Contents

Analysis

Introduction to the problem

I want to undergo the development of an introspective adaptation that is targeted to Windows but not specifically exclusively to Windows, of Chris Danford's 2001 arcade style rhythm/dance game: "StepMania", which is one of the earliest forms of VSRG. In StepMania's gameplay, the difficulty selection of maps has entirely no difficulty calculation system and it is up to the map creator to determine and input the map difficulty itself during the map creation (I will discuss the features of map creation in my essential features later). This is problematic as the difficulty is subjective and prone to human error and will yield to dissatisfaction from the user. My adaptation intends to add a difficulty calculation system and to improve the problem of chart difficulty calculation of most modern VSRGs through an algorithm to determine arrow patterns prone to being miscalculated and introduce a reduction/promotion system of the difficulty calculation based on these factors, whilst also maintaining the regular core aspects of gameplay. This will be beneficial as it will add accurate difficulty calculation and allows players to indicate their true ability in their gameplay without their scoreboard performance being inflated or

deflated. This will also prevent mis-ranked players from taking advantage of maps with miscalculated difficulties (this is often referred to as “farming”).

Stakeholders

As VSRGs have grown in popularity since their birth in 1998, there is a varying age range and demographic of players who play them. For my game there will be two main audiences (stakeholders).

Early VSRGs from the late 1990s and early 2000s such as StepMania and Konami’s “Dance Dance Revolution” (DDR) (more examples will be included in research), typically have a much older age range since most players were in their teens, i.e. 13-18 years old, when they first played them during the era they were released and have either continued to play them as they have got older or have returned to playing them when they have matured. This means the age group of i.e., 21-30+ years old, will be included as part of my Stakeholders. The reason being is that the much older generations return to VSRGs is due to feelings of reminiscence and/or the nostalgia of the gameplay. Another factor is that during the 1990s and early 2000s, most VSRGs like Konami’s “Beatmania” Series were only playable in arcade dance machines and the computers themselves that were required to run these games were not a common household item due the expensive of owning a computer. Often the older age range continue to play the much older VSRGs as a hobby or a way to pass time and shy away from long hours of play. This is due to older VSRGs not integrating online features and therefore not having competitive scoreboard rankings of the much younger modern VSRGs. This is also since the older age group have less free time due to increased responsibilities such as working for a job and providing for their families (more on this later).

However more modern VSRGs such as osu!mania that were released in 2010 and onwards (osu!mania was released in 2007), have a much younger age range as since most players were born the time they were released and have picked them up as they became teenagers, during the 2020s modern era. This means the teenage age group of i.e., 12-18 years old will also be part of my Stakeholders. From the teenage perspective, there is more free time due to decreased responsibility. This means the younger teenage demographic is more likely to play VSRGs for longer periods. This is evident in the fact that many content creators, a famous and popular example being “jkzu123” on the platform “YouTube” (Channel link may not be feasible, however at the time of writing this, the link was active: https://www.youtube.com/channel/UCeL9uQhZ8WsXFxGvQCDzrYw), have arisen in number and gained a large audience due to consistently having the time to play and upload content on VSRGs. jkzu23 even stated he began playing VSRGs “since I was 12 years old” which further solidifies my belief. Furthermore, due to the teenage population being more

socially interactive due to things such as education and increased time, teenagers are more likely to share VSRGs with their peers, thus leading to an increase of the younger audience playing them. All this evidence suggests that there is more of an audience for the younger generation of VSRG players.

Therefore, I intend to adapt my game to be inclusive for both younger and older audiences. My adaptation will try to cover all the age ranges of VSRGs and appeal to both audiences. This includes the much older age range of 21-30+s of older VSRGs from the 1990s and 2000s and the much younger age range of 12-21 from 2010s and onwards. I will do this by including the nostalgic and retrospective arcade aesthetic of the much older VSRGs i.e., Dance Dance Revolution but also adding the modern-day functionality and mechanics that are appealing to the younger audience that play VSRGs i.e., osu!mania. I also intend to keep my focus on the fact that older VSRGs typically have features (as mentioned and will discuss later) that the much larger younger generation consider to be "flaws." This is due to the modern technology improving from the time of the older generation and VSRGs themselves improving also.

To give an insight into the benefit of my solution, I will represent and outline real-life stakeholders from the stated audiences and demographics and explain the certain features that will benefit them and features that will fit their needs as shown in the list below. I will also interview the stakeholders and include their results of their perspective on VSRGs:

- Nathaniel Binuhe is a peer in my computer science class, and he picked up playing the modern VSRG: osu!mania, during the pandemic and has continued to play them ever since. He enjoys playing the game and often plays it in his spare time (as mentioned earlier that the younger generation has more spare time). He represents the much younger audience that play VSRGs in their teens. However, he also states that the "grading (difficulty calculation of charts as mentioned earlier) system" of osu!mania and other VSRGs is "inconsistent" and is not satisfied (as mentioned earlier). Therefore, my solution of an adaptation using an improved algorithm to create a better difficulty calculation system for charts will benefit him and the rest of the younger audience as part of my stakeholders as a consistent and accurate difficulty calculation system will cause less dissatisfaction and improve their experience playing the VSRG.
- Maria Sheehy is representative of the upper bound of the older generation as she recalls playing the original VSRGs such as Dance Dance Revolution from the 1990s. The most notable aspect of is that she played them when she was 28 years old when they first came out meaning that from the time, they came out to the time of

writing this, she has significantly aged and mostly reminiscent of the game. Maria also stated, “I used to play those games with my daughter” and “my daughter still plays those games with her kids as well.” This gives an insight into the point that the upper bound of my game audience will be 30+ years old. Maria also states that she only plays VSRGs like Dance Dance Revolution, “when we go the arcade” and “which is a couple of times a year.” This further signifies my earlier claim of the older generation only playing these games to “live in the past” and reminisce and do not play these games often for prolonged periods albeit to the younger generation. This is due to Maria having more responsibilities such as caring for her family (as mentioned earlier). Therefore, it will benefit the older generation of my stakeholders if I retain the aspects of the arcade style graphics to provoke feelings of nostalgia in my audience. However, a major drawback to my adaptation is that it will not be able to support dance pads (more on this lay) which is fundamentally one of the core reasons of feelings of nostalgia, therefore keeping the style of the arcade style user interfaces.

- Jean Obungu also grew up playing rhythms when she was a teenager and thoroughly enjoyed them. She recalled that you had to “hit the notes to the beat really fast.” She also recalls that she played it with her older brother and said, “I remember having a lot of fun.” Since then, she has now aged and relinquished from playing the game and feels “nostalgic” about them and further iterates “it was fun at that time.” She further represents the older generation of the stakeholders who are reminiscent of the nostalgic 2001 arcade-style graphics from early VSRGs. Therefore, keeping the aspects of original Stepmania/Dance Dance Revolution game screen and menu user interfaces will benefit the audience of the older generation and fit their needs.
- Samuel Smedley is also a fellow peer of mine who I personally played VSRGs with during the pandemic era of 2021-2022. He initially picked up the popular VSRG “Friday Night Funkin’” at its peak of popularity in late 2021 and advanced onto osu!mania as he was further interested in playing VSRGs. Samuel Smedley recalls finding osu!mania “hard but really fun” and said he “likes the cool graphics and skins” (Which I will further explain in detail in the similar systems section of research later). He is representative of the younger teenage generation who picked up VSRGS in their spare time and enjoyed the much more modern implementation of VSRGs. Because of this it will be beneficial for my adaptation to retain the modern useability aspects of rhythm games such as Quaver and osu!mania (which I will point out in the research section).
- Louis White is also representative of the younger generation who has had experience playing arcade style games, as well as experience playing the VSRG: Harmonix’s “Fortnite Festival” and is interested in rhythm games and contemporary

games in general. Louis represents the more casual group of the younger audience who play for shorter periods of time. When asked what Louis's intention of playing games was. Louis responded with "Just to pass time, whenever I have free time." This further shows my point of the younger generation playing due increased free time.

- Harrisson Jarvis has had experience in playing rhythm games such as Beat Saber and Harmonix's "Fortnite Festival" (The same game studio that made the VSRG Guitar Hero). Harrison is representative of the younger generation that finds rhythm games very exciting. Harrison even stated that he found rhythm games like Beat Saber, "very immersive."
- Hamish Lindsay has also had experience playing VSRGs such as Friday Night Funkin.' He stated it was "fun and challenging" and had experience playing it when it first came out. However, he pointed out that there are certain aspects that he did not like about the game and even other aspects of VSRGs. Hamish represents the VSRG players who have had experience with VSRGs and recognized their flaws.

Research

Ever since their origination in 1987, dance/rhythm games, modern examples being Ubisoft's "Just Dance" to Harmonix's Guitar Hero (more on this in my research section) have been prominent in today's modern game industry. In online communities, a certain sub-genre of rhythm game(s), commonly referred to as a Vertical Scrolling Rhythm Game (VSRG) is a type of rhythm game in which the user must yield input and "hit" icons (commonly in form of arrows) "on time to the beat," on par with a set of stationary arrows (commonly referred to receptors), synchronous to music.

In VSRGs, sequences of arrows (interchangeably referred to as notes) can be arranged to form patterns. These sequences of arrows are reflective of the music's tone and beats per minute (BPM). For example, a fast-paced song's "beat drop" will have multiple arrows sequences that must be hit quickly, whilst a slow based song's break will have fewer arrows and are hit slowly. As the arrows scroll, there are normally a stationary set of target arrows. Once the arrows meet the stationary set of arrows, the user must hit the corresponding arrow via their input device. As music progresses, arrow sequences may vary in pattern and build up to form maps of arrows (interchangeably referred to as charts or "beatmaps"). As a user plays a map, the user is evaluated on their accuracy of how on time they hit the note coordinated with the beat of music. Each time a note is hit, it is given a judgment of accuracy based on how accurately they hit the note, or if they even hit the note at all. The user is then given a performance "grade" based on the average accuracy of how on time they hit the notes. This forms the basis of most VSRGs. Different songs can

have different maps. Within these maps the arrow sequences can vary, some being harder to hit than others. A song can have multiple maps with varying arrow sequences. This forms maps with their own corresponding difficulties. As VSRGs have progressed and allowed multiple songs with maps with multiple different difficulties, these difficulties are given a value and ranked quantitively, e.g., Difficulty 1, 5, 11, or qualitatively. easy, medium, hard, expert.

Various research resources of VSRGs, of which two examples being on Dean Herbert's "mania" game mode of his game "osu!" (Referred to as "osu!mania") and on Swan's "Quaver" (I will discuss and analyze modern examples of modern VSRGs in my research in more detail later), have become online and have increasingly large online communities. This has led to VSRGs becoming competitive with scoreboard rankings based on performance and the difficulty of maps achieved. As a result, typically in modern VSRG's such as osu!mania, the value the difficulties calculated is based on the average density of the sequences arrows in a map. This means the quantity of arrows that are to be hit in a certain amount of time (typically per second).

However, this approach is problematic as it has caused incorrect estimates of the chart difficulty. Incorrect estimations of chart difficulty can make maps seem harder or easier than they are. This has led to inflated or deflated scoreboard ranking; Thus, players being mis-ranked on online competitive scoreboards. Often mis-ranked players take advantage of maps with mis-calculated difficulties by specifically playing those maps only. This further inflates their scoreboard ranking, giving an incorrect measure of their ability. This leads to dissatisfaction and anger in the online communities for VSRGs and even leads to players quitting the VSRGs they play. (Evidence of this will be covered later)

Therefore, this is why it is a sensible approach to develop my adaptation to be better suited for map difficulty calculation by introducing a

VSRGs and rhythm games are currently amassing a large amount of traction and revenue during the modern era of gaming due to their intricate action-based game style. This can be shown according to Wikipedia's analysis (which can be found by in the follow link: https://en.wikipedia.org/wiki/Friday_Night_Funkin%27) of Ninjamuffin99's popular VSRG "Friday Night Funkin'", revealing that the game's Kickstarter funding was able to "reach[ed] its goal of $60,000 within hours." and in the end the "Kickstarter ultimately raised over $2 million". This then led to an article report from IGN (which can also be found in the following link: https://www.ign.com/articles/kickstarter-record-number-games-2021) stating, "that Friday Night Funkin': was one of the most funded Kickstarter projects of 2021". Another example of the popularity and market revenue of VSRGs can be seen in another Wikipedia analysis of Dance Dance Revolution

(https://en.wikipedia.org/wiki/Dance_Dance_Revolution) states that “In 2004, Dance Dance Revolution became an official sporting event in Norway” and that through recent years Konami (the brand behind the game) dedicated “their own competitive tournament, [:] the Konami Arcade Championship”” allowing “different regions around the world to sign up and play in specific online events to earn a spot in the grand finals, typically held in Tokyo, Japan.” The article also states that this led to countries like “Korea, Taiwan, and other Asian countries” were “allowed to enter,” then in “February 11, 2017”, competitors from the “United States were eligible” and in 2020, “eligibility for players in Australia and New Zealand” was available. The article then states that competitors have moved on to win the “global tournament.” This signifies the international outreach of VSRGs alike and considering that StepMania is based off Dance Dance Revolution itself, a modern-day adaptation will undeniably have the capability of amassing popularity. Furthermore, the article states that “In March 2023, the first ever *upbeat* tournament was held at Round1 in Denver, Colorado” with a “$10,000 prize pool”. This further shows the resolution of popularity and revenue that VSRGs have the potential to create. This further justifies my reason for making an adaptation.

A list of the similar systems that I will research, some already mentioned, include Friday Night Funkin,’ osu!mania, Konami’s Dance Dance Revolution, Roxor Game’s “In The Groove,” and Guitar Hero. I have chosen these systems as all these games still retain the core functionality of a VSRG, but each has their own characteristic and customized “spin.” Despite this, each has their fundamental flaws within their games. To provide evidence of what features of my adaptation are to be prioritized and what features are to be ignored. To this I gained insight from the interview results of my stakeholders and described their thoughts on the different similar systems.

Before I delve into the research of similar systems, it is fundamental to highlight the core aspects of StepMania and the reasons for a newer adaptation.

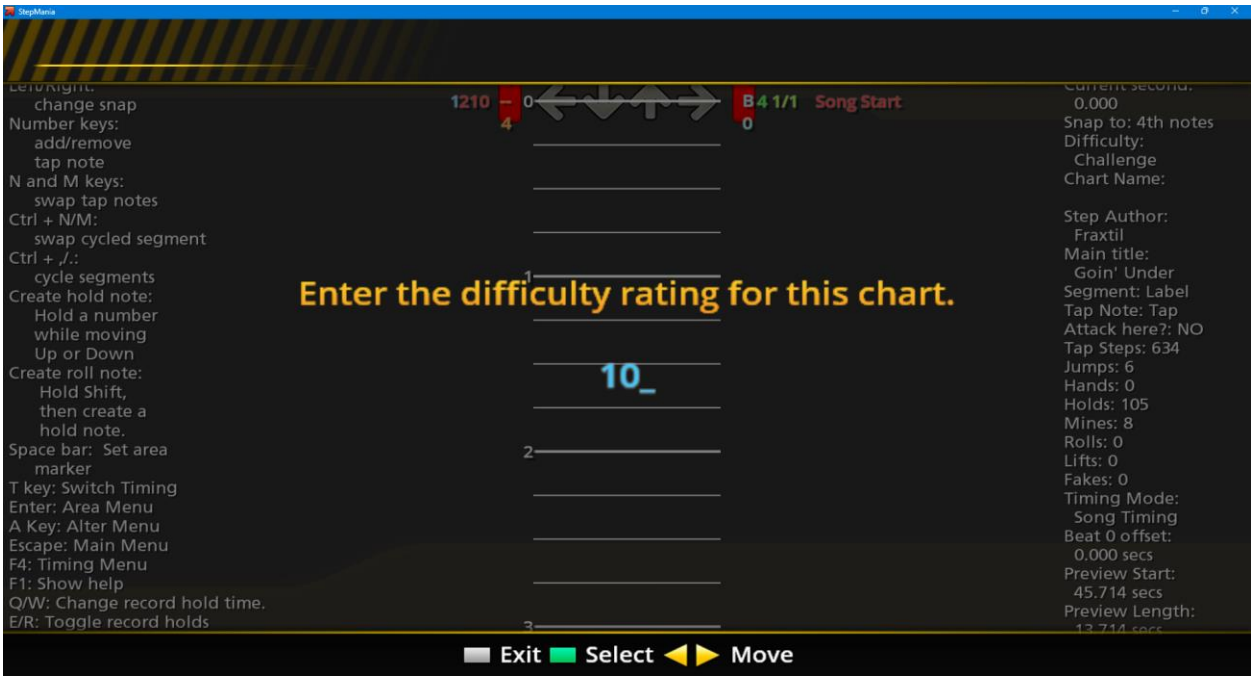
StepMania

StepMania consists of the basic VSRG gameplay and is tailored to both the arcade and keyboard users. In the gameplay there are customizable settings, navigation systems to play maps and an area to play songs. The game features multiple game modes that are selectable and multiple modes to play with more than one play cooperatively.

