever you like).

2. Once the dance has loaded, rename the dance “Second Dance”.

3. Create three positions - the first where the dancers are in a horizontal line in the back of the stage (lasting from 00:00 to 00:03), the second where the dancers are in a horizontal line in the front of the stage (lasting from 00:06 to 00:10), and the third where the dancers are in a vertical line in the middle of the stage (lasting from 00:14 to 00:16). Play the dance to make sure it looks right.

4. Hit save until the changes are saved. Refresh the page and open up “Dance 2” again to make sure your work was saved.

5. Delete the second keyframe (originally from 00:06 to 00:10). Play the updated dance.

6. Add a new keyframe in its place where the dancers are all spread out over the stage - two in the front corners close to the audience, and one in the back of the stage in the center. Make this keyframe last from 00:06 to 00:10. Play the dance.

7. Move the second keyframe (originally from 00:06 to 00:10) to the new time 00:20. Play the dance to see the changes were made.

8. Save your changes.

B.3 Advanced Use

This list of tasks will introduce you to the more complex features of Dance Studio.

1. Make a new dance called “Dance 3” with 3 dancers.

2. Upload an .mp3 file from your computer that is at least 30 seconds long.

3. Create three positions - the first where the dancers are in a horizontal line in the back of the stage (lasting from 00:00 to 00:03), the second where the dancers are in a horizontal line in the front of the stage (lasting from 00:06 to 00:10), and the third where the dancers are in a horizontal line in the middle of the stage, where the order of the dancers from left to right is reversed (lasting from 00:14 to 00:16). Play the dance to make sure it looks right.

4. Go to a time in between the first two positions, and check the "Edit Paths" box to edit the dancer's paths to be more rounded.

5. Edit the paths between the second and third positions to avoid collisions between dancers.

6. Play back the dance to check your moves.

7. Move the second keyframe (originally from 00:06 to 00:10) to the new time 00:20. Play the dance to see the changes were made.

8. Save your changes.

B.4 Advanced Use (Optional)

Add one of your own choreographed dances to Dance Studio, or experiment with creating a new dance entirely.