CSC 586A Final Project Report

PartyVote: Democratic Music Selection in Co-Present Social Environments

Group Members
Nadia Rashid (nadiar@uvic.ca)
David Sprague (dsprague@csc.uvic.ca)
Fuqu Wu (fuquwu@csc.uvic.ca)
Abstract
In many informal social settings, disagreements about music choice can lead to conflict. Alternatively, some party attendees monopolize the music selection, subjecting everyone to their musical preferences. We present PartyVote, a democratic music voting system that provides both a traditional text based music selection system and a music library visualization to help users collaboratively choose what will be played during a social gathering (party). We describe the features and implementation of PartyVote and identify how our design addresses our four main design goals: system transparency, promoting awareness about decisions made by other users, interface intuitiveness, and system enjoyment. Perceived system strengths and weaknesses are discussed and finally we address future system developments planned.

1 Introduction
Conflicts in informal social environments such as house parties often arise from discrepancies in individual preferences. The party host or a self-designated party DJs can control the music being played and limit the control to a select few. Alternately, disagreements about what constitutes good or quality music can arise when multiple people attempt to control the music. Some hosts may attempt to reduce conflict by playing music that everyone already knows, frustrating more “adventurous” party goers. Other participants find familiar music enjoyable and relaxing and will get apprehensive when unfamiliar music is played.

Members of tightly knit social groups (parties) tend to behave by implicit or explicit rules but individual tastes vary. A visualization system for informal setting would ideally support these unwritten social rules while attempting to mediate conflict and reduce social tension. The PartyVote system described in this paper, attempts to leverage natural social interactions for users to regulate their behaviours, by providing a computer mediated method for democratically voting for the music to be played in a social environment. The primary visualization goals for the PartyVote system are: system transparency, social awareness, intuitiveness, and enjoyment. These goals are described in more detail in the System Design section.

The PartyVote system allows each participant in a social gathering to choose a song, album or artist from a local music collection. Each selected song is guaranteed to be played in the first four hours period of a six hour period. At least one song from a selected album or selected artist is guaranteed to be played during this period. Selected songs will also weight similar songs, improving their chances of being played. Finally, selected and weighted songs will define the boundaries of a region in the collection’s music information space. Songs outside of this region will not be played during the social gathering. The likelihood of each song in the region being played is relative to a weighting. In this way, each participant gets to hear what he/she wants and all other songs are optimally chosen to appease the greatest number of people possible.
2 Related work

There exist a number of systems and tools that share functional similarities with PartyVote. These functionalities include multiparty voting in an informal social gathering, generating playlists of similar songs based on user selections and visualizing songs in 2 dimensional space using similarities between songs as the clustering criteria [4].

Jukola [3] is a democratic mp3 jukebox designed for use in a public place. Its visualisation components assist users in nominating songs (with no guarantee of playing), voting for songs, and monitoring how many votes each song has received. Jukola is primarily text based and users need extensive explanations before they began using the system. Jukola also requires constant user feedback as every song played is voted for. PartyVote requires users to vote once at the beginning of the event, and guarantees that a voted song will be played.

MUSICtable (see Figure1) [5] is a collaborative music library visualization system, designed for private social gatherings. Its primary aim is to enable users to discover new music. Songs in the system’s database are visually represented using a static geographic map metaphor. Songs are clustered by region and color in specific locations according to their similarities based on sound and genre. Users cannot directly select songs or place votes for songs. Instead the current focus “drifts” across the music landscape based on the map’s “wind” direction. In turn, the wind direction and speed is controlled by buttons on the edge of the table. MUSICtable does not offer guarantees on which songs are played and doesn’t address the “self-designated DJ” problem. MUSICtable also does not generate a playlist so user interactions are regularly required. Although Staveness et al. [5] suggest tabletop displays because they believe the computer can get in the way of collaborative interactions, we chose to use a PC because table tops are not very accessible to the general public. We hope to make PartyVote as practical as possible.

Figure 1: An image of the MUSICtable collaborative music visualization system.