To gain insight into the experience of the gameplay, I interviewed Hamish Lindsay. Hamish played a song on StepMania and immediately noticed a difference in gameplay. Hamish stated, “The game(StepMania) felt “unresponsive” and “I feel like there should be more visuals to audio feedback”. He then noted that “It is a lot different to the newer games

(VSRGs) I played." This is because StepMania's gameplay is more tailored towards This shows evidence of the immediate differences between older and modern day VSRGs and how much older VSRGs still have fundamental flaws to their gameplay.

Of these fundamental flaws is the map editor system. As mentioned earlier, although there the system allows for users to create customized maps, the difficulty calculation of the map is entirely up to the map editor/creator to input in themselves which leads to problems mentioned earlier of incorrect estimations of the difficulty. Figure 1 shows a screenshot of what entering the difficulty of a map scenario would look like.



Furthermore, I showed Hamish the system of map editing/map creation and asked him the follow question:

- What do you think about the map editing system?

Hamish responded by saying "It can lead to human error" and "alongside human error, there is also bias" "A lot of experienced players are more likely to play a higher rated map, for example rated 10 rather than 1, which leads map creators to inflate the difficulty to play them." This shows the fundamental flaws of being able to input the difficulty and clearly identifies the issues of bias and misjudgment. The is due to every map creator has had different learning curves whilst playing the game, therefore perception of difficulty is relative; Somone else's perception of being difficult may be different to another. This suggests that in my newer adaptation of StepMania there must be constant, quantitative measures to determine difficulty to avoid misjudgment and bias.

Osu!mania

As mentioned earlier, Dean Herbert's osu!mania is a popular online VSRG with a competitive scoreboard ranking. This is evident in Osu!'s official country rankings website (https://osu.ppy.sh/rankings/osu/country) that show, at the time of writing this, that there are currently 981,025 active users in the United States alone and a total of 24,202,202 total registered users, as seen on their home page (https://osu.ppy.sh/). On top of this there is a specific scoreboard ranking for osu!mania as seen in the osu!mania rankings page (https://osu.ppy.sh/rankings/mania/performance), showing that the higher your performance score is, the higher your scoreboard ranking. Thus, giving osu!mania the most competitive nature of modern VSRGs. This is a crucial piece of information as the score of your performance, or as it is termed in Osu! as "performance points" (commonly abbreviated as "pp") you gain is proportional to the difficulty of the charts you play. This is evidently shown through Osu!'s official FAQ (https://osu.ppy.sh/wiki/en/FAQ#scoring) on performance points, which states "The easiest way to improve it is to score high on difficult songs and playing more songs." Often higher-ranking players must continually play difficult maps to maintain their status due to this very reason. Furthermore, in Osu!'s Wiki[1], it describes a "weightage system" to "prevent the rapid and repeated gaining of lower pp scores" on "easy beatmaps" by "reducing the amount of pp that is gained." This means that a percentage of pp is lost the lower the rank the map is, meaning a pp score that ranks second place in best plays will be lose Further suggesting that the only way to increase your "pp" is to play maps continually and progressively with increasing difficulty.

1 https://osu.ppy.sh/wiki/en/Performance_points/Weighting_system

All this evidence suggests that the difficulty of the maps a user plays is a key factor. Therefore, the calculation system to determine the difficulty is substantial.

Before the map difficulty must be calculated, the map must first be made in the map editor system. Osu!mania contains a section of the game in which audio files can be dragged into the game. Upon doing this, a screen allowing the user to enter the song details and customize the metadata of the song appears. Afterwards the user is taken to an area where they can place notes and navigate through the different sections of the song in a customizable manner. After the user is satisfied with the way they have arranged the notes for song.

This means that my adaptation must contain a system to create and adapt maps in real-time . My adaptation should also include a map selection system to select maps to play and an audio file management system. Much like osu!mania's system, my adaptation must also contain the ability to edit maps metadata and background images. Furthermore, Osu!mania's map creation system does not contain an autosave system meaning you

must save maps manually. Therefore, my adaptation may add a system to automatically save maps whilst editing them. In osu!mania, once the map has been saved (often manually), The difficulty of the sections of the map is calculated.

As mentioned earlier osu!mania uses density system to do this. The vertical column with pink and white rectangles indicates the sequences of arrows that are within the song as it scrolls. The vertical bar with yellow bars indicates the density of the patterns (notes per second). The taller bars the denser the pattern. If the bar turns pink this signifies an extremely dense
sequence of arrows.

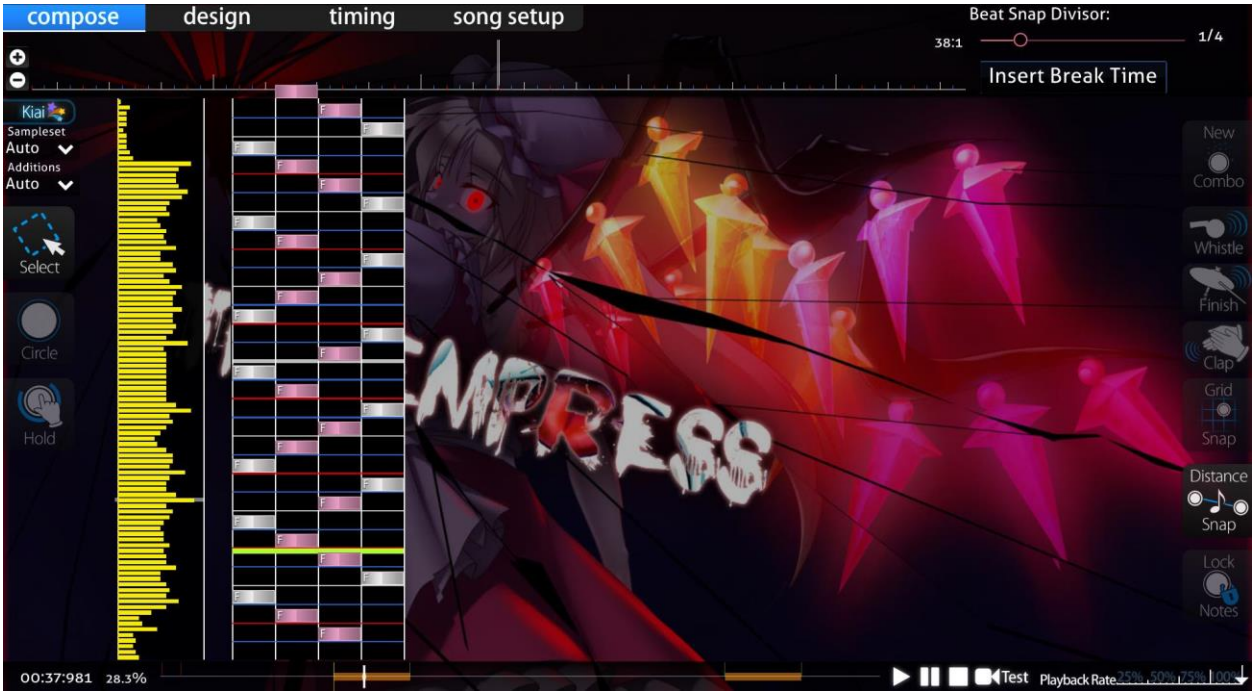


Figure 1 - An osu!mania beatmap in map editing. This reveals the inner workings of the map and the density of the notes. Yellow bars show density.

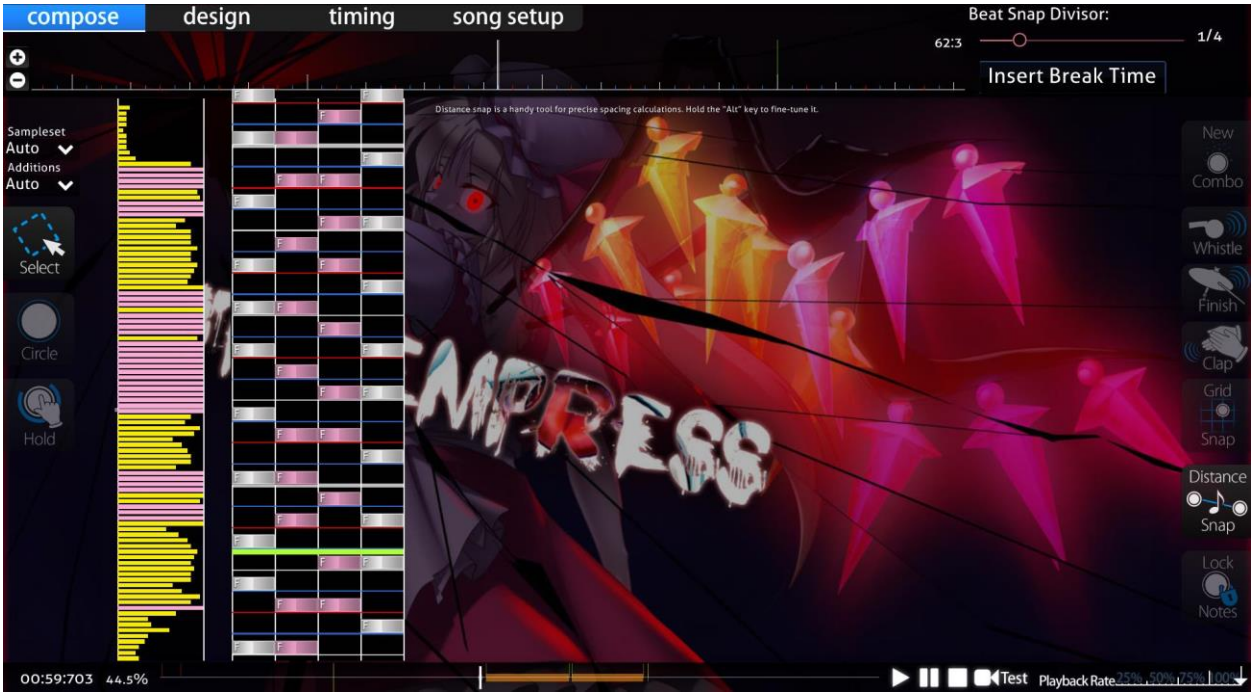


Figure 2 - The same beatmap but a higher difficulty. The pink bars show the extemely dense nature of patterns

Figure 3 shows the nature of maps with high difficulty requiring more dense patterns for longer periods of time. This is shown as most of the map is pink with extremely dense patterns.

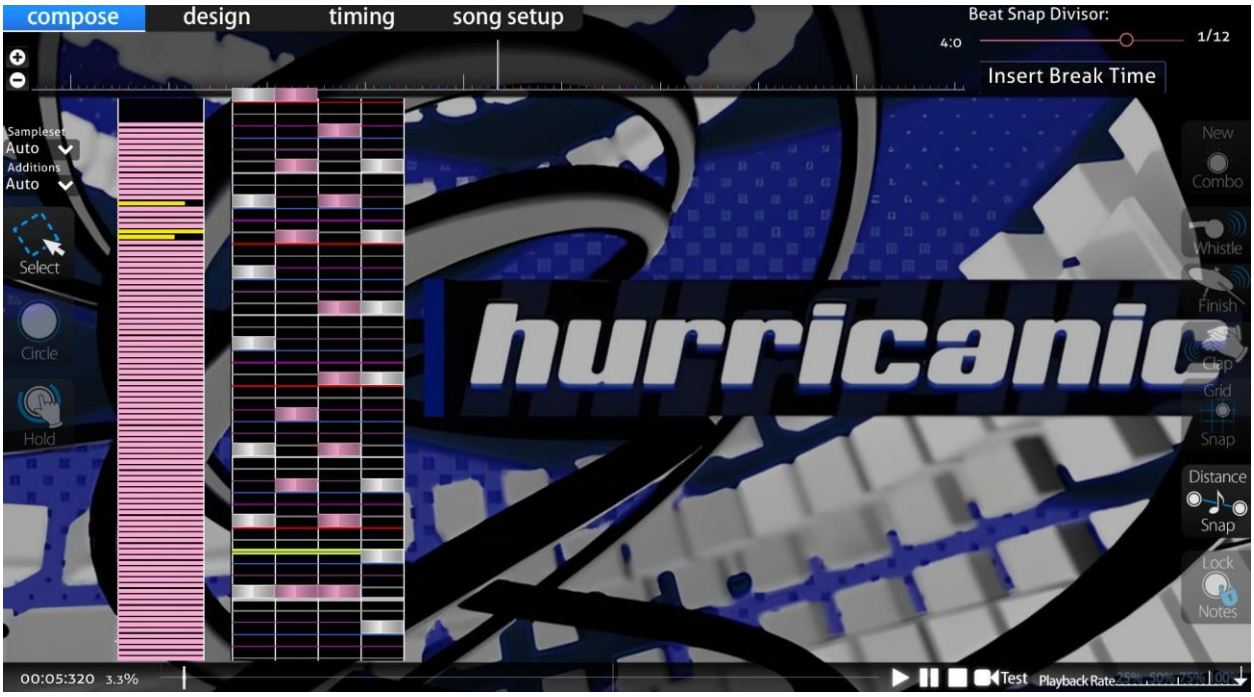


Figure 3 - The pink density graphs of beatmap with 5-star difficulty rating in osu!mania.

There are flaws to this system. This is evident in the fact that Osu! have tried to improve their difficulty calculation system by releasing an update to the difficulty calculation as seen in this official article ：https://osu.ppy.sh/home/news/2022-10-09-changes-to-osu-mania-sr-and-pp. This already gives an insight into the issues that are faced with difficulty calculation. Despite these changes, the difficulty calculation flaws have remained and are still prominent to this day.

To describe the problem with this system and show evidence of the issue of difficulty calculation, I interviewed Nathaniel Binuhe (as mentioned earlier). Nathaniel agreed to play a couple different beatmaps on osu!mania to discuss the problem of incorrect difficulty calculations. To start, Nathaniel played a song that rated 3.88 stars but with a high BPM of 270 and with technical patterns (see earlier sections). The star rating, BPM, artist name, source (record label), beatmap creator, overall difficulty, long note count (will discuss this later), key count, etc. Can all be seen in the top left corner of the map selection as shown in Figure 4 and 5.

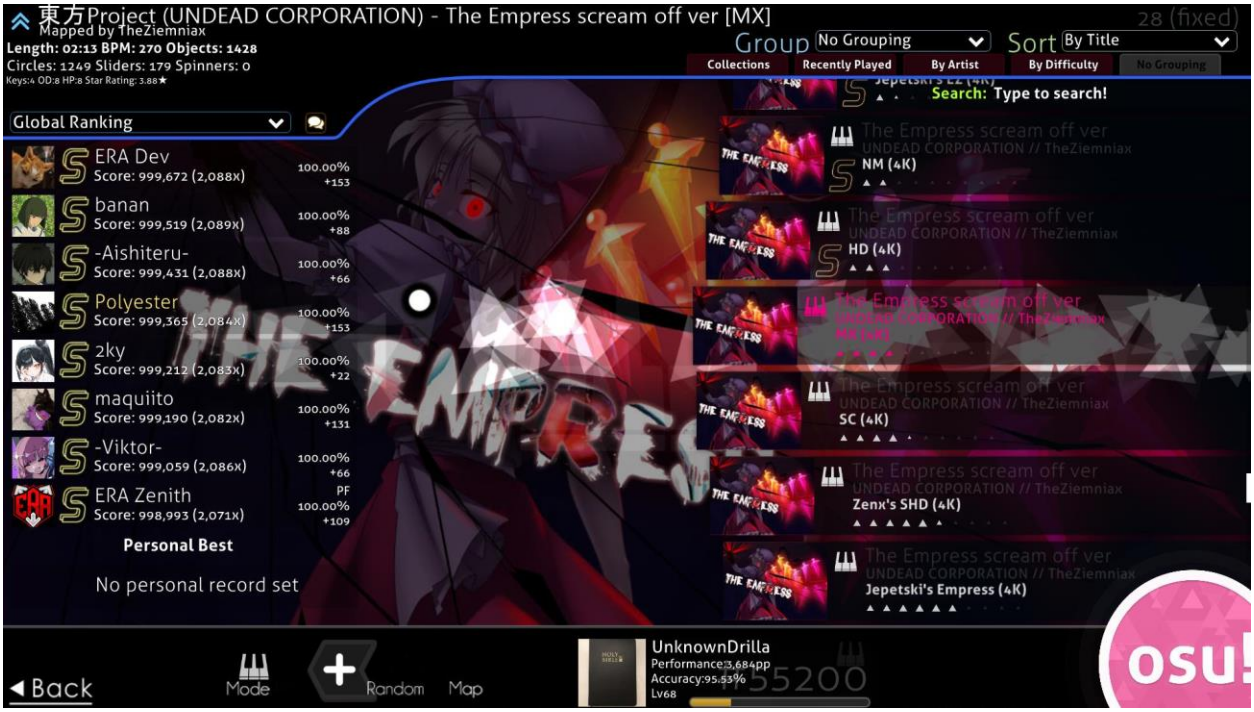


Figure 4 - Map selection screen of osu!mania. Top right shows details and difficulty calculation.

Figure 5 - A zoomed in photo of the beatmap details located in the top right corner of the map selection.

Nathaniel then played a song rated 4.27 stars with a lower BPM of 140 and full of patterns that are prone to being miscalculated. He played the first map and could not complete it. He stated it was "much harder" because of the "higher bpm" and the "insanely fast and hard to read technical patterns." He also stated that "the first map should be harder (have higher difficulty calculation) than the second map," even stating that the first map "should not be 3.88 stars." This signifies the misleading nature of osu!mania's difficulty calculation and the need for improvement on difficulty calculation. On the other hand, the second map, although having a burst of technical patterns that are prone to being miscalculated, Nathaniel stated, "it is much easier" and "slower due to the lower BPM." He was surprised stating "the second map is harder is than first map," further stating that "It should not be harder than the first map." Overall, Nathaniel stated that the main issue with the game is the "misleading ratings." Despite the fundamental flaw of the difficulty calculation, Nathaniel still stated that the game is "still fun to play" and commends the "nice modern graphics" and the ability to "add skins."

A discussion on Osu!'s official GitHub (https://github.com/ppy/osu/discussions/12980) can be found to show a user named "BrokenGale's" disclosure of the need for "osu!mania star difficulty improvements", proposing a new and "augmented star rating for mania".

Thus, it is evident to see the potential issues of mis-judged rankings that can arise, especially if there is a malformed difficulty calculation system to determine the difficulty of maps. Therefore, It is justifiable to identify my adaptation's approach of a new difficulty calculation system.

Friday Night Funkin'

Ninjamuffin99's Friday Night Funkin' is one the most popular and prevalent VSRGs of the modern-day era. It is widely renowned for its unique cartoon style graphics and its upbeat and catchy video game style music. This can be shown in two reviews on Metacritic (https://www.metacritic.com/game/friday-night-funkin/) by a user under the alias of "izack" states that the game has "banger songs" and "the sprites are fire". EndlessFlame928 also states that "The mods are keeping the game alive" and "the songs are a banger". This shows that the more modern and younger audience prefer a more modern, retrospective style of interface. This contrasts with older VSRGs such as Dance Dance Revolution and Stepmania which include more of perspective 3D style of interface. To get an insight into the modern graphics that are common in VSRGs I interviewed Samuel Smedley by asking him a series of questions:

- Did you like the cartoon art and graphics of the game?

- Did you like the music and the character sprites?
- Did you think that the difficulty of maps was harder/easier than they should be?
- Did you feel like it lacked better difficulty calculation/assumption of charts?

Samuel Smedley then responded, “I like [the] cartoon art because it has a unique style that’s extremely vibrant and exciting.” This shows the evidence of the younger generation having more of an appeal of more modern and smooth graphics. Samuel also disclosed his taste for the music of the game by stating “The music is very high energy which fits the setting of each “map”.”

Therefore, it is evident that an approach of keeping the more modern useability and functionality of the user interface but at the same time maintaining the older arcade style 3D renditions of older VSRGs is an identified and justifiable approach to my adaptation. 3D re

Samuel also responded with criticism of “The character sprites I think lacked variety in how it was the same for every map and situation which left more to be desired.” Samuel’s criticism justifies the need for an ability to integrate customized design in the creation of maps. This will allow variation and not leave users with

Alongside the graphics, Samuel responded by stating “the very high energy… doesn’t always translate to the charts as the notes were not always synchronized.” This adds evidence to my previous point of VSRGs sharing a widespread problem of difficulty judgement and as Nathaniel mentioned earlier, “inconsistent grading.” Samuel also expressed “For a beginner player, the difficulty is a good standard but quite quickly after a few hours the hardest difficulty is too easy.” This led Samuel into suggesting “There should be a setting which progressively makes the encounters more difficult” . This suggestion shows the effect of underestimation of difficulty calculation and further signifies why a quantitative measure of difficulty calculation system must be approached. This is evident in the difficulty selection screen for songs as shown in figure 6. The difficulty system has no calculation and is based upon the game developers and mod developers to determine the difficulty. Another factor is that there is no quantitative measure. As mentioned earlier. the difficulty system of Friday Night Funkin’ is based on qualitative measurements e.g. “Hard”, “Medium” and “Easy”, that are relative to perspective and somewhat abstract. These measurements are prone to being misjudged generally and sometimes may give an incorrect indication of the map. Thus, leading to inconsistencies. This means that my adaptation must implement a quantitative representation of the difficulty.

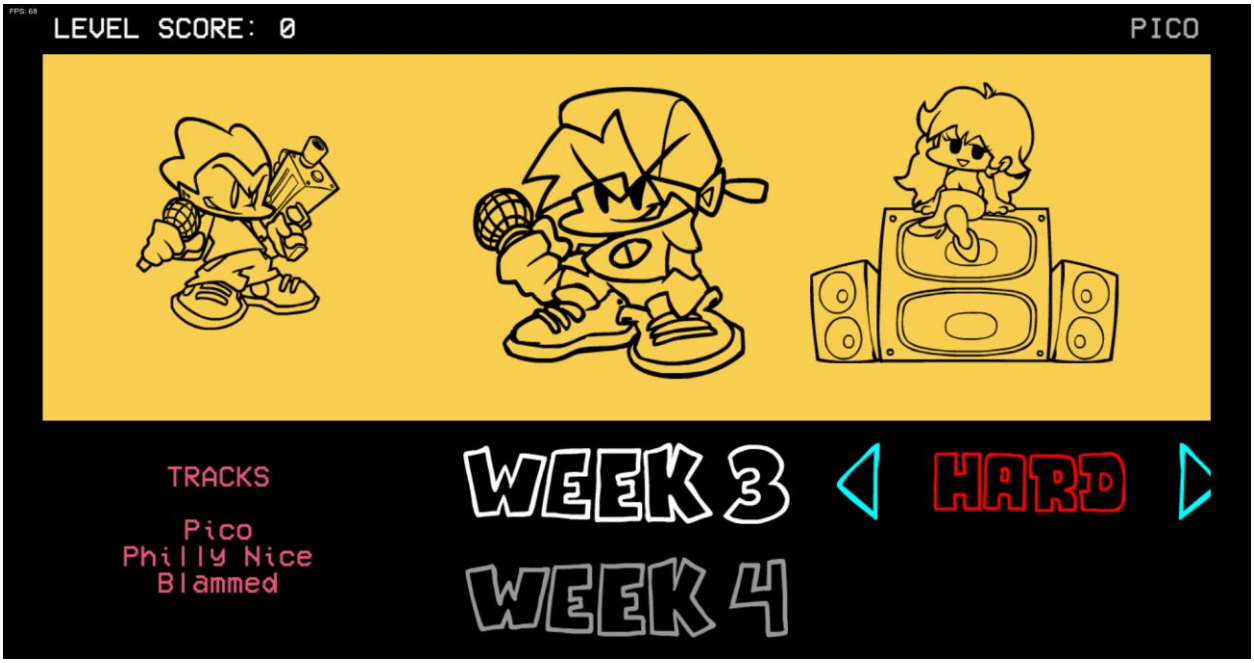


Figure 6 - Abstract difficulty calculation system of Friday Night Funkin'

Dance Dance Revolution

The significance of the success of DDR can be seen in Wikipedia's "Similar Games" section in their article about DDR (https://en.wikipedia.org/wiki/Dance_Dance_Revolution), stating "Due to the success of the Dance Dance Revolution franchise, many other games with similar or identical gameplay have been created." This evidence shows that the onset of multiple VSRGs are a result of DDR and therefore the basis of VSRGs after DDR took inspiration from it. This is also evident as the article mentions "Fan-made versions of DDR have also been created... The most popular of these is StepMania." This indicates that core aspects of early VSRGs such as StepMania originated from Dance Dance Revolution as they were either fangames or spin-offs of the game.

As a result of my interview with Maria Sheehy, stated that she is "reminiscent" of the game and particularly the art style of the older VSRGs. Figure 7 shows an insight into the style of the graphics on Dance Dance Revolution. It consists of less cartoon stylized graphics as seen in Friday Night Funkin' and more solid 3D text and a realistic perspective. Therefore, for my adaptation to maintain the older arcade style of older VSRGs, 3D aspects of text and design shall be an implementation of my adaptation's user interface. To implement these 3D aspects, I must program and utilize a 3D rendering system into my game engine that will allow rendering of 3D and allow transformations in the z-plane.

As well as the user interface, DDR is notoriously known for its dance machines that are required to play the game. However, I do not intend to add this functionality as the younger audience of my stakeholders do not wish to have this functionality and prefer more simple forms of playing VSRGs. This is evident my interview with Louis White. Although Louis stated, “I enjoy going to the arcade,” he feels “not everyone has access to an arcade in their local area” and that “the cost of going to an arcade may not be worth it.” He then stated, “playing at home with a computer is much easier and saves time.” To further elaborate on Louis’ point, an article from Kineticist[1] states that the average cost of a DDR machine is “$3,000-$20,000+” and a new machine is upwards of “$9,000-$20,000+”. This shows if the younger generation would want to play for longer durations personally, they would have to deal with the grievous expenses of buying a dance machine. This is evidence to suggest that integrating features of dance machine hardware into my game would not benefit my stakeholders needs.

1 https://www.kineticist.com/post/buy-a-ddr-arcade-machine#:~:text=TL%3BDR%20on%20how%20much%20a%20DDR%20machine%20will,a%20game%20room%20retailer%20like%20Game%20Room%20Guys.

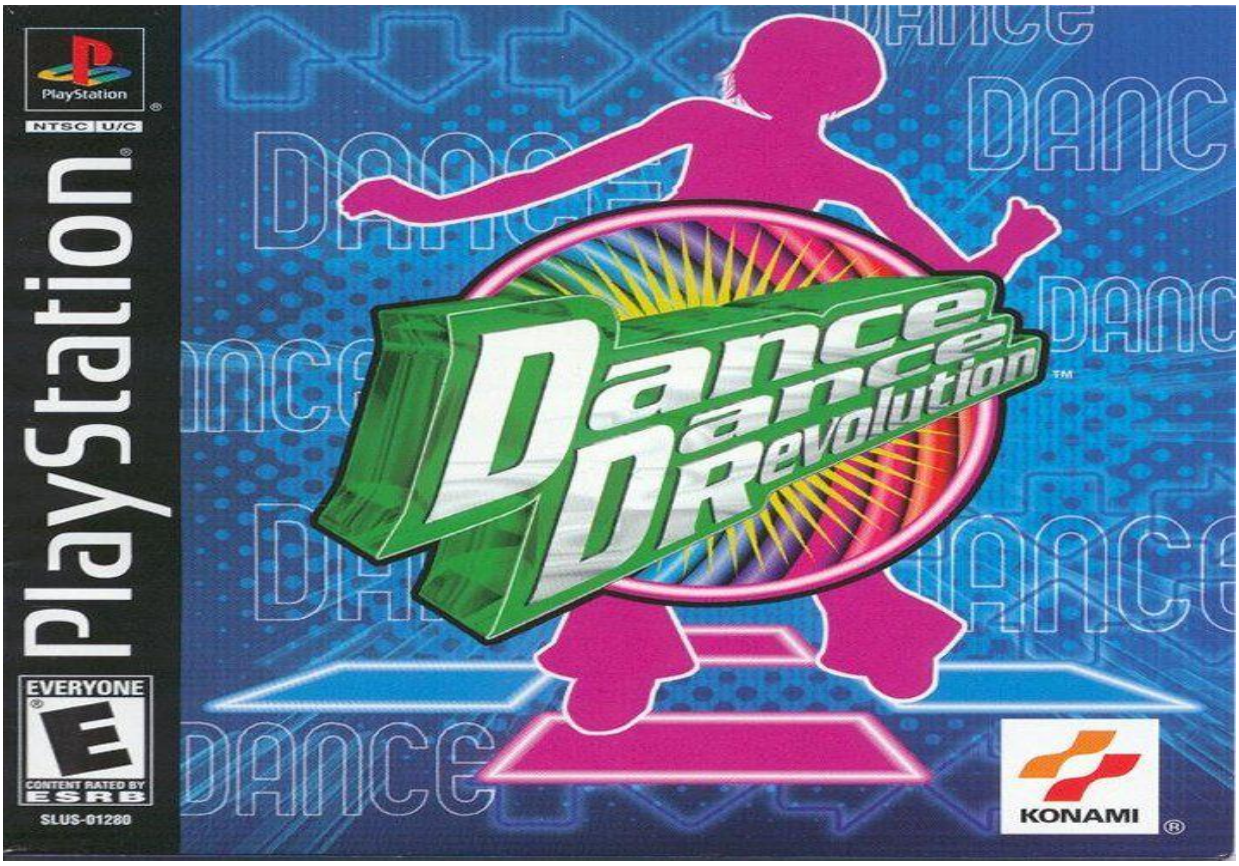


Figure 7 - PlayStation Release of Dance Dance Revolution's game cover.