Musicland [2] is another music library visualization system designed to explore a music collection and it enables a single user to query the system for songs of a particular mood.
or style. The query results are displayed as clusters based on music similarity using a rectangular 3-term Venn diagram analogy. Different regions of the Venn diagram are represented by different colors. Both Musicland and PartyVote allow users to find similar songs based on a visual search.

Artist Map [6] is a single user system which displays songs in a music database in 2-dimensional space. Songs are clustered by the similarities they share in their metadata. Although this system has a playlist generation algorithm, it does not enable multi-user participation and voting. Its visualisation does not have any noticeable song selection indicators either.

PartyVote represents multidimensional song data using a 2D graphic interface. We decided to use multidimensional scaling to achieve this and so we used the program MDSteer [8]. MDSteer handles distance measures and incrementally adds points to the 2D projection, making the system functional for even extremely large data sets.

3 System Design

3.1 System Goals

The PartyVote visualization is designed around 4 major design goals: system transparency, social awareness, intuitiveness, and enjoyment. The ultimate goal of PartyVote system is to give each participant a voice in the song selection process via interacting to an intuitive and fun system interface.

The process defining how songs are chosen and the order they are played in should be made apparent by the visualization (transparency). We believe that transparency will both help reduce user frustration, increase enjoyment, and help users make informed voting decisions when using the system.

PartyVote is specifically designed for use in informal social groups rather than for personal use. Subsequently, awareness about the decisions made by other users is paramount. Strategic and “competitive” voting are both possible when awareness of other users is provided. Furthermore, we believe that knowing how other users voted can help initiate conversations, or provide a topic for discussion. The PartyVote system should not be intrusive, however, and we designed the system to minimize the required number of computer interactions. PartyVote is designed to help social interactions, not be the focus of them. The PartyVote voting system should ideally give each participant a voice in the song selection process. Additionally, all songs that are played should be optimally appealing to all users. Each song played minimizes the distance metric between itself and each of the user selected songs (as defined by the convex hull of the selected songs in the music space). Thus, party goers make computer mediated decisions about the ideal music to play so that individual tastes are satisfied, but frustrations about the music are minimized.
Informal visualization systems should not assume expert users will be the primary adopters, or that every person who uses the system has seen it before. PartyVote is designed specifically to be intuitive and require no training. Our visualizations are intentionally chosen for clarity rather than purely for their ability to convey information. Traditional music library navigation techniques and familiar analogies are provided to ground our interface in well understood concepts.

Finally, a major PartyVote design consideration is enjoyment. It will be very difficult to convince people of PartyVote’s utility if it was frustrating or annoying to use. Target users of our system will be relaxing and socializing with their friends and it seems reasonable that an aggravating program will quickly be abandoned in such an environment.

A user interacted system cannot be user-friendly without an intuitive interface. PartyVote visualization clearly indicates the current song being played, what each participant’s selection was, and what weighting each song has. After seeing how others have voted, some users will choose a song in the middle of the selection space because they know such a song will not offend many attendees. People with more discerning musical taste may strategically vote to try and alter the shape of the selection space in their favour. Knowing how the system is affected by each vote, and how people voted can act as a point of discussion for people looking at the visualization. This may also serve for healthy competition among party goers.

\section*{3.2 Voting System}

After all song weights have been assigned, the guaranteed to play songs define a song space. All songs outside this area are never played while the probability of a song in this region being played is directly proportional to its weighting. Song lists are generated using random selection without replacement and guaranteed songs are asserted to be played within the first 2/3s of the allocated event time. Other songs in the song space are referred to as potential songs.