In The Groove

To delve deeper into the graphical design of older VSRGs, I researched Roxor Game's In The Groove, commonly known as ITG on online forums. ITG is another PlayStation 2 clone of Dance Dance Revolution and was released shortly after. ITG's use of 3D aspects in their logo design, gameplay and menu give the game its nostalgic acquisition of the general style of arcade games of the early 2000s. I believe this is thoroughly shown thrown ITG's PlayStation 2 game cover, which involves a 3D representation of the "arrow" notes (commonly found in the main gameplay of Dance Dance Revolution and other VSRGs) at a different perspective angle and with a 3D depth to its design. This really amplifies the early 2000s feel of the game and gives it a nice touch that is distinguishable from other modern VSRG's such as Friday Night Funkin'. Therefore, in my adaptation's system, including forms of 3D perspective shifts, as seen in the game cover of ITG and Dance Dance Revolution alike, will add to the general feel of mid 2000s VSRGs. These perspectives shifts may be included in micro interactions and parts of user interface that require animation and movement.

Harrison Jarvis has experience with the immersive 3D aspects of rhythm games such as Beat Saber. When asked about his take on the 3D immersion and perspective of ITG. Harrison stated that "It is not too crazy" and "I like the simple design of the logo and game as it makes the game style and gameplay more fluid." He stated, "I like the retro style of ITG and "It shows you what it is from the cover… you can tell it's a dance game." He concludes with "The "big arrow" reminds of the classic dance games of the arcades." Harrison's perspective further suggests that the simple retrospective 3D designs that use clear features relating to VSRGs i.e. arrows, people dancing, musical terms, etc. are what make arcade VSRGs so iconic and appealing. This is especially true for the older audiences as such distinct features are what causes feelings of reminiscence.

Figure 8 - PlayStation 2 Release of In the Groove's game cover.

Another aspect of ITG's style is its main gameplay's arrangement of arrows using different z-depths in the interface of the arrows and effects of outlining, shadows and in game lighting, which creates a 3D parallax/overlay effect that further enhances the 3D

retrospective feel of an early 2000s VSRG.

Figure 9 - In The Groove's main gameplay.

To achieve the desired 3D aspects and designs of older VSRGs, my adaptation must contain a 3D environment to render and perform the different perspectives shifts Therefore, I must use different computational 3D rendering techniques such as model and view matrices and matrix transformations with the use of perspective projection. This shall be done programmatically, and the 3D aspects shall come from coding the transformations instead of it entirely being based on design.

Computational Methods

Thinking abstractly

Areas of my adaptation will need abstraction to make the solution to complex problems clearer and easier to solve. Using abstraction will help in focusing on what a part of my code does and its function (what it will output), rather than how it works. This is through hiding the actual implementation of functions after writing them and then reusing them within larger sub-routines.

These abstracted functions will then help in piecing together solutions without needing to care so much about how each individual function works. This will benefit my process of development as solutions and methods shall be compounded and based only on the relevant information required for the specific problem. This means that during solving

problems, I will only require knowledge of what data needs to be input and what function is needed to process the data that gives an expected outcome.

This will benefit during debugging problems as I will have only the data, I need to solve the problem and do not have to dissect other irrelevant information. It will also greatly speed up the process of writing code itself and allows focusing on conceptualization of solutions to problems and other computational methods.

Abstraction will also benefit in areas such as making program code more readable and maintainable through the reuse of abstracted sub-routines. It will also benefit the users themselves as they will not need to understand the complicated processors of my adaptation's inner workings. Examples of how abstraction is used in my program include :

1) The use of external libraries, such as glm, to perform complex mathematical operations in accordance with development. I will need to use mathematical constructs such as:
    a) matrix transformations
    b) matrix multiplications
    c) vector addition
    d) vector subtraction
    e) matrix addition
    f) matrix subtraction
    g) multiplying matrices by a scaler
    h) multiplying matrices by a vector
    i) use of trigonometric functions
    j) compute the magnitude of a vector

All functions provided by the library will hide the unnecessary details of how to compute these mathematical concepts and purely focus on performing their intended tasks. Another factor is that library functions have been optimized and thoroughly tested by their developers. This will especially benefit debugging as it will indicate that the error is mostly likely to do with my own source code and how I used the function and not the function itself. Furthermore, the library functions are less likely to have large computational complexity. For example, in a standard matrix multiplication algorithm, to multiply two $n \times n$ matrices, it will require n^3 new multiplications of scalars and $n^3 - n^2$ new additions to compute its product. This means a standard matrix multiplication algorithm has an average time asymptotic (time) complexity of $O(n^3)$. However better matrix multiplication algorithms that are likely to be included in the library functions, have a lower asymptotic complexity of $O(n^{2.3751552})$, thus saving time and being less intensive on the CPU/GPU.

2) Navigation of the user interface - the user does not need to know the details of how the menu navigation system works, just the ability to know what inputs are required to move from one section of the game to another. I will require functions that integrate user interface controls that abstract the process of drawing/rendering triangles to on the graphics pipeline and load textures/images. These include:
    a) Functions/subroutines that abstract the updating of uniforms variables within shaders and GLSL files to be part of the 3D rendering process.
    b) Have function/subroutines embedded within rendering/drawing GUI classes to abstract the view/projection matrix and remove the need to attribute each matrix transformation when updating source code.
    c) Have reusable sub-routines to simplify and abstract the process of drawing GUI elements. For example, the entire process of rendering a triangle, texturing it, converting it from local space coordinates to v space coordinates and then implementing accessibility to the triangle to make it a GUI element, can all be simplified into one draw function e.g. drawGUI()
3) The use of external window rendering libraries such as SDL2 that will abstract the process of rendering a window on the screen. These libraries will simplify window creation and hide all the details that happen in the process of forming a window. Instead, I will only have to focus on a singular window creation function and the aspects of my window. SDL2 will also provide an OpenGL context for me to render in. Rendering in the OpenGL context is the focus of my adaptation as it is where every element of the user interface shall be rendered. Window libraries, such as SDL2, will abstract features such as
    a) Anti-aliasing process - the library comes with pre-defined anti-aliasing functions meaning the process of integrating it into my adaptation is thoroughly abstracted to just a few functions calls.
    b) Polygon line smoothing process – along with anti-aliasing , the smoothing process of lines when drawing triangle on the graphics pipeline will be abstracted and simplified. This is crucial for aspects of 3D rendition such as view models during perspective projection.
    c) Providing an OpenGL context – The library does not require me to implement me a 3D rendering context/environment myself
4) The use of an OpenGL toolkit/API such as glad that will abstract the process of implementing the specification functions/subroutine calls to the drivers that the graphics card supports. OpenGL is only a standard/specification and there are many different version of OpenGL drivers, therefore the location functions/subroutines to user OpenGL is not known at compile-time. This means it is up to the developers to retrieve the location of the specification functions/subroutines (typically in pointers)

and store them for later use. This process is cumbersome as you may need to retrieve the location for every function that has not been declared. However, these toolkits/APIs have abstracted the need to this. Examples of the abstractions due to toolkits/APIs:

a) Not having to focus on the entire process of retrieving the graphics specification function/subroutine location within the drivers and instead focus on the actual use of the subroutine. For example, the subroutine glGenBuffers (), would not have to be retrieved and I can focus on the use of it. It eliminates the unnecessary process of having to retrieve it beforehand.
b) The use of the toolkit/API means that the subroutine calls are updated and thoroughly tested. This provides access to the most up-to-date function/subroutine calls within the drivers and allows me to know worry with the function itself and instead focus my adaptation's code.

Thinking Ahead

Before designing my solution, I must think about the sub-routines required to form a working GUI navigation system as the majority of my adaptation's functionality and accessibility shall be through a GUI. Furthermore, I must think about being able to integrate the 3D aspects within a 2D GUI, to provide a multi-scene perspective. For this, I must ensure that my navigation system is fully responsive to mouse clicks on GUI elements and button presses such as WASD and the Left/Right/Up/Down arrows keys. I must also be able to integrate both functions at the same time to allow customization of keyboard input which is essential for allowing the modern functionality and usability of the adaptation.

I must think about the different reusable classes needed in forming a core GUI system that is maintainable, reusable and quick to form new sub classes such as different menu screens and game states like the editing/creation maps section. Features of the core GUI class/system include:

1. Universal functions that are inherited in all derived classes that are required to render the GUIs onto the screen. These functions will be reusable all throughout the GUI system and will simplify the process of making an interactable GUI. These functions include:
    a. A draw function to draw the GUI element on screen
        i. Within this function it will have:
            1. Position of GUI
                1) Converting local space coordinates to screen space
            2. Rendition of triangles and quads for the element
            3. Width and height of GUI element

            4. Percentage of fill of the texture
            5. Texture interpolation type (more on this later)
        ii. Within this function a function to load images for the texture
    b. A rotation function that integrates matrix transformations to be able to rotate the GUI without rewriting code
    c. A scale function that integrates the matrix transformations to scale GUIs accordingly
    d. A function to adjust the color, hue and alpha values of the textures and GUI elements without re-texturization.
    e. A function to draw untextured polygons and fill them with a color. This will benefit in making some UI elements for menus
2. An animations derived sub-class that packages and abstracts the transformations into more feasible general subroutine. These include:
    a. Flip the GUI element vertically and horizontally and on its axis
    b. Animate these transformations as transitions by implementing the animation functions as callbacks.
    c. A movement/transform animation that integrates a matrix transformation with iteration to give a "moving" animation on the GUI element. This will be useful during implementing gameplay.
3. Accessibility within the core GUI system to ensure useability of the GUI elements once it's been created. This includes:
    a. Checking if mouse corresponds to the range of the coordinates of the GUIs "size"
        i. Converting from local space coordinates to view space coordinates
            1. Vector and matrix multiplication
        ii. Collision detection algorithm

I must also think about the audio aspects of my game. To do this I must reuse a core audio class. This will also involve outputting audio through the user's sound system accurately and synchronously to the different aspects of gameplay. Features of audio and timing can include:

1) Starting and stopping the map's audio whilst playing a map on time to the input
2) Adding a preview of the map's soundtrack during map selection#
3) Ability to upload audio files for map creation
4) Ability to load different audio files for maps during the map selection in real time
5) Ability to preview sections of audio files for a short period of time during map selection.

Alongside audio, I must pair this with input timing at the same time. Audio and timing go hand in hand and are crucial aspects to the main gameplay. I must be able to utilize an input timing class that is able to correctly time the users input and give a valid response to their timing in real-time without any inconsistencies. This is so that users can receive accurate judgment based on their timing when hitting notes in gameplay. For the timing I must:

1) Form a system that will not have inconsistent input registration and be responsive to the different timings of input. This will be evident during the main VSRG gameplay.
    a. The system must be able to handle multiple keyboard inputs simultaneously and expect accurate output.
    b. Must be able to calculate the timing of the input to an accurate degree and compare the timings of the data within a map. This will involve real-time processing and will be used during the gameplay when analyzing hit time results. This will be crucial in determining hit accuracy (as discussed earlier).
    c. The system must be able to give the timing to a suitable degree of accuracy and have a universal accuracy gauge to determine the timing.
    d. An average timing system to determine the g

As well as classes and subroutines, I must think about the different data structures that I will use to implement the different aspects of my game. The data structures must allow me access data efficiently and store large amounts of data without affecting performance and main memory. Some of these data structures include:

1. Arrays – Needed to store vertex buffer data and objects to the data of the coordinates and vertices of the triangles that will represent interfaces and icons
2. Vectors/Lists/Dynamic arrays – During the process of storing the map files and comparing them to the input timing
3. Hash Tables – For providing fast access to maps whilst searching for them in

Thinking Procedurally

To form a stepwise approach to my problem, I must break my problem into more manageable sub-programs. Each sub-program is broken down into their own retrospective elements to the solution of the problems. Examples of the sub-programs include:

1) Enhanced difficulty calculation system to determine note patterns prone to being miscalculated
    a. Data processing algorithms and pattern recognition algorithms to determine the certain patterns within map files.

i. Multiples that contribute to difficulty calculation algorithms such as strain, readability, playability and speed, not just density alone.
    1. Process and read map data
        a. Have suitable data format for processing e.g. text files.
            i. Organize that data to be processed in a fashion such as:
                1. Note timings
                2. Column placement
                3. Beat snap
        b. Have predetermined data to signify certain patterns e.g. four continuous notes in the same column are considered as a “Jackhammer” pattern (This is the terminology used to describe repetitive notes in one column in VSRGs) and therefore is a strenuous pattern and should be a factor to increase the difficulty of the map.
            i. Count notes in adjacent and previous columns
            ii. Count notes in adjacent and previous rows
            iii. Measure notes timings
            iv. Count quantity of notes in each time frame
                1. Determine average quantity of notes throughout entire time frame (mean)
    2. Calculate the difficulty based on the data
        a. Add discrete factors such as readability and playability of the pattern and multiply it by continuous data such as strain and speed to give a final calculation
            i. Calculate averages
            ii. Calculate the magnitude of continuous intervals of data
        b. Add a deduction system for patterns that are prone to inflate difficulty
            i. A tier/ranking/hierarchy class system of all the patterns and their different presidencies of playability.
            ii. Contribute the averages of all these factors to prevent short bursts of difficulty spikes overinflating maps.

b. Processing of difficulty efficiently and quickly to factor in large quantities of maps.
    i. Efficient use of data structures
        1. Arrays
        2. Hash tables
        3. Linked list
    ii. Efficient use of searching and sorting algorithms

2) Adding 3D renditions to the adaptation's user interface and gameplay
    a. Use of perspective projection and matrix transformations
        i. Use of model, view and projection matrix in perspective projection
            1. Constructing an identity matrix (A $n \times n$ matrix filled with 1s diagonally)
            2. Applying matrix multiplication using the math library functions
                a. Apply multiplication to model, view and projection matrix
                    i.
        ii. Apply to the model, view and projection matrix uniforms within shaders
        iii. Allow multiple renditions of this process on different elements for reusability.
    b. Integrate this into the core GUI elements
        i. Have the updating of the projection matrix as part of the GUI class process.
            1. Potential inheritance or class association of the GUI and the graphics pipeline class

Thinking Logically

I will require logical thinking in the difficulty calculation system of my adaptation especially during the forming of algorithms when determining patterns. I will also need logical thinking for handling mathematical concepts and constructs throughout the code. Examples of logical thinking in the development of my adaptation include:

1) Determining the note patterns based on conditions such as position. My algorithms must be able to determine and factor in the positioning of the notes in their respective columns and rows.
   Here is just one example of a note pattern that can be determined using logical thinking. This is not all the examples however it does show the logical thinking involved with determining the algorithms.

a) The “trill” note pattern
    i) The use of if-else statements or switch/case chains to check If notes are alternating between columns on adjacent rows
        (1) “Trills”, where the first note begins on column *n with a column length of l*, where *n=1, l=4*
            (a) If the $n+1^{th}$ note from note n, on the $n+1^{th}$ column is on the $n+1^{th}$ row.
                (i) Check if note *n+2* is in the same column as note n and repeat checking of condition (i) until note n+k is not in the same column as note n.
    ii) To do this I must use while loops to count the notes until the condition is broken.
    iii) Once the condition is broken the count of the loop is I must set the strain factor to the count of the notes and total time taken of the map, *t,* from note *n,* to note n+*k*.
        (1) The use of multiplication

2) Determining whether the user is clicking/interacting with the user interface
    a) The use of if else statements in coordinate calculations to determine if they are overlapping
        i) Determine whether the user’s mouse is colliding with the GUI element
            (1) Check if the mouse coordinates lie in range with the GUI element’s “box size”
                (a) Calculating the difference between the GUI element’s center and the mouse position.
                    (i) Integrate magnitude checking using if statements
                    (ii) Translating local space coordinates to screen space coordinates
                        1. Multiply local space coordinates by model matrix to get world space coordinates
                        2. Multiply world space coordinates by view matrix to get view space coordinates
                        3. Multiply view space coordinates by projection matrix clip space coordinates
                        4. Pass in the result into the uniform variables in the shader files
                        5. Lasty set the position of the coordinate within the shader files
                    (iii) Determining the difference between the GUI element’s top left and the center of the element by finding the midpoint of the coordinates.

3) The use of switch/case statements to determine whether the inputs are within the different ranges of accuracy measurement. An example of this can include:
    a) Checking if the timing lies within millisecond intervals

i) For example, If the judgment window to achieve “Flawless” hit accuracy is 22.5 milliseconds and the input timing is 16ms then use an if statement to compare then
   (1) Use switch/case chains to compare if the input accuracy is equal to the judgment in real time.
      (a) Use of an array or hash table data structure to provide instant access to timing data to minimize delay

Backtracking

I will need backtracking for various parts of my algorithms in where I will need to sequentially go back to previous the note *n* from note *n+k* to determine the duration of the and pattern sequence and the time taken to hit the sequence.

I will also need backtracking to go back over the user’s gameplay data in their gameplay replay to determine an accurate “grade” based on the average count of accuracy measurements.

Heuristics

I will need Heuristics for forming new algorithms that are like each other. These algorithms will generally have the same format except the position of the columns are shifted. For example, the VSRG note pattern commonly referred to as a “jumptrill” is much like the note pattern “trill” except it consists of notes alternating in two adjacent columns instead of one adjacent column (An example of this would be two notes in the first and second column and on the next row, two notes on the third and fourth columns). The algorithm to determine whether the note pattern is a “jumptrill” will be very similar to the algorithm to determine whether the note is a “trill.” Therefore, taking a heuristic approach of going back and reusing the structure of the “trill” note pattern algorithm shall be beneficial to approaching the structure of the algorithm.

Divide and Conquer

I could potentially use divide and conquer in the process of searching during map selection. However, the use of data structures like hash tables may not require me to search for maps using divide and conquer due to their constant access time of O(1) and their ability to be accessed via a key.

Visualization

I will use Visualization during the map creation process. As mentioned earlier, many VSRGs such as StepMania and osu!mania use a visual representation of the notes to allow the editing of the map file. Despite the data itself being in text file, the sequences of notes

and their patterns are represented as adjacent icons in a horizontal row of vertical columns. The user can then place or delete notes by clicking on them. The user can also adjust the beat time of the maps and make the visual representation of the gaps between the notes smaller. This thoroughly helps in the map editing/creation process as it abstracts/simplifies the need to edit and place the map data in a file one note at a time.

Complex Calculations

My adaptation shall require multiple complex calculations, many of which are provided by external libraries. Many of these calculations have been abstracted and therefore do not need to be manually written. However, they must be implemented to allow the essential aspects of gameplay, user accessibility and interface. Most of these calculations have been abstracted by the external math library. These calculations include:

1) Calculating the timing offset to draw the arrow based on the users scrolls speed
   a) Calculate the vertical magnitude of the distance from the center of the receptor to the top of the column by subtraction
      i) Transform local space coordinates to view space coordinates
         (1) Multiply local space coordinates by 4x4 matrices and multiply view, model and projection matrices together. Most of these f
            (a) 4x4 Matrix Multiplication:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} =$$

$$\begin{bmatrix} a + 5b + 9c + 13d & 2a + 6b + 10c + 14d & 3a + 7b + 11c + 15d & 4a + 8b + 12c + 16d \\ e + 5f + 9g + 13h & 2e + 6f + 10g + 14h & 3e + 7f + 11g + 15h & 4e + 8f + 12g + 16h \\ i + 5j + 9k + 13l & 2i + 6j + 10k + 14l & 3i + 7j + 11k + 15l & 4i + 8j + 12k + 16l \\ m + 5n + 9o + 13p & 2m + 6n + 10o + 14p & 3m + 7n + 11o + 15p & 4m + 8n + 12o + 16p \end{bmatrix}$$

            (b) Matrix by vector multiplication (for local space coordinates)

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{bmatrix} ax + by + cz + dw \\ ex + fy + gz + hw \\ ix + jy + kz + lw \\ mx + ny + oz + pw \end{bmatrix}$$

   b) Divide it by the users scroll speed to get the time in milliseconds
   c) Display the notes at the top of the screen and let it scroll vertically at a velocity of the users scroll speed. By the time it scrolls to the center of the receptor, the timing of this event would be the same as the time data in the map file data.
2) Adding 3D perspective rotations for GUI elements integrated within perspective projection
   a) Matrix rotation

i) Rotation around the X-axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{bmatrix} x \\ cos\theta \cdot y - sin\theta \cdot z \\ sin\theta \cdot y + cos\theta \cdot z \\ 1 \end{bmatrix}$$

ii) Rotation around the Y-axis:

$$\begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{bmatrix} cos\theta \cdot x + sin\theta \cdot z \\ y \\ -sin\theta \cdot x + cos\theta \cdot z \\ 1 \end{bmatrix}$$

iii) Rotation around the Z-axis:

$$\begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{bmatrix} cos\theta \cdot x - sin\theta \cdot y \\ sin\theta \cdot x + cos\theta \cdot y \\ z \\ 1 \end{bmatrix}$$

3) Adding translations to GUI elements to give them aspects of animation
   a) Translating a vector coordinate in local view space to a new coordinate on the local space
      i) Vector by matrix multiplication (see above)

Real Time Processing

I will require real time processing for the main rendering process in the graphics pipeline. Programming graphics pipeline is the process of turning the raw coordinate and texture data that is currently held in the GPU buffers to the on-screen pixels that is seen on the display monitor. The graphics pipeline must process data and render it onto the window's framebuffer in real-time. Problems with this process happening in real-time will cause graphical artifacts such as tearing and freezing and affect the user's visual perception of gameplay.

The graphics pipeline is essential as it is fundamental for rendering anything on display. The process of the graphics pipeline must be implemented in my source code programmatically (by writing code).In my development many This process is as follows:

1) Vertex Specification/Generation – Specifying the vertex positions and data in local space coordinates on the GPU.
2) Vertex shading – The programmable part of the graphics pipeline in which we can program what to do with the vertex data. What we program will then be executed on each vertex data. Processes like converting to local to view space coordinates occur here.
3) Primitive assembly – Assembling the vertices to form triangles which in turn make up the basis of the shapes seen on screen

4) Rasterization – The process of determining which pixels to be represented based on factors such as depth testing
5) Fragment shading – Another programmable part of the graphics pipeline that determines the final fragment (color) of each pixel to be displayed on screen.

This process must happen all in real-time, typically before each frame is rendered for my adaptation to have its core functionality.

User Requirements

For my adaptation and for most VSRGs in general, the user does not require any foreknowledge or background information requirements on how to play VSRGs. The user can simply engage with the adaptation/VSRG for the first time and be able to experience gameplay. This is due to the VSRGs game mechanics and interfaces being mostly intuitive and indications on the mechanics of gameplay. However, some knowledge and experience on playing VSRGs would be recommended. This can include general knowledge on how to set up key binds for gameplay and general experience with map editing/creation in VSRGs.

The only feasible user requirements that the user needs are a laptop/desktop with a functioning keyboard and mouse and monitor to play the game. However, certain factors such as a keyboard with a high polling rate (1000Hz+) and a mechanical keyboard (preferably with red switches) shall greatly enhance user experience. These factors are not a requirement.

The user will be able to run the adaptation on any form of operating system if the OpenGL drivers are supported within them. Most computer operating systems such as Windows, MacOS and most Linux kernels and graphics card manufacturers integrate the OpenGL drivers into their systems. This means that use will have a wide range of systems to play my adaptation on. Despite my adaptation conforming to multiple operating systems, my primary operating system will be tailored towards Windows.

Essential Features

There are multiple factors that are essential to my adaptation. The factors all include different features and their justification.

Feature:

1. A working user interface with different menu screens for the different game modes
    a. A main menu
        i. Working 3D interface and micro interactions
        ii. Section button into to allow access into map editing/creation

iii. Section into gameplay settings

b. A map selection menu
    i. Displaying map metadata
        1. Song name
        2. Map difficulties
        3. Map BPM
        4. Thumbnail
        5. Map length
        6. Map artist
    ii. Ability to scroll and select each song

Justification:

A working GUI menu is an essential part of gameplay as it is the gateway to a user playing a map and experiencing the main gameplay. If there was no map selection, the user would not be able to decide which maps they can play. This means issues such as playing maps not tailored to their skill level can arise. Therefore, the having the choice of maps of different difficulties is an essential feature

Feature:

2. A working map creation and editing system.
    a. Uploading audio files system
    b. Allowing input of metadata for map file
    c. Previewing of song sections during map editing
        i. Ability to inspect the song in real-time
        ii. Automatically set a preview section that play during the selection of the map
    d. A visualized map editing system consisting of four vertical columns that can be navigated through each section of the map
        i. The deletion and placement of notes
            1. Placement of long notes
                a. Algorithm to determine the release of hold note
                    i. Account to beat timings
                    ii. Ensure the key release is judged
                    iii. Ensure key press is judged
        ii. Beat snap divisors
            1. Divisors that to the 4^{th} beat up to 32^{nd} beat
                a. Zoom in / scale editing preview factor for sections with densely populated notes

e. Map saving system
    i. Save map data to text file
    ii. Auto save
        1. Update map in short intervals

Justification:

As mentioned earlier, for the difficulty of a map to be calculated, the map must first be created. Therefore, a system to create and play maps is an essential feature of the adaptation. If there was no visual map system, then maps would need to be created manually via inputting data into a text file. This would mean a very cumbersome process as the timing and column data will need to be input for each note. This would potentially cause errors and difficulties as large quantities of text are data are prone to mistakes.

Feature:

3. The standard working VSRG gameplay – Once a map is selected to play with its desired difficulty, the gameplay shall commence. This consists of a stationary receptor at the bottom of the screen and notes that will scroll from the top of the screen to the bottom of the screen. All of this will happen typically to the "beat" of the map's music.
    a. Notes to be drawn outside the maximum viewport y-value to ensure notes do not just "appear" and seem like they are scrolling in from out of the screen.
    b. Notes to scroll from outside screen to bottom of receptors
        i. Position to be update every frame
            1. Position updated by distance between the notes starting position and receptor, all divided the user's scroll speed
                a. Scroll speed system
                    i. Accessed through main menu game settings
                    ii. Numerical value between

Justification:
Without the fully adept, core gameplay mechanics of a VSRG, my adaptation will not have succeeded in amending the problem of difficulty calculation. This is because if the main gameplay system does not function fully intended then the difficulty calculated for the map will be a misrepresentation of the user's true ability as the user's true ability is not truly accounted for. Therefore, the functionality of the gameplay must be working to the correct standard.

Feature :

4. An improved difficulty calculation system to accurately and gauge map difficulty.

a. Algorithms to process the map data and determine the note patterns within maps
    i. Pattern detection system
        1. Conditions for the certain patterns
b. A system to open the map files
    i. System must integrate the difficulty calculation in real-time after map saving
        1. Map saving system
c. Different Factors to attribute to difficulty calculation
    i. Speed
    ii. Strain
    iii. Readability
    iv. Playability

Justification:

This is the main feature of my adaptation. As mentioned in research, a new and improved difficulty calculation system will prevent the map difficulty of maps with note patterns prone to being miscalculated from being overestimated/underestimated. I conclude in research that the difficulty should not be calculated based on density alone and other factors such as strain and the maps BPM should be considered into calculation. This way an accurate estimate to map difficulty shall be achieved.

Hardware and Software Requirements

For my adaptation, I will be using the latest version of OpenGL (4.6) as of now. However, most features of OpenGL 4.6 are backwards compatible with version 3.3. Therefore, The minimum hardware and software requirements must be suited to run at least OpenGL 3.3. The minimum hardware and software requirements run my adaptation include:

1. Windows (OpenGL 3.3+)
    a. CPU: Any dual-core based processor e.g. Intel core 2 duo E6850 SLA9U
    b. RAM: 1 GB or more
    c. OS: Windows XP SP2 or later
    d. Graphics Card: Must support OpenGL 3.0; typically requires a card from NVIDIA GeForce 8 series or AMD Radeon HD 2000 series or later.
2. macOS
    a. CPU: Intel-based Mac processor
    b. RAM: 2 GB
    c. OS: macOS 10.6 or later

d. Graphics Card: Supports OpenGL 3.0, typically found in NVIDIA GeForce 8600 or newer, or ATI Radeon HD series.

3. Linux
    a. CPU: Dual-core processor e.g. Intel core 2 duo E6850 SLA9U
    b. RAM: 1 GB
    c. OS: Kernel 2.6 or later
    d. Graphics Card: Supports OpenGL 3.0; NVIDIA GeForce 8 series or AMD Radeon HD 2000 series or later.

Regardless of the operating system and its version, the main requirement for the user is the appropriate drivers installed for your graphics card to utilize OpenGL. This means any system with the essential OpenGL drivers can be able to utilize my adaptation. Proprietary drivers (like NVIDIA or AMD) often provide better support than open-source drivers for this reason.

Limitations

The main limitation to my adaptation is not integrating compatibility for dance machine support. As mentioned in my research earlier, it would not benefit my stakeholders to include dance machine support due the expensive of owning a machine and that most of my audience prefer to engage with VSRGs on their own personal home device instead of commuting to an arcade to access a dance machine. This is also due to the limitation of not having direct access to a local arcade or branch that owns dance machines.

Another fundamental limitation to my adaptation is it only consists of four keys of input during main gameplay. Many VSRGs such as Konami's Beatmania series have more than four keys for input, allowing users to hit notes on 5 or more columns. This is another feature found in osu!mania. osu!mania allows users to create and edit beatmaps that allow different amounts of keys. The key count of the maps ranges from as low as single key of input to as high as ten keys of input. This means that you can play a one key map using a single finger and a ten key map with ten fingers. The reason for this limitation is that the original DDR and StepMania did not allow more than four keys, so to keep the aspects of my adaptation like StepMania, it is necessary to maintain strictly four keys of input only.

Success Criteria

For my adaptation to be successful, there are a set of essential criteria that by the end of development, my adaptation must have met. These criteria include:

- The user interface must be fully working with access to the different game sections via clickable buttons

- The user interface must also allow keyboard navigation with the use of left/right/up/down and enter input keys
- The user interface must contain a main menu screen with the logo of the game and the ability to use input to navigate to gameplay
- The main menu must contain a background that with aspects such as underlap based on different z-depths (layering)
- The main menu must contain a text button in which the user can click to indicate progression in map selection i.e. a play button
- Interaction of the start button must successfully transition you into the map selection screen
- The map selection must contain a list of maps ordered and grouped by artist or difficulty
- The map selection system must allow scrolling through each map via mouse wheel or input left/right keys
- The map selection system must display a thumbnail of the maps song cover/ thumbnail image
- The map selection system must display a moving background as the underlay/background for the map selection
- The map selection system must play a preview of the song in the background of the current selected map
- The map selection should display the maps metadata and song information within a specified GUI section beside the list of maps
- Each map should have the ability to have more than one difficulty
- The map selection system should allow the changing of the maps different difficulties
- The map selection system should allow the pressing of enter to signify the progression into gameplay
- The map should have standard VSRG gameplay - notes that scroll from out the top of the screen to stationary receptors
- The gameplay must accurately start audio in synchronization of beginning gameplay
- The gameplay must allow the player to hit notes once they are in the timing interval (near) the receptors and give an on-time judgement of accuracy without delay
- The gameplay must keep track of the average accuracy of the user and give an end “grade” based on performance
- The gameplay must save the “grade” as a score inside an external file and
- The user must display their average accuracy as a percentage overall along with their “grade”

- The user must allow the exiting of the “grade” screen and can repeat the cycle of selecting a map and entering gameplay
- The user must have the ability to go back to the main menu via back button GUI
- The user must have the ability to enter map editing/creation via a GUI
- The map editing screen must first have a list of all the current maps
- The map editing screen must have a section to upload an audio file and commence map creation
- The map editing must have columns in which the song can be previewed, and notes can be placed
- The map editing system must have a feature to set a thumbnail/song cover for the map
- The map editing system must have a feature to save maps and update their data in real time
- The map editing system must store the map data in an external folder with an external file in a representable and readable data format e.g. a text file.
- The map editing system must allow returning to the menu screen from saving a map
- The adaptation GUI system must have quick interoperability and interchangeability of each section of game i.e. map selection to map creation to gameplay, etc. Without error
- The menu screen must a GUI to allow access to customization of gameplay i.e. a “settings” button
- The settings section must allow changing of scroll speed and changing of key binds
- The settings section must allow the changing of note/receptor size
- The settings section must allow the changing of scroll direction
- The user must successfully be able to exit for the game by pressing the “x” on the window
- The user can also exit the game via the menu through an “exit” GUI.