\section*{3.3 Visualization Design}

The PartyVote visualization uses an outer space analogy to convey relational information about a music library. Songs in the library are represented as "planets" and "stars". Both guaranteed and potential songs are represented as planets, which can be played during the party. All other songs (non-playable songs) are represented as stars, which will not be played. Each planet is given a particular weight by the voting system. This weight is directly represented by the size of the planet. The probability of a song being played corresponds with the weight given by the voting system, thus, larger planets are more likely to be played. The distance between two planets is determined by the similarity of the corresponding songs. The more similar two songs are, the closer the two corresponding planets/stars are. We attract attention to the currently playing song using a moving spaceship icon. Selected songs also attract attention by an animated ripple surrounding the planet/star.
All 1607 songs are displayed in the music space as points in the 2D plane. Initially, the sizes of all songs (points) are the same. Songs voted on by the user have the point size increased and similar songs that can potentially be played have an increased point size. The sum of user vote weightings provide each song with a total weight. This weight is directly represented by the size of points. The probability of a song being played corresponds with the point size. The similarity between a selected song and related songs is represented by the distance between the points that represented selected songs and similar songs respectively.

A high dimensional convex hull algorithm is used to isolate the area in music space that encompass all songs weighted greater than 0 (see Figure 2). This will define the potential song space. The visualized song space has a black background color, while other space is coloured with light gray. We believe the black background will make song space more salient but also doesn't distract users [7]. Each song that is not selected but similar to some selected song is associated with the most similar selected song. Guaranteed songs are first randomly distributed over the first 2/3s of the playing time. All other songs are then played between the selected songs with playing probability based on song weight.

Guaranteed and potential songs are colour coded to indicate the participant who voted for that song. Each participant is assigned a non-replicated color as his/her identity. When participants vote for a song, the color of the planet changes to the voter's identity color. If there is more than one vote for the same song, the color of the planet will be the same color of the dominant voter, i.e. the voter who gives the most weight of this song. Currently, in our system, we limit the total number of voters to 10. We have ten distinct colors for user identities. According to Ware [7], ten or fewer colours can be identified pre-attentively, thus, participants should be able to observe the song they have voted for quickly. Planets that have been played will lose their luminance, making them blend in with the black background. Users can hover over all planets/stars allowing mouse hover queries and pop-up information about the song, artist, album, genre and voted weight is provided. The user is also able to zoom in and out of the display using a mouse right click. The zoomed in view allows users to see songs in more detail. This functionality is particular useful when users are voting for an item by clicking on its corresponding point.

The song playlist progression is represented by a spaceship travelling from point to point. The ship will park on the point until the song ends, and then proceeds to the next song in the play list. We believe this spaceship will make the visualization more interesting and may encourage participants to predict which song will play next. Users will not be able to directly interact with the spaceship since that will inadvertently give them control over the play list.

Users can list all artists, albums, songs and genres in our music library by simply clicking a corresponding radio button (see Figure 1). For instance, if radio button “Album” is clicked, all the albums in our library are listed in the left dropdown box. If one album in left dropdown box is further selected, all the songs belong to that album are listed in the right dropdown box. At the same time, all the songs are highlighted by animation in the
visualization. Similarly, if a song is selected by clicking on the planet/star icon in the graphic music space, the name of the song will be listed in the left dropdown box. This two-way brushing and linking functionality makes our tool more intuitive.

Figure 2. Overview of PartyVote Visualization

[1] Guaranteed Songs
A song which has been voted for by users and will be played in the party. The size of the planet icon relates to the weight allocated for this song.

[2] Potential Songs
Songs that are similar to a selected song are coloured based on the most dominate voter.

[3] Non-playing Songs
Songs that are outside the songs space / convex hull are not played during a party. Non-playing songs are represented as small light gray star icons.

The convex hull, the black polygon shown in Figure 1, represents the boundary between guaranteed and potential songs, and unrelated songs. This shape clearly identifies which songs may be played in the context of the entire music library.

[5] Left Text Song Selection Box
This box displays a list of possible songs, albums, artists or genres (depending on the radio button chosen).

[6] Right Text Song Selection Box
Presents a list of songs dependent on the left selection box [5].

[7] Radio Buttons
These buttons enable users to determine their library search strategy.

[8] Search Field
Users can type in the name of an artist, album, song, or genre, and the corresponding search result will be displayed in [5].
[9] **Vote Button**

Users can vote for a song/artist/album by pressing this button.

[10] **Spaceship**

The song play list progression is represented by a spaceship traveling from point to point. The ship will park on the planet that represents the song currently being played until the song ends, and then proceeds to the next song in the play list.


Three buttons, Play/Pause, Stop, and Next Song, are used to control the song player.

[12] **Attention-Attracting Animation**

Artist, album or genre selections in [5] select all corresponding planets/stars. Selected planets/stars are highlighted by animating ripples. Mouse over animations also display animated ripples.

[13] **Pop-up Information of Currently Playing Song**

Detailed song information about the currently playing song is displayed on a pop-up window. Song information such as song name, artist, album, length, the total weight and individual voting weights (with corresponding voter names) are displayed.

[14] **Mouse-Over Song Information**

When the mouse is over a planet/star, song information is displayed in a pop-up tool-tip.

**Zooming**

Figure 3 illustrates the zoomed-in view of the music space by right clicking on a spot that is close to the spaceship. Right clicks zoom into the area where the mouse is located. Subsequent right clicks zoom out the view back to the normal visualization.
3.4 Implementation

The goal of our implementation approach is to build a lightweight system, and we were willing to simplify implementation details at the cost of computational performance. There may still exist some space for improvement of real-time performance in terms of implementation, however, for the purpose of this project, our implementation approach is technically sound.

3.4.1 Usage Specification

The PartyVote tool is aimed for use in moderately sized established social groups. We expect the system to scale to thousands of songs being visualized for approximately twenty users over an effectively unlimited time period. For the purpose of this project, we limit our system as follows:

- 10 concurrent users (voters or people at the party). We are currently using color to identify people. Future implementations may use colour and a unique symbol/identifier on each song point to increase this number. Voter identification does not need to be preattentive, but voting colours need to be differentiable.
- A music library of 1607 songs
- The ability to generate 6 hours of continuous non-repeated music with 10 votes.

3.4.2 System Hierarchy

Our system has a three-tier hierarchy. The first tier is the user interface/visualization, which deals with all the user interaction with our system. The second tier is the system core acting as the brain of PartyVote, which processes all user requests and provides the feedback information for updating the user interface/visualization. The third tier is the back-end of the system. Our music library contains large amount of metadata, which need to be processed in advance to ensure real time performance. The back-end consists of helper classes and code for a priori data processing and helper functions.
Figure 4. PartyVote System Hierarchy. Objects in the blue dashed box indicate tier 1 user interface/visualization classes, yellow boxes constitute the tier 2 system core, and all remaining objects represent the tier 3 back-end of the system.

The system hierarchy is illustrated in Figure 4 in more detail. The user interface/visualization is indicated in the blue dotted line box. Objects that highlighted in yellow belong to the system core, the “brain”. MDSteer, DistanceRecorder together with the RevisedLibrary_ascii serve as backend.

Tier 1: User interface/Visualization
Visualization is the essential part of our system that displays system information and receives user requests. Tier 1 contains a fairly large number of components: the visualization container holds two sub visualizations: the space visualization panel and the text visualization panel. The space visualization panel is the graphic music space that displays all the songs in our library as well as voting information for each selected song. It communicates with other components such as StarVis, PlanetVis and SpaceMouseListener, which support the various functionalities in the music space visualization. The text visualization panel is a smaller text space in which users can list all the songs, albums, artists and genres of our music library. Search for a particular song, artist or album in the text visualization panel are provided by the “go” button. Pause/play, stop and skip buttons help control the song playback (although their use should be discouraged to avoid “cheating” like skipping other people’s songs). Most importantly, the panel contains the “vote” button used to vote for an item.
Tier 2: System Core
The core of PartyVote system consists of four components, EngineInitializer, EngineFront, SoundController, and VoteSystem. EngineInitializer is a lightweight object acting as the system initializer. It initializes the EngineFront and the VisualizationContainer and also builds the communication channel between the two objects. The EngineFront is the coordinator that talks to both tier 1 and tier 3. All user interactions and user requests from the visualization are processed in EngineFront. The EngineFront coordinates vote functionality, metadata reading, and sound player control. Finally, SoundController controls and coordinates music playback for the system. It also provides currently playing song information.