Design

As mentioned earlier in my research, the design of my adaptation must appeal to both the younger and older generation of stakeholders. Most of my stakeholder audience will be of the younger generation who tend towards the modern aspects. The younger generation also engage in gameplay for longer periods of time, therefore a focus on retaining the functionality and usability of modern VSRGs is a plausible approach to design. However, this must be done whilst retaining the retrospective, arcade-style design of older VSRGs.

This must be achieved by combining the two features uniformly, both in development and design.

Start Menu

For my adaptation's start menu's design, my approach is a dynamic between the old and retrospective arcade dance design theme that is across older VSRGs whilst keeping the modern functionality and useability. The general aesthetic is meant to conform to the iconic designs of older arcade dance games by using features such as arrows

Game Cover (Logo)

To begin, one of the most fundamental aspects of a VSRG is its game cover/logo design. It is often the first impression that users behold and often what they judge the expected outcome of the gameplay to consist of; It forms the initial expectation of what the game is about. For this very reason my game design must include the features of that will suggest that it is aimed at both the older and younger generation alike.

Many older VSRGs designs have a focus on bold, smooth and perspective text. This is prevalent in examples mentioned in research such as ITG and DDR. To stay inclined to the theme of older VSRGs and the general arcade aesthetic found in them, my adaptation's logo design will consist of smooth text with a horizontally tilted perspective. This design choice is intended to mimic the 3D perspective shifts commonly found in older VSRGs designs like DDR.

During the era of many old arcade games and VSRGs the graphical processing power was not advanced as the modern era. This meant the polygon count used to render 3D models and designs was much lower. Therefore, my design consists of a lower sample of curve segments in text rendering. This is to give it the same nostalgic essence found in many older VSRGs and arcade games alike (An example being SuperMario64) that will appeal to the older generation.

As well as the smooth, bold text with perspective, many VSRGs design logos, that the older generation tend towards, such as DDR and Beatmania often have their text stylized with a smooth but heavy outline. This is also prevalent in many modern VSRGs such as Friday Night Funkin', too therefore adding outline to my design will be appeal both to audiences. Because of this, I implemented my design with an initial white outline of the text then an additional black outline and additional perspective shift. This is to maintain the maintain the general 3D text aspects of older VSRGs.

Color Scheme

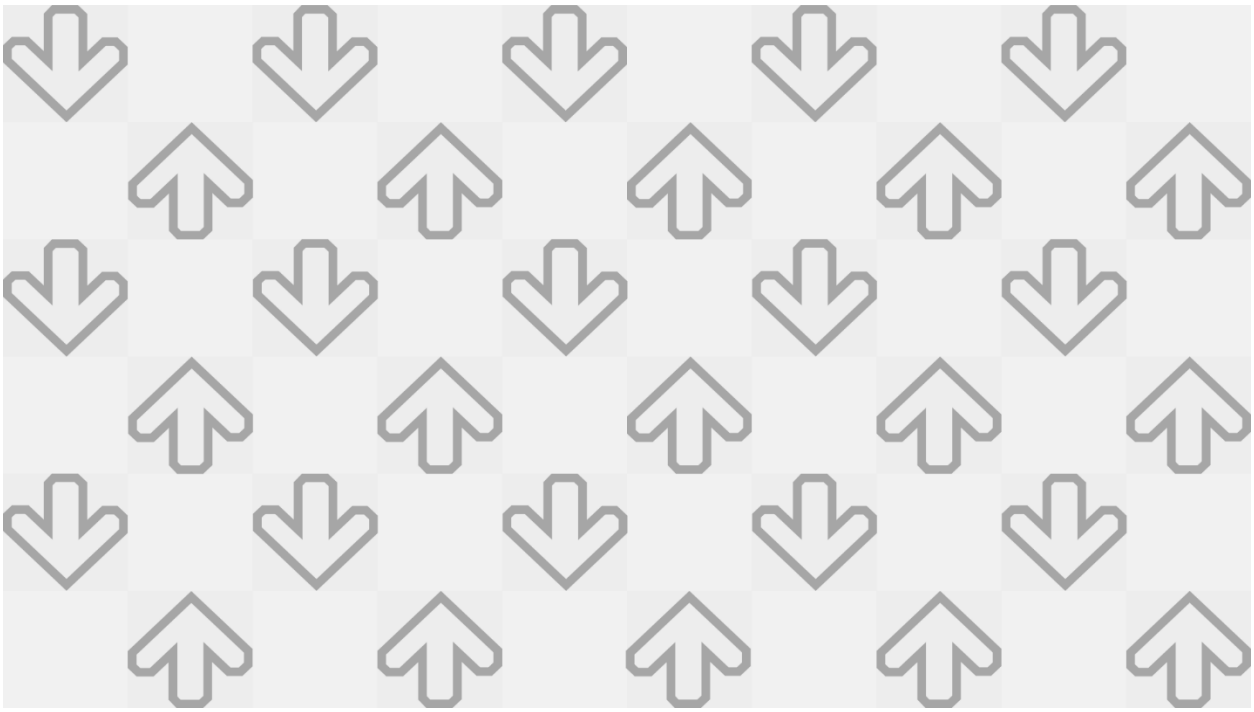
Even though the main features of design are generally appealing to the older audience, I have decided to maintain a relatively simple color scheme of solid black and white colors. The reasoning behind this, is although most of the design will be tailored to the older generation and mimic the aspects of older VSRGs; to maintain some aspect of modernity, the use of minimal colors will attribute both to the retrospective style found in older VSRGs but also to the more modern minimalist style. However, this does not mean the essential design features of my adaptation will still not maintain the pattern of modern useability and functionality but older design, it is more of a balancing act to benefit stakeholders of all age ranges.

Background

One theme that is prevalent in many rhythm games and VSRGs alike is distinctive features that make it clear to the user what the game is about. Earlier in research, the evidence I gained suggested that features like arrows, musical terms, etc. are what make it clear to a user that game is in the genre of a rhythm game. Therefore, for my design, I implemented a background design with the use of an array of alternating hollow arrows to enhance the

arcade dance game style theme. This will benefit the older stakeholder age range and further allude to the design of older VSRGs.

Furthermore, as I commence development of my adaptation, this design will be implemented as an animated moving background to add to the classic dance game aesthetic and generally give the design a fuller and vibrant appeal. My justification for this is the aim provoke the feelings of reminiscence and nostalgia within the older generation as discussed in research earlier.



User Navigation Assist

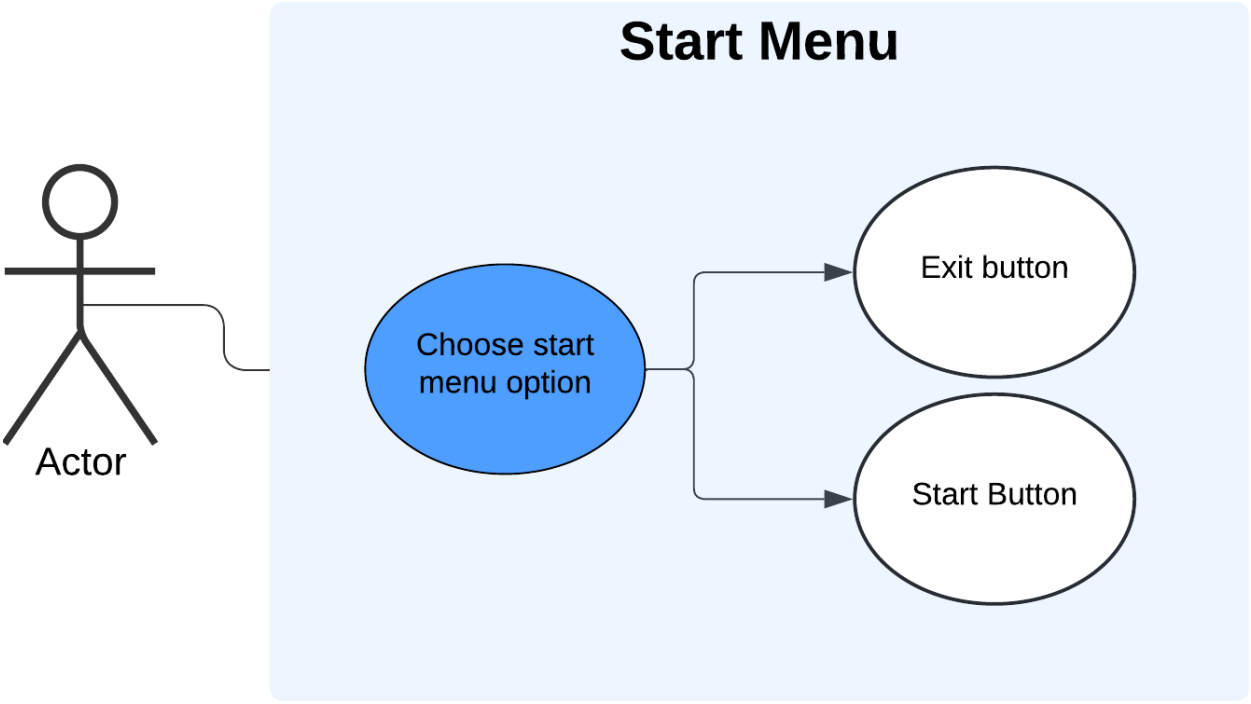
As mentioned, my design will follow the typical design attributes to older VSRGs however to benefit all my stakeholder audience age ranges, implementing modern functionality and useability is a must. Part of the functionality and useability that will benefit the audience of my younger stakeholder is clear and instructed guidance on how to navigate the user interface. Modern VSRGs such Friday Night Funkin' tell the user how to This is through keystroke indication or user input guidance, typically either by directly addressing the keys to be pressed or by showing an image or sort. To add to the modern functionality and useability of my design, I implemented a user navigation assist bar that will be situated at the bottom of the start menu. The bar will give a clear aid on which keyboard inputs are

required to navigate the user interface through buttons to represent keyboard icons. My justification for this design is to generally enhance the intuitiveness of my adaptation and add to the modern functionality and useability.

Start and Exit Buttons

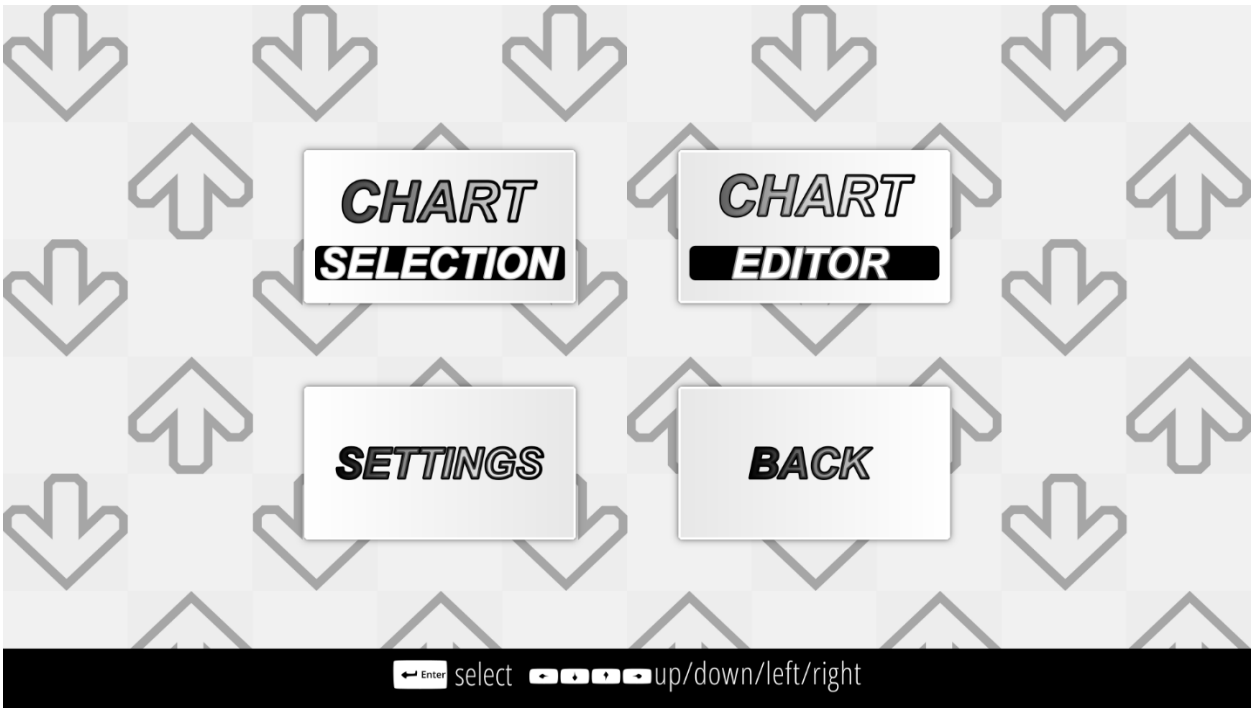
As well as features relating to dance games i.e. arrows, musical terms and the use of bold perspective text, the use of stylized and centered text that a user must interact with via user input to allow progression into main gameplay. This general theme of text that signifies to the user that they must press a key, e.g. enter, to start the game is a common occurrence in many old arcade games and VSRGs alike. Furthermore, many older arcade games tend to leave the text without any form of encasement around the text i.e. the text has no aspects of being a "button" but purely just the text alone. Therefore, for my start menu's user interface, I implemented this style of start and exit buttons to mimic the retrospective menu design of older VSRGs and keep on the pattern of an older design that is appealing to the older audience.

Use Case Diagram



Main Menu

The start menu of my adaptation focused on implementing the desired design aspects. The main menu of my adaptation will focus on navigation and be the core system in which all aspects of the adaptation are accessed. The menu system will allow progression into features such as gameplay, map creation and customization of gameplay settings.



User Interface

The user interface of the menu system is coordinated using left/right/up/down keys. My justification for this is to keep with pattern of the retrospective arcade style feel of older arcade dance game. Furthermore, my intention is to add micro-interaction animations to the interfaces whenever a user is selected on it. This adds to useability and the older VSRG design.

Although the user interface will be accessible through the arrow input keys, I intended to keep the availability using the mouse to navigate a viable option for the user. This is to keep the duality of retrospective design and useability and functionality.

As well as the user interface being accessible, as part of the older design features, I implemented a slight drop shadow around the edges of the interfaces and a linear gradient as the background color. This is justified as older VSRGs like DDR do not tend to use a singular still color but often use gradients alongside their perspective backdrops. Furthermore, the aspect of older design is added through a bold, smooth and outlined text.


CHART
SELECTION


CHART
EDITOR


SETTINGS

Another feature that adds to functionality is the method of going back to the start menu either via a “back” button via mouse input or key input navigation. This adds to the ease of access and further enhances the intuitiveness of my adaptation.

Another aspect of useability and functionality is the user navigation assist that is tailored to the navigation of the main menu. This user navigation bar continues to indicate to the user on how to maneuver around the menu, further keeping the aspects of useability and functionality that is tailored to the modern audience.