Tier 3: Back End
Our music library contains 1607 songs, however, the size of the song distance file increases rapidly as music library gets larger. For a library with n songs, a distance file has n² entries in it. One of the system goals of PartyVote is to be scalable. Therefore, the optimal option is to process the large amount of metadata information in advance to ensure the real time performance. The third tier, back end, is used to process the metadata information. RevisedLibrary_ascii.txt is the music metadata text file generated by iTunes (www.itunes.com). DistanceRecorder calculates the distance measures between every pair of songs based on the information in RevisedLibrary_ascii and saves the distances in the file SongDistance.txt. MDSteer [8] reads in SongDistance.txt and produces a set of coordinates for all the songs in our library. These coordinates can be used to plot songs in a 2D space and coordinates are stored in the file RevisedSongLayout2.csv. The logic behind tier 3 is explained in more detail in the implementation details section. SoundController controls the theoretic mp3 player (not implemented in our system). It retrieves the play list from EngineFront and coordinates the mp3 player to play songs.

3.4.3 Language, External Systems and Platform
All PartyVote code is written using Java and Java swing. Java offers extensive debugging tools that greatly reduced development time. MDSteer [8] is a multidimensional scaling tool that we used to convert the multidimensional song metadata to 2D x-y coordinates for each song. We also attempted to include an open source Java mp3 player, but this is not currently included due to time constraints. Our system currently runs on a P4 Windows XP PC as a customized Java application.

3.4.4 Data
All 1607 songs are from David Sprague’s personal music library. The input metadata is stored as a tab-delimited file, which contains 26 categories of song metadata, of which we used 14. Song metadata used in our system is limited to generated iTunes metadata comprising of: song name, artist, composer, album, genre, track number, track count, year, date modified, date added, bit rate, kind, comments and play count. Using text based metadata simplifies our coding implementation while still providing our system with song specific high dimensional data to conduct multidimensional scaling on. Song similarity from signal processing analysis is beyond the scope of this project [4].
3.4.5 Implementation Details

The biggest challenge encountered when building our system was converting the text based multidimensional song metadata generated by iTunes to the planar coordinates of each song plotted on our graphic interface. Song positions reflect song similarity. The more similar two songs are, the closer they are plotted. To solve this problem, two questions must be answered:

1. How can we lay the multidimensional datasets on a 2D plane?

Fortunately, a multidimensional scaling tool can do the job for us. MDSteer reads a distance measure file of multidimensional datasets as input and produces a file containing coordinates for each song. The distance measure file contains all distance between any two objects that intended to be plotted. For instance, if we have three multidimensional data (say a, b, and c) that we want to know their 2D coordinates, the distance measure file should contain the distance between a and b, a and c, b and c. To use MDSteer we developed an algorithm that calculates the distance between any two songs in our library.

2. How do we calculate the distance between two songs to reflect the similarity among songs (similar songs have a shorter distance)?

We have 14 pieces of metadata information that are used. The similarity of two songs should be determined by these metadata. String metadata either needed to be an exact match or the number of characters in common was used as a similarity metric. Metadata number similarity was based on the percent difference between the numbers. Among all 14 metadata entries, some fields are more significant than others for determining similarities. We assigned different weights to each metadata category and normalized the sum of the weights to between 0 and 1. The distance measure between two songs is 1 minus the normalized similarity value.

After the above two questions are answered, we use our algorithm to calculate the distance measures between each pair of songs and then have the MDSteer produce the coordinates of each song on the 2D plane.

3.4.6 Music Space

Among 1607 songs, users definitely have preference to some songs than others. Our music space should clearly reflect songs being selected and songs similar to selected songs. The “gift wrapping” convex hull [1] algorithm is employed to define our music space. After all song weights have been assigned, a high dimensional convex hull envelops all guaranteed songs. All songs outside this hull are never played while the probability of a song in the hull being played is directly proportional to its weighting (planet size).
4 Application

4.1 Scenario

Lucy enjoys hosting house parties. Although Lucy is not very discerning about the music she plays, her younger brother John likes to act as party DJ and he has a specific inclination towards country music. Most party attendees do not enjoy country music and this leads to arguments between the siblings. Lucy now regularly makes a play list from her music collection, approximating which songs her friends will like most by checking the latest hits online and asking them what kind of music they’re in the mood for. She then prevents anyone from being a self-designated DJ in an attempt to ease social tensions.

The PartyVote system will help Lucy keep the peace during the party and still let her brother play a couple of country songs. It will also save Lucy the time of having to make a playlist she “thinks” her friends will like. This system will let her friends vote for exactly what they like and it will ensure that nobody designates themselves the DJ during the party.

4.2 Using PartyVote

Lucy has no specific song preferences so she uses the PartyVote tool (with David Sprague’s personal music library included) to help her decide which song/album/artist will offend the least number of people. The tool displays which songs have already been voted for and which are most the popular (by the size of the planet icon). Furthermore, songs in the center of the convex hull are considered acceptable / pleasing to the most number of party goers. To get information about songs already voted for, Lucy moves the mouse over the visualization and a pop-up information box appears as she moves over each represented song. Lucy notices a large planet in the convex hull, hovers over it and learns that it represents the song is “Take me out” by Franz Ferdinand. She selects the song and votes for it, which gives it a greater weight and improves the weighting of songs that are similar to it. In this way Lucy’s vote attempts to appeal to the greatest number of people and to keep her guests happy.

John continues to want to listen to country music. Since all country songs are outside the convex hull, it is more efficient for him to search for his desired song rather than using hover queries. He types “favorite” into the search bar and presses the search button. Two songs are displayed in the text selection box: both sung by Neko Case, but each version is from a different recording session. John is unsatisfied with the results because he knows Neko Case actually released 4 versions of this song; 3 live performances and one studio recording. He’s looking for the studio recorded version of ‘favorite’ so he types in a search for the album “Blacklisted”. This outputs a list of track numbers (the song names were not added to the music library); he selects track 8 (causing the planet representation of the song to be selected) as he knows this album well and knows this is the studio recorded version of favorite.

Once John has voted for his country song, it appears in the convex hull as a large planet, and he’s assigned a specific color (e.g. pink). Songs related to the one he voted for get
highlighted in pink but are much smaller in size. John wants to see what other songs the system linked as similar to what he voted for, so he right clicks mouse over his voted pink planet and gets a zoomed in view of his music choice. Now he can hover over the other smaller planets to find out the names and details of the songs the playlist generated to be similar to his. This gives him something to anticipate during the course of the party. It also enables John to potentially discover new songs he may like.

### 4.2 Performance

The PartyVote system is an interactive real time system designed for visualizing large music libraries. On a Duo Core Centrino laptop with 1GB of RAM, the system shows few moments of lag and appears to have a refresh rate of over 20 frames per second (as the visualization appears to animate smoothly). Every song in the music library has to be assigned a distance measure in relation to all other songs before it can be input into MDSteer. Although this process took less than 5 minutes to output all the distance measures, as the music library expands the processing time to write this file will increase exponentially ($n^2$ entries in the distance file for every $n$ songs). Our current 34 megabyte song distance file generated expands to over 143 megabytes when David’s entire 3200 song mp3 music collection is included. Processing time to generate this full size file also increases to over 40 minutes. However, increases in library size should not affect regular system performance because the distance measures are processed a priori. Regular system performance should degrade linearly with linear increases in library size, however, the current design implementation is not optimized for performance and some algorithms used may not linearly scale. Currently, the system operates with little to no noticeable lag.

### 5 Discussion

Although the PartyVote system implemented for this study is not a complete product, it proves that it does enable users to collaboratively generate the playlist in an informal gathering while still being exposed to new music. The system was implemented for 10 users with 1607 songs in the music library; however, this can be scaled to much larger dimensions of users and songs. As mentioned in section 4.3 the time taken to process distance measures between songs will increase exponentially but this only affects system setup procedures. Although the growing number of users and songs will create visualization challenges (discussed in section 5.2), these can be addressed as the system evolves.