Chart (Map) Selection

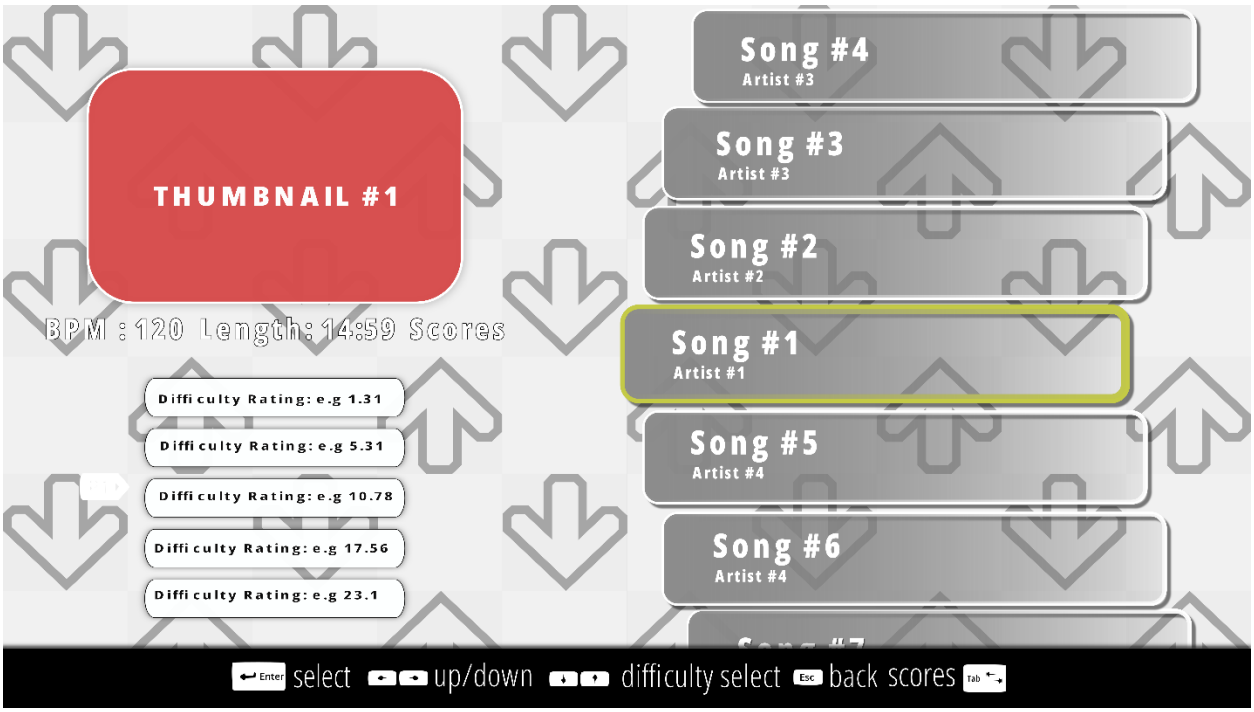
The chart selection menu of my adaptation will allow the user to select a song and commence into the main gameplay. This menu is accessed after the user selects the “chart selection” user interface. The chart selection will consist of a list of all the users’ maps they are able to access on their device. The design remains like most VSRGs and StepMania itself. This design simplifies and heavily abstracts the actual process of loading a chart file into the game. The user does not need to see the internal process happening within the game system. The user only needs to be able to navigate the user interface and access the maps based on the information the map gives. Therefore, simplifying the process to just a few button presses. This design allows the user to focus on aspects of gameplay such as an evaluation of their skill level, whether they can complete the map, which difficulty they should select for the map and if they have the time for the length of duration of the map. All of this will benefit the user’s functionality and allow them to have an immersive experience.

As well as the process of loading maps being abstracted, the chart selection menu design also provides a means of ordering the charts in an organized manner. In reality, the files for

maps are stored in different locations in memory, but the chart selection design visualizes the charts as one continuous list. This will improve the accessibility of charts and speed up the time taken to get into the main gameplay as the user will not need to search for them in memory. This design makes the game smoother and allows fast repetition of entering gameplay, finishing gameplay and then select another chart to enter gameplay. The seamless transition between gameplay and chart selection will benefit the stakeholders of my younger generation as they are more tailored to spending long hours on the game and as mentioned in research. This means they can spend more of their total time practicing playing the game and take less time during navigation.

The chart selection design will also show the most crucial feature to my adaptation, The new and improved difficulty calculations. As mentioned in research, the calculation must be of a quantitative measure and give an accurate measure of the chart difficulty. The actual difficulty of the map is calculated in real-time during and after the map creation process and is displayed in both the editor and selection menu. This feature is a major part of my success criteria and therefore must be a fundamental aspect of my design. As well that, the ability to change between different map difficulties and view scores must be a feature of my design as the difficulty of maps and the scores achieved play a large part in gameplay as they are the main indicators of progression in skill level.

The final feature of my chart selection design is the displaying of a thumbnail/background for the chart. This feature will help in differentiation of charts for when charts may have the same name. This feature along with an audio preview of the song whilst it is playing will further improve the user experience and meet the standards of my success criteria.



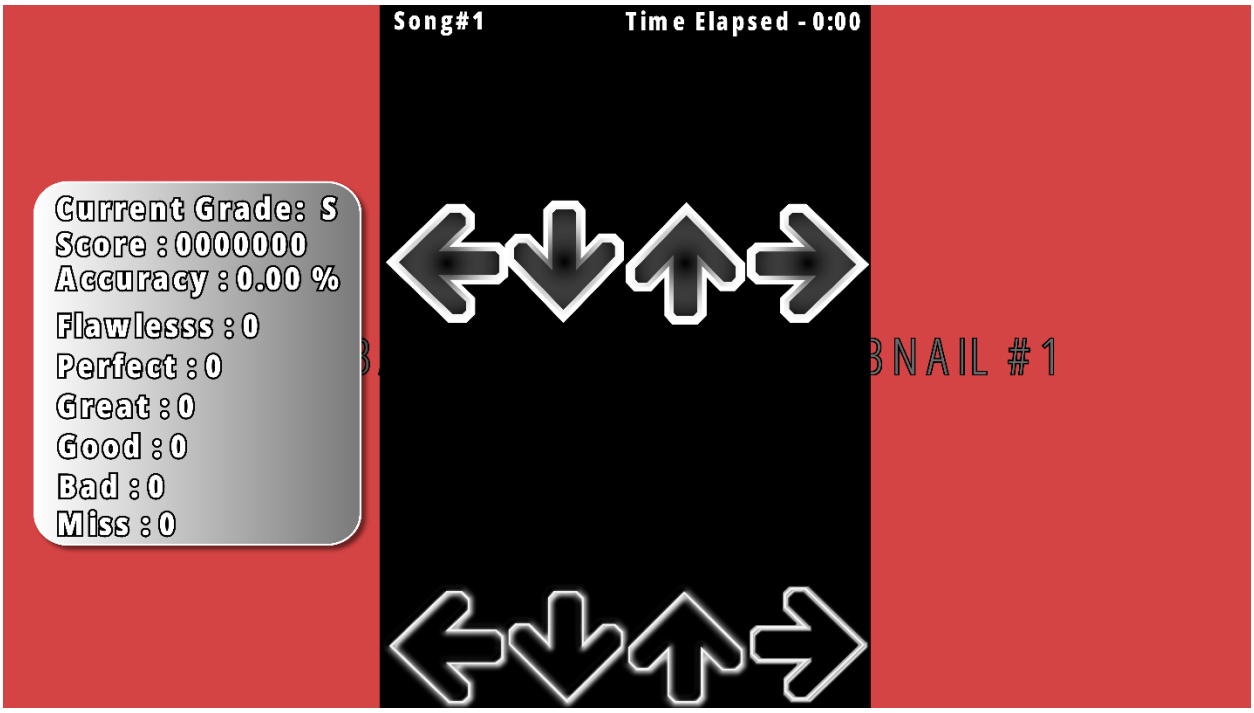
The user navigation assist bar is still present within the design to keep the aspect of usability and functionality. In this section of the game, the bar contains more input icons to tailor to the increased controls to navigate through the chart selection.

Main Gameplay

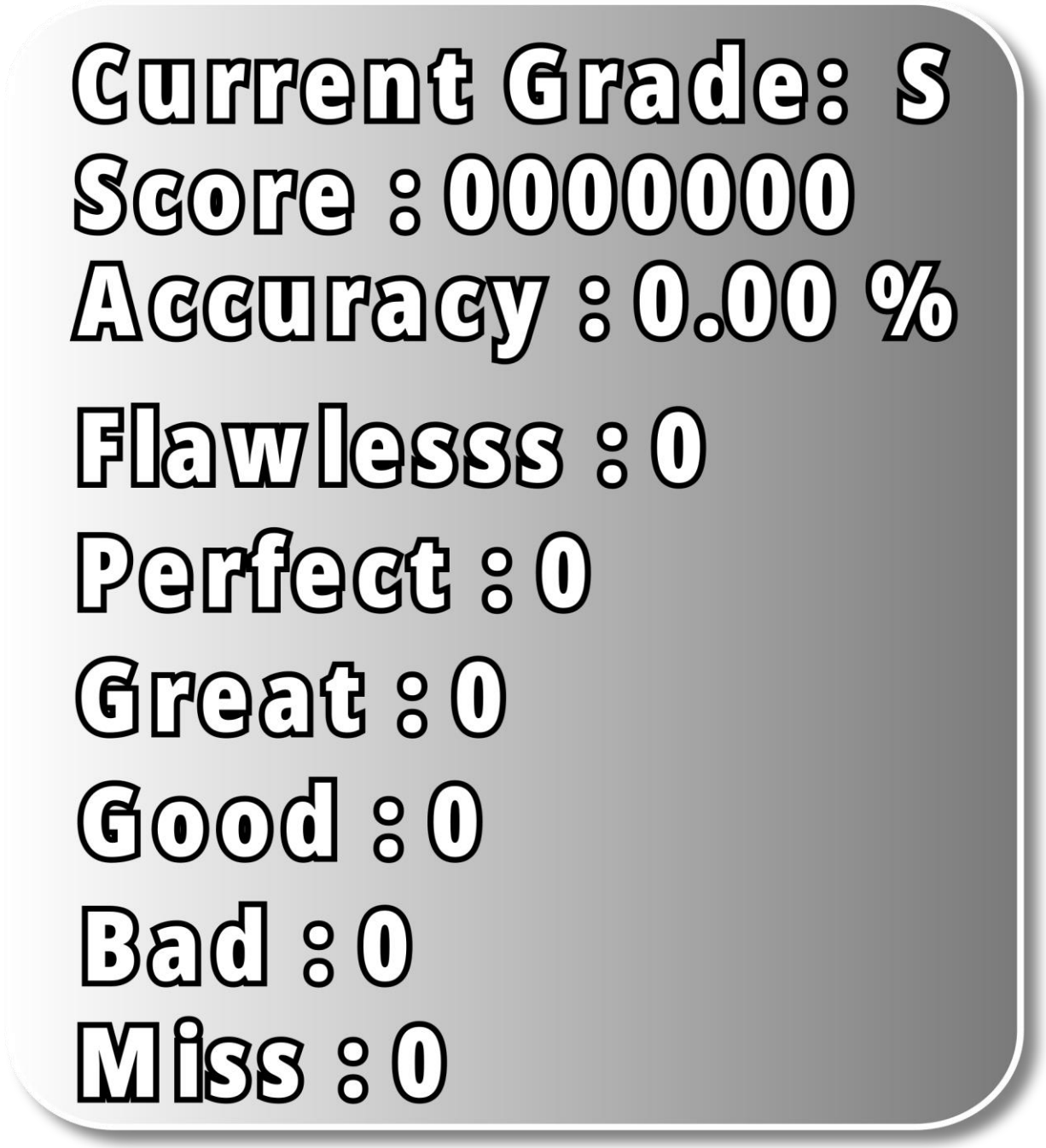
The main gameplay will consist of the standard VSRGs gameplay. The user will be able to accordingly respond to notes that are scrolling vertically and yield input based on the visual information. As well as the visual information of how close the notes are to the receptors, the user will also have audio information. There must be audio playing in the background of the main gameplay which will aid in timing the notes and set the basis of the 'rhythm' aspect of the adaptation.

In the background of my adaptation's gameplay design, there shall also be a timing class that will process the input timings in real time and give an accurate response on timing judgement. The timing class will form the basis for setting up the user's scoring, thus in turn forming the basis of the grade and accuracy system.

As well as the gameplay and timing, in the actual background of the columns in which the notes scroll, there will be a background image that the user can set in map creation. This



The gameplay statistics will be what is used to indicate the user's performance and the general skill level of the user on the map. The statistics will show the quantity of hit judgements they have received and the average hit accuracy throughout the map. A higher hit accuracy will contribute to a higher accuracy which will increase the score. The displaying of real time gameplay statistics will be beneficial to the user during gameplay. This can be during areas of which there are not many notes in which the user can take a glance. It will also benefit them as it will allow them to monitor their performance progress between their current gameplay and past gameplay score if they are replaying the map. The ability to monitor and track progress statistics in real time adds to the functionality and useability of the adaptation that is appealing to the modern audience of my stakeholders.



Grade Screen

Once the user has finished the map, the user will be “graded” based on their performance. The user’s grade will be accompanied by the final iteration of the gameplay statistics. This design forms the more of the functionality and useability of my adaptation. This is because scoring systems can be used to keep track of how the users’ scores change over time and

give an indication of improvement in gameplay skill. Being able to monitor and track improvement can give the user a sense of motivation. It can also give the user an indication of what needs improvement. As mentioned earlier, maps can have different sequences of note patterns. If the user's score is consistently below expectation for maps that are tailored to a specific note pattern, then this design can help indicate which maps the user should focus on in gameplay.

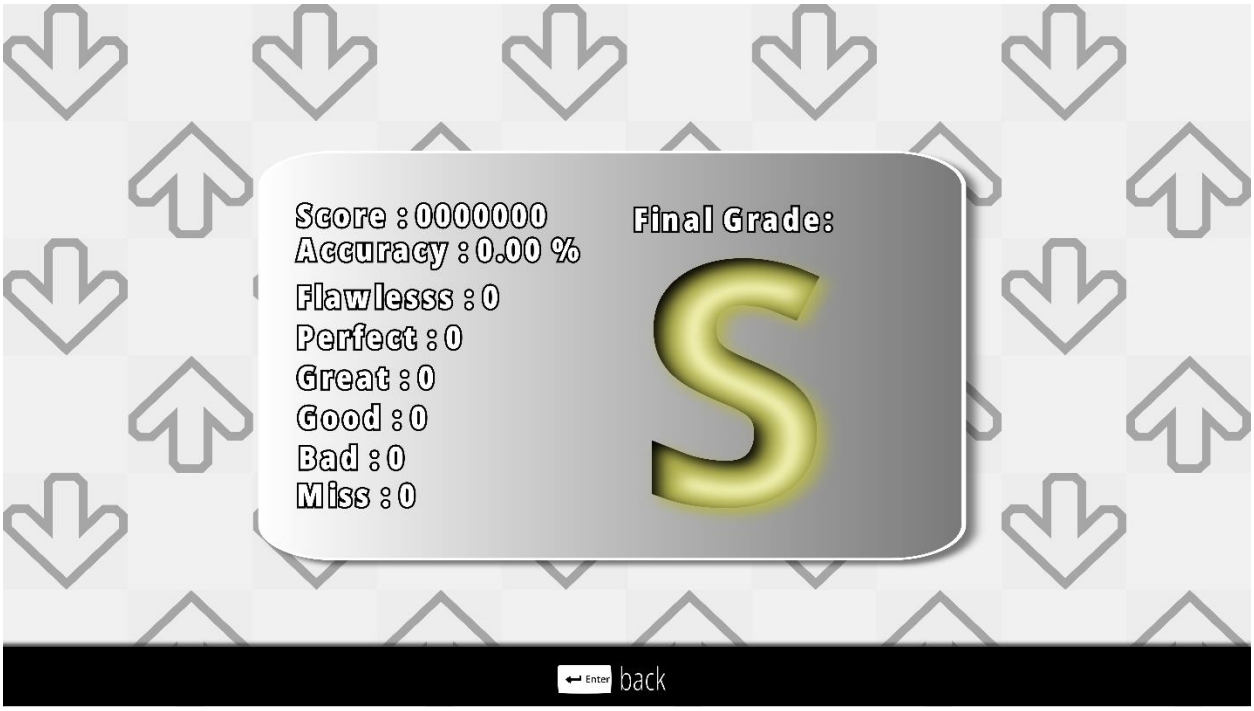
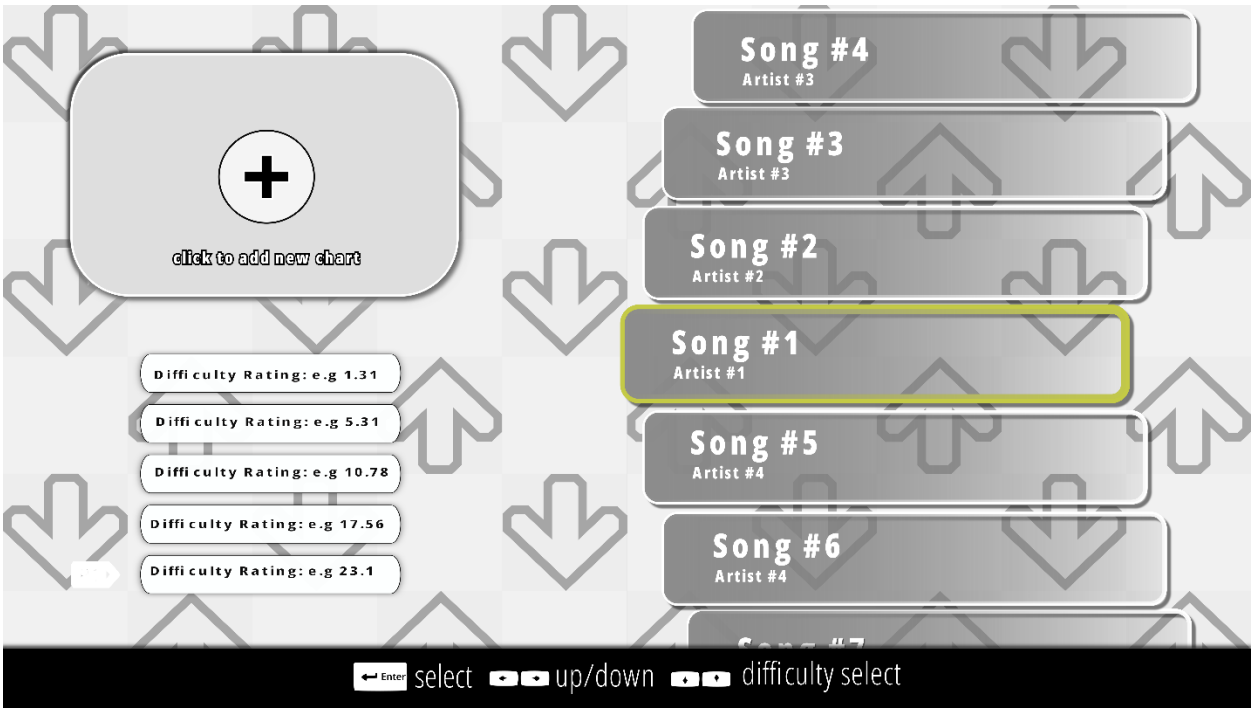
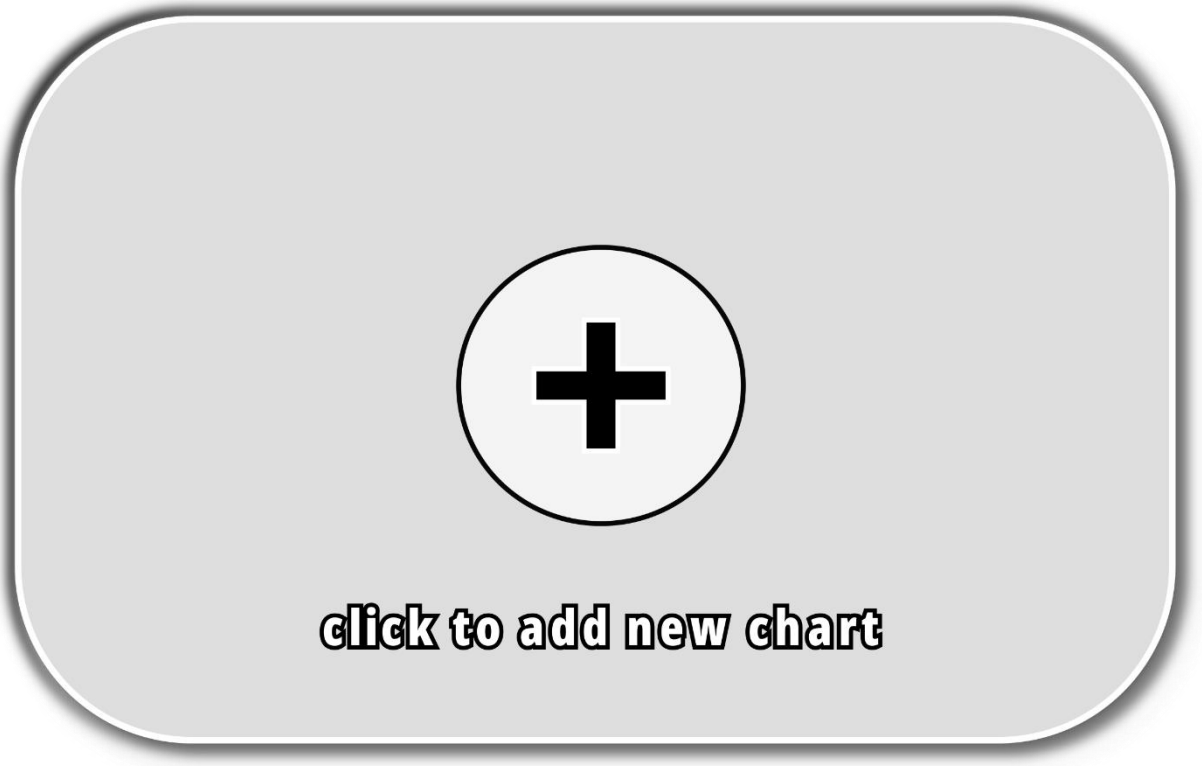


Chart Editor Selection

The chart editor is the second choice within the main menu and will mostly resemble aspects of the map selection menu. However, the difference between the chart selection editor and the map selection is the ability to upload audio files to initiate the creation of a new map. As mentioned in research earlier, this is an essential aspect of the adaptation's functionality as it is the conduit into which map the difficulties calculation system is exercised.



The user will be able to initiate the process of creating a new map via the “create new chart” button via mouse input. The reason for limiting it to mouse input only is to limit accidental keyboard mis-input leading to the creation of unwanted new maps.



Once the user has clicked on the chart button, the user will be able to enter the song descriptions and choose the accompanying background and thumbnail images. This will form the basis for previewing the song in the map selection editor.

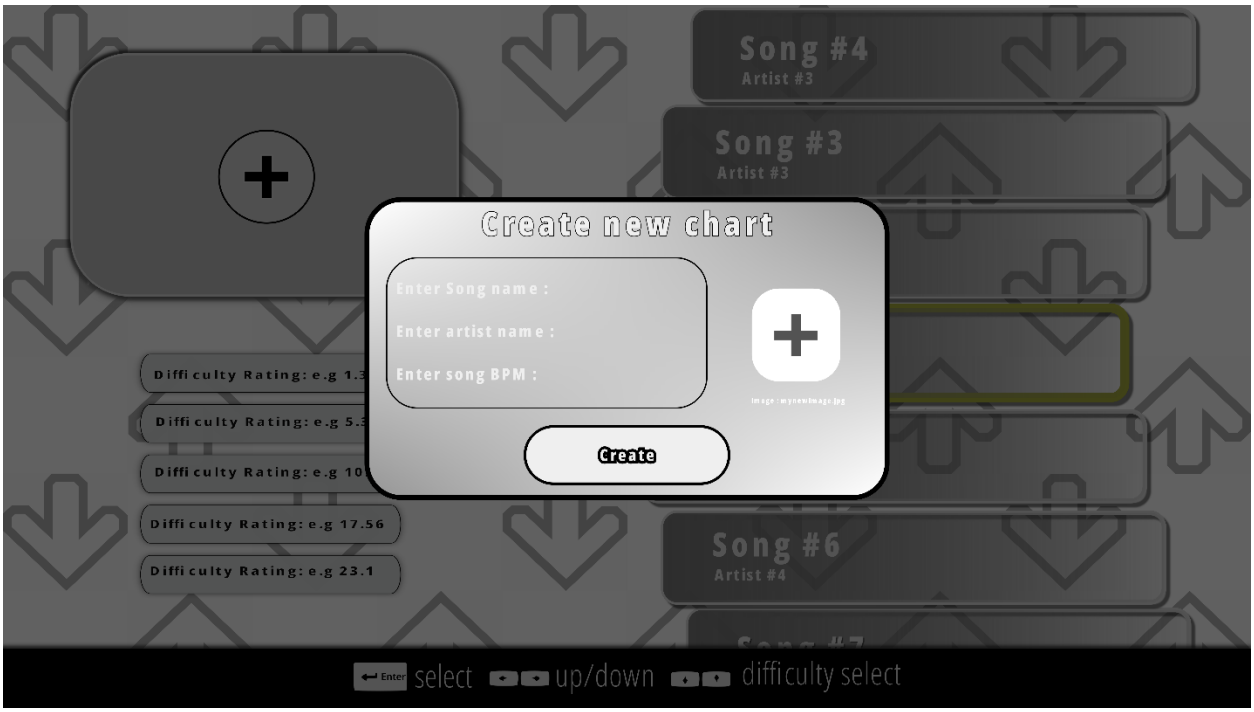


Chart Editor

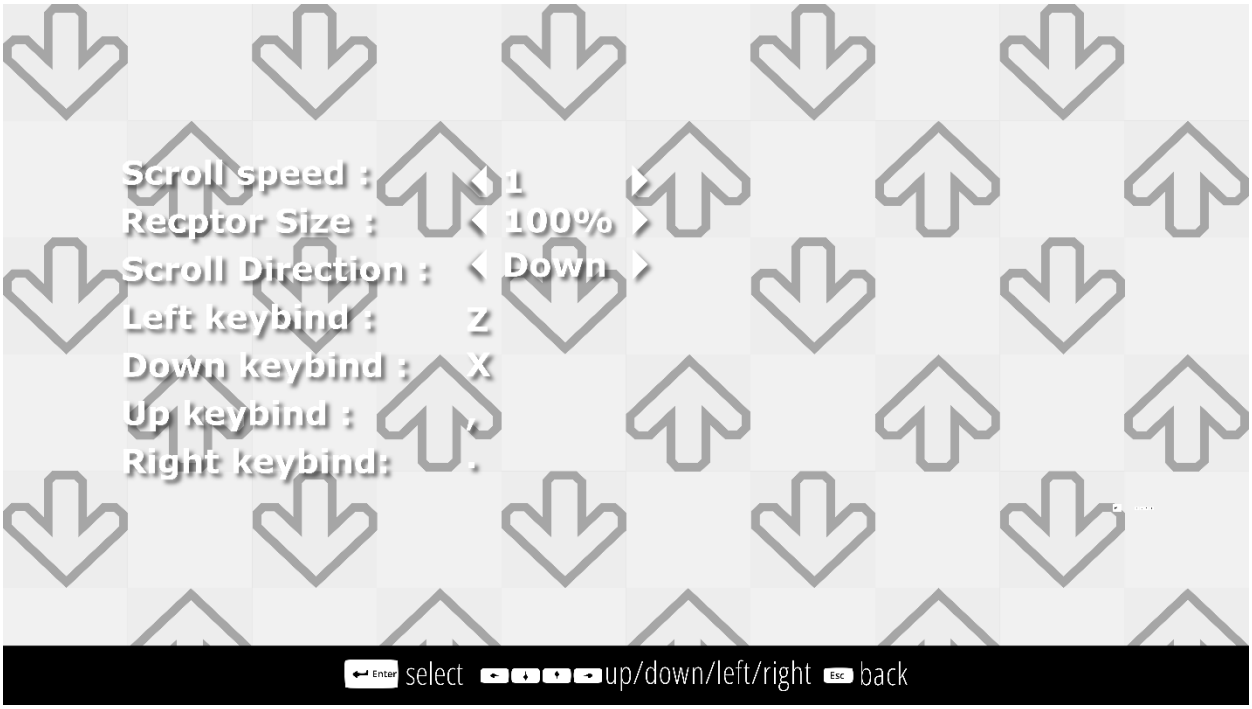
The chart editor is one of the most crucial parts of gameplay as it will be where the main aspect calculation of difficulty will take place. The chart editor will consist of a range of statistics that will be updated in real time as the user is editing a map. The statistics will include the different factors that contribute to difficulty calculation and the terminology for the different note patterns within the map. As well as this, the chart editor will provide the abstracted and visualized process of placing down notes. As mentioned earlier in research, the design of this abstracts all the background process of reading and writing to a chart file down to a few mouse inputs. This further adds to functionality the adaptation.

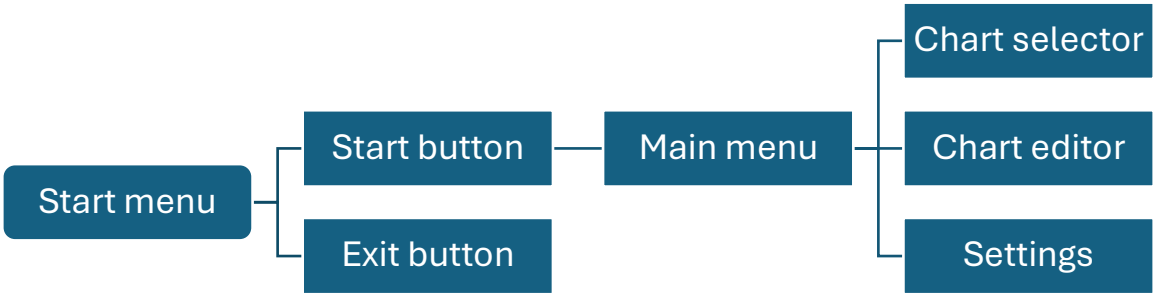
The process of determining the note statistics such as patterns and counts will be abstracted from the user but will be happening in the background in real time via algorithms to determine them.

As well as displaying the char statistics, the user will be able to change the beat snapping to allow for more of a precise representation of flow of the music on the notes. The user will also be able to scroll through the length of the map via the scroll bar and preview the notes that are within that section of the chart.

Settings

The final section of the game that use will be able to access is the settings menu. Including settings will add to the useability and functionality of the adaptation as it will allow customizability of gameplay. This will be useful for people who have a preference in the way they engage in gameplay. This is because there are a variety of different preferences that a user may use to achieve the best possible gameplay, with each preference being relative to the user. This will appeal to the modern audience who are more prone to longer hours of gameplay as it will add extensibility for changing styles as they progress. Furthermore, older VSRGs lacked this feature of customizability, therefore adding a settings design will better suit my adaptation for all age ranges of my stakeholders.


Scroll speed : 1
Recptor Size : 100%
Scroll Direction : Down
Left keybind : Z
Down keybind : X
Up keybind : ,
Right keybind: .
Enter select up/down/left/right Esc back


Start menu
Start button
Exit button
Main menu
Chart selector
Chart editor
Settings