The PartyVote system is one of a kind in that it allows groups to determine the type of music playlist generated and safeguards individual preferences and choice. Although several existing systems enable informal groups of people to work together to explore music databases, all of these systems require regular program interactions. PartyVote incorporates a playlist visualization with an “outer space” analogy, using planets and stars to represent songs and a spaceship to indicate the currently playing song. We believe our visualization is intuitive, non-obtrusive and fun to use. By using votes to generate a playlist, individuals have a say in the music played with minimal system interaction required.
Initially we intended to use a solar system analogy as our visualisation, where a voted song would be represented as a sun and similar songs would be in orbit around the sun as planets. Planet sizes would vary with their weighting and distance from the sun would indicate similarity between the planet and the sun. However, this analogy can be misleading as planet distances may not accurately reflect song similarity. We instead opted for a simple “outer space” analogy. Although this new analogy is not as intuitively clear, the information conveyed is more accurate.

5.1 Strengths of PartyVote
PartyVote uses a fun and interactive visualisation tool to help users decide what to vote for and to explore a music library. The use of planets to represent songs, with a spaceship’s random flight pattern determining which song plays next is intuitive; users will not require tutorials or extensive training to understand the visualization and how to use it. Planets increase in size as they receive more votes, which may assist indecisive voters on what to choose. Clearly demonstrating group preferences can help party attendees make more informed decisions. The search function also helps simplify voting. When a user selects the artist/album/genre/song radio button on the interface, and then selects a text item, corresponding songs in the visualization are also selected. This will help the user see whether the songs in the search already exist in the convex hull, or understand how the distributions of songs from an album or an artist are. Similarly, clicking on a planet with the mouse means that the corresponding song appears in the text visualization. We believe this brushing and linking allows users to leverage the best features from both text based song selection approaches and from music library visualization approaches.

Interactive features of the system include a zoom in and zoom out feature, allowing users to see selected clusters of music in more detail, or to see an overview of the music space. This is useful for music discovery. A user sees which songs the system calculated as being similar to their choice. Roll over pop-up information is also provided. When a user moves the mouse cursor over a planet on the visualisation (inside or outside the convex hull), an information pop up box appears on screen, providing information about the song name, artist, album, play time, and voting weights. Voting awareness helps make system functionality more transparent and facilitates social awareness about other people’s preferences.

As mentioned above, the number of users and songs can be increased in the system, making it scalable to any number of users and a continually growing library of songs. The PartyVote system stands out among other music visualisation tools because it generates a playlist based on user votes; increasing the likelihood that more users will listen to more preferred songs during the course of the informal gathering. Although the system does not make use of collaborative tools such as touch screen displays on interactive table top displays, this system is aimed towards informal social gatherings when a PC is far more likely to be found. The use of a simple desktop computer is therefore more practical for our system. Mouse and keyboard use is also more intuitive for most computer literate people.
5.2 Weaknesses of PartyVote

There are several identifiable problems with the party vote system. The increase in either the number of voters or the number of songs in the music library will cause visualisation problems. As the number of songs in the music library increase, the visual distinctiveness of each song will decrease, visual clutter will increase, and songs may begin to occlude each other. As the number of users increases, the colors used to represent each voter will become less distinctive and in turn discernable. However users can still roll over a planet and find out which song they voted for by finding the planet it represents. Finally, when multiple users vote for the same song, its size increases to a point where it occludes other planets from view completely. Occlusion and clutter problems may be solved using a more elaborate data navigation approach such as overview + detail. Such a technique seems extraneous for our current system specifications.

Metadata errors and inaccuracies are perhaps our system’s greatest weakness. At least five albums from our current library contain tracks labelled “track01” or similar track names. This means that the album “Blacklisted” by Neko Case is closely related to Sigur Ros’s album “(“), although musically they have little in common. Similarly, missing or incorrect metadata will result in our distance metric being inaccurate. These problems can be solved by manually confirming the metadata entries for each song in the library, but this is extremely labour intensive. Other similarity measures based on sound signal analysis may be more accurate as a result, but such techniques are difficult to implement and have their own set of problems.

The system’s zoom in and out functions do not use focus plus context and could confuse and disorientate users if they continually zoom in and out of the visualization. We assume the visualisation will be zoomed out the majority of the time and that the zoom in function will be used simply for users trying to vote or for people looking for their assigned color. We also note that users do not have the option of choosing their own color to represent them on the visualization. Choosing their own colors will help users identify their voted song and similar songs on the display easily. It may also increase the system’s “fun factor”.

We have not conducted a user study to test our system so many of our claims may be unfounded. However at this stage of development we are simply focused on testing whether playlist generation based on user votes was actually possible. In our future work we intend on improving the system, collect user feedback and conducting several case studies.

The positions and relative distances between songs in the visualisation were determined by normalizing all the metadata of all the songs in the library. This means that our final layouts are highly dependent on our similarity calculations. The importance of each metadata entry was judged by the authors and metadata weightings were based on these assumptions. We did not use any other approach to parse the metadata and develop coordinates, and we have no benchmark metric for judging out heuristics. Currently many albums, such as Revolver, contain songs highly distributed across the song space. This is not accurate.
Currently the PartyVote system is not connected to an actual mp3 player. We intended on using an open source java based mp3 player to integrate with our visualization and engine, however due to time constraints and insufficient documentation for the java based mp3 players, we were unable to connect the two. Currently, the system simulates how the visualization would work if it was playing real-time music, including timing events based on each song’s play time. We intend on connecting PartyVote to an mp3 player in our future work.

6 Lessons Learned

Perhaps the biggest lesson learned from this project is to keep our implementation plans simple and testable. Our initial project goals aimed at 2-3 visualizations, but in the end we had difficulties finishing one visualization in time. We were extremely fortunate to plan out system design details well in advance, and to adhere to these plans. We frequently had to re-synchronize our code and consistent coding structures reduced (but did not eliminate) synchronization overhead.

Our project relied heavily on 3rd party software for music playback and multi-dimensional scaling. Although source code for their projects is not always possible, additional time for finding appropriate systems and debugging these systems should be allocated in the future. We could find no simplistic mp3 Java players at the time of this paper, and MDSteer had 3-4 known crash bugs that took time to work around. Metrics for code correctness before and after external code use should also be implemented. It is currently impossible to judge how appropriate our distance metric calculations are before MDSteer performs the layout.

7 Future Work

In the immediate future we hope to integrate and test a Java based mp3 player for our system. Although this project can be considered a proof of concept, music playback and improvements to the graphics, the distance algorithm, and the voting system mechanics would make PartyVote far more appealing. Graphics improvements would include a better looking space ship, enabling the ship to orbit planets, and more visually interesting planets (not just coloured circles).

Later research on PartyVote should include the addition of other visualizations/analogaies such as pinball analogy and driving analogy described in our project proposal. User testing is also critically important as we have no metric on how intuitive our system is, whether people will enjoy using the system, or how social interactions are affected by MusicVote. An initial 4 hour case study examining a group of 10 users socializing with the PartyVote system present would be ideal. The visualization should be visible during half the party and removed in the other half. Pre and post study questionnaires should also be administered. From this case study future studies can be more accurately designed.
8 Conclusions
In this paper we have discussed the PartyVote system and described the various design decisions made. We believe that by linking the text visualization panel and our visualization, our system leverages the strengths of both text based music library selection and music library visualization systems. Furthermore, PartyVote is intuitive and the mechanism underlying the voting system is transparent to users. Finally, the automated playlist creation based on user votes minimizes user interactions with the system, meaning PartyVote facilitates song selections for closely knit social groups without being central to socializations. The interface is designed to be pleasant and intuitive but the voting system allows party goers to have a say in the music selection without having to spend all their time in front of a computer screen. We are not central to a party; it's just designed to make it more enjoyable.

9 References:


