Not Into These Graphics

A Computer Graphics Project Based on Not In The Groove

Computer Graphics
CS 4361.001

March 6th, 2022

Project Description:

NotITG, also known as Not In The Groove [1], is a rhythm game based on the game In The Groove. It was designed to allow creators to make custom playable song files (known as step charts) to be heavily modified and utilize custom visuals and animation effects (creating what is known as a mod file). This purpose of this project is to use the knowledge gained from our computer graphics course and implement it through the NotITG engine to create a custom mod file that can be played and contains unique animations, effects, and shaders to enhance the experience.

Not In The Groove (NotITG) is a rhythm game based on the game In The Groove (ITG) where players try to step on a pad of four directional arrow panels as arrows scroll from the edge of the screen to a row of receptors. The players must step on the corresponding panel when an arrow meets a receptor, and these steps are timed to the music.

Tools and Software Used:

This project utilized the open source NotITG Engine, which is a fork of OpenITG, which itself is a fork of the open source game Stepmania. The NotITG Engine allows for defining and adding textures and actors into the scene through the use of XML. The engine also allows for the movement of elements in 3D space, such as rotations, scaling, and transformation, as well as creating and using visual effects through the use of lua. An add-on to NotITG chart files, called The Mirin Template [2], was also used. The Mirin Template allows for easier creation and use of visual effects, including a simplified method of creating complex custom effects, a simplified method to syncing effects to specific beats of the music, and a simplified system to smoothly transition between different intensities of effects over time (ease functions). The NotITG Engine also supports the use of GLSL shaders programmed in C++ which were used in this project. This project also utilized multiple programs that are not directly utilized by the NotITG Engine. The open source GNU Image Manipulation Program (GIMP) [3] was used to create a few images used in the project. REAPER [4] was used to edit the song audio, including cutting and syncing the song to the beat. ArrowVortex [5] was used to create the initial step chart file and to resync the song further, as well as set a few fields of metadata in the step chart file.

Design and Implementation Details:

It is recommended that the reader views the project demo before continuing this section, either by running the demo provided with this report, or by viewing a recording of the demo which can be found through the following link:
https://youtu.be/Q8HPMLrn-rg

Before starting work on the creation of the actual mod file, a base step chart would need to be created. To do this, I needed to select the song in which I wanted to base the whole project around. I decided I wanted to choose a song I would legally be allowed to use as getting sued in the process of creating this project was not on my to do list. After a few hours of searching, I came across Max Brhom's *Humanity* [6], and felt that it would be a good choice. The song made many different auditory elements that I felt I could create visual effects for, while still being rather simple which would make it easier to chart. After downloading the song, I opened it in REAPER to cut out around half the song (as a shorter song would be easier to work with as I would have to chart less and create less effects). I also made sure that the first beat of the song started at the very beginning of the song, trimming any silence that was at the beginning. This was done to help sync the song up with the notes I would then write. Using ArrowVortex, I created a step chart file for the song. I used the built-in auto-sync tool to further refine the sync of the song, and added the required metadata elements to the file. I then placed a note down on every major rhythm in the song to get a working chart that would allow me to get a feel for the effects I would write, and would come back later to re-chart it with a proper step chart.

To turn a step chart into a mod file, you must point the engine to a mod file in the step chart, and from there carefully wave xml and embedded lua to craft a mod file. This, however, can be very confusing and complicated for beginner and veteran modders alike, and thus many different add-ons and templates have been created to take care of the overhead and provide useful ease-of-use features for modding. I used The Mirin Template in my file, which took care of this overhead and separated the mod file into two parts: an xml file that defined the different visual elements of the mod file (layout.xml) and a lua file that defined the different effects and other custom scripts used in the mod file (mods.lua). Both of these can be found in the lua folder of the attached demo project folder. From here, I still had to learn how to use actors, proxies, textures, shaders, and other visual elements as well as how to create and apply effects using The Mirin Template. For this, while I did rely heavily on provided documentation [7-10], I must also give special thanks to the person under the alias "SheepyPeka" who provided a lot of explanation and insight when the documentation fell short [11].

The mod file begins with a long section of the song with long, held-out bass was played, to which no notes were charted. This slow build up, while long and rather empty, I felt was too important to the song as a whole to cut. However, sitting through over twenty seconds of nothing is rather uninteresting, and thus the first thing I did was hide the receptors, quickly fading them into different alpha levels as the bass would start up and slowly increasing in volume each time, and slowly faded them out as the bass did the same. After playing with a few methods to hide the receptors, the best method I could find was to use the “dark” mod built into NotITG, which would change how much transparency the receptors had on a scale of 0 to 100. The receptors were set to a level 100 “dark” at the start of the song, and would fade to decreasing increments of 10 (90, 80, 70, etc) each time the bass would come in, until they hit zero where they would remain. Eases, which were added by The Mirin Template, were used to make the fading effect. The ease works in three parts. First, the beat that the effect should start needs to be defined, as well as the number of beats the ease should last. Next, the ease function would need to be defined. The ease function is a shaping function that specifies the manner in which the transition should take place. For example, the inExpo ease function would cause the transition from 0 to 100 in an exponential manner, transitioning through the values slowly at first but speeding up over the time period in an exponential fashion until it reaches 100 at the end of the time period. The Mirin Template provides many different predefined ease functions, and they can all be found in the official documentation. Finally, the desired final value of the ease as well as the desired effect needs to be defined. This effect lasted the entire beginning section of the song.

As the more rhythmatic sections of the song start coming in, so too do the notes. During this beginning section, a few snare hits in the song are emphasized, and so I decided to create a bracket note on these snare hits coupled with an effect. The effect I settled on was a column swap effect, in which the different columns of arrows in the playfield would swap with each other, creating unusual configurations of the standard right/down/up/left arrow order. There are two predefined mods in NotITG that change the column order: “invert” which swaps adjacent arrow positions (left swaps with down, up swaps with right) and “flip” which reverses the arrow positions (left swaps with right, up swaps with down). By combining different percentages of “flip” and “invert”, different configurations of arrow positions can be configured. Throughout the snare hits, I alternated between swapping the two left columns combined with swapping the two right columns and swapping the two inside columns with each other. I used a 3rd party tool created for this exact purpose (though it does not have a name) [12] to calculate the exact values of “flip” and “invert” that I would need for each configuration of columns, and added the eases to the file. The column swaps ended up being more challenging than I intended for the player, but not too challenging that I felt the need to

change it. I instead compensated for this added difficulty when creating the final step chart and created a much more basic and repetitive pattern of notes during some of the more unique and hard to read sections of the column swaps that also (though it may not be obvious) flow quite well into each other in a way that makes it easier to feel your way through even if it seems quite difficult to read at first.

Figure 1: Centered Playfields with Low Opacity Receptors and Column Swaps

The column swaps happened to end with the columns being in the complete reverse order from what they are by default, which gave me the idea to have the playfields start spinning slowly and increase in speed as the final one and a half measures of the beginning section played out. This would make it so that the arrows would be facing the correct way but be in reverse order when the front of the playfields were facing the camera, but when facing away would be in the normal order but facing the opposite way. However, the arrow model itself does not have a backside to it, and thus would be invisible if I were to spin it. To overcome this, I decided to use player 2's playfield as the backside of the arrows and use player 1's playfield as the frontside. This would require that I center the playfields, however, which ended up being more challenging than I anticipated. Centering the playfields themselves was quite easy, as NotITG has a built-in method to position elements anywhere on the screen by pixel position (starting from the top-left corner), as well as a way to get the screen height and width. On top of this, The Mirin Template adds an easy way to get the center position of the screen (for more readable code). However, all rotations are done around the point (0,0,0), and thus trying to rotate the playfields after moving them to the center just causes them to spin around the left-side of the screen instead of in-place at the center. I was able to fix this by moving the playfields to the top left corner of the screen and then using a proxy to move the playfields back to the center of the screen. A proxy element essentially just copies any element(s) from the xml file, allowing you to apply effects to an element(s) that may already have carefully crafted effects being applied to it without interfering with said effect. By using a proxy, the proxy can be moved to the center of the screen while the original playfields remain centered to a relative (0,0), allowing them to be spun in-place. I used a predefined function to rotate the playfields with player 2's rotation value being offset by 180 degrees, and defined my own mod to allow me to change the amount of rotation on the playfields. I used an ease function with this newly defined mod to spin the playfields with increasing speed. In addition to this, player 2's columns needed to be flipped from what player 1 had to ensure they were in the correct position, and the left and right arrows needed to be reversed so they appeared to be facing the correct way for the backside of the arrows. To accomplish this, I re-defined the column swaps for player 2 to match up correctly and used a predefined mod to set the z-rotation on the left and right arrows of player 2 to be offset by 180 degrees. With this, the spin effect was completed. To ensure that the chart was still readable throughout this spin effect, I made sure to root the right foot notes to 8th note divisions such that the right foot would only be used to step on blue arrows during this section. I also used a 16th note roll once the rhythm started to get quicker, and made sure to ease the spin in such a way that it would only get fast once the roll had already started. To my surprise, this was extremely effective in improving the chart's readability during this section.

To finish out the beginning section, I decided to add “warning graphics” that would flash in time with the synth used in the beginning section of the song. To make these, I created a column of five strips of even height and spacing and skewed them, wrapping anything cut off at the top or bottom around to the other side so that, if I wanted, I could use two copies of the image offset vertically from each other to mimic an infinitely scrolling column of strips (this scrolling effect was fully programmed and still exists in the mod file, though unused). I also created a triangle graphic with an exclamation point inside of it. I used GIMP to create these graphics, as I was the most experienced in GIMP as a photo editing tool. I placed two to the strip columns in the scene, one at each edge of the screen, and placed the triangle in the center. I then put these graphics in a single actor frame, which would allow me to manipulate them all as one element., and put this actor frame between the background and the playfields. I then used the predefined function “diffuse” to change how much of the red, green, blue, and alpha channels of the images were shown. Using this, I changed the images to display as red, and defined a mod called “warningalpha” to change the alpha to whatever level was specified between 0 and 100. This allowed me to use the diffuse effect through my custom defined mod in an ease, which I used to smoothly flash the warning graphics at increasing opacities as the sync came in during the song.

Figure 2: Rotated Playfield with Semi-Transparent “Warning Graphics”

After the beginning section ending with the spin, there are a few measures where the music stops and the vocals “It’s all about humanity” can be heard, before going into the final section of the song called “the drop.” This section begins with four short snare hits with one more drawn out snare before the song picks up again with the bass and melody. Since these snare hits were used more as a transition into the drop, I decided to not place any notes during these hits and emphasize the very next bass note with a jump hold. However, I still wanted to add an effect to these snare hits, and after a while decided to use the first four short snare hits to make each note column appear one at a time, though in the incorrect column order, using the final snare hit to ease them into the correct place. To do this, I set each column size to be “invisible” by making each column so small that it was unseeable. This was done by setting each column with the predefined mod of “mini” with a value of 200 as soon as the roll from the previous section was finished. This hid both the receptors and the upcoming notes, only displaying them when I eased them back to 0 “mini” on the snare hits. Then I needed to perform a column swap during the time that the arrows were hidden. To do this, I once again used the “invert” mod to swap the playfield column. However, since I didn’t want the players to realize immediately that the columns were swapped, I also counter-rotated each column by 90 degrees on the z-axis through the use of the predefined “confusionzoffset” mod. I also reset player 2’s playfield to be facing the same way and be inline with the effects on player 1’s playfield at this time, since I no longer needed the playfields to have a backside. During the final snare hit, I reset the z-rotation and column swaps back to default, revealing the intended note pattern just before the jump. This last-second column swap is generally not considered to be a fair effect, as it completely changes what notes the players need to step on at the last second. However, when done right before a jump, it is known as a “haunted house”, which is a fairly common gimmick in modded files. In general, haunted houses are typically only done on jumps right after a section without any notes, and are typically marked with mines on the other column at the same time as the jump to indicate to the player that it is a haunted house and to hit the opposite of what the jump appears to be. However, I forgot to add the mines in the project demo file.

As we progress through the drop, the melody does a series of 16th notes being occasionally interrupted by longer synth notes in repetitive sections that are two measures in length for a total of 8 sections, with a small interlude after the first four sections and increasing in musical intensity for the last four sections. For this, I decided to split the drop into two different parts according to the two different sets of four sections. In terms of charting, since the sections were very repetitive, I did my best to create unique but interesting and even patterns for the 16th notes, using holds during the synths and jumps during the snare hits. When it came time to chart the last four sections, I decided it was best to just mirror the notes from the first four sections,

simply swapping the left and right arrows. This would emphasize the repetitive nature of these sections while also helping to ensure the chart was balanced.

During the first half of the drop, inspired by effects I had seen in other mod files, I decided to decouple the background graphic from the playfield and move and rotate each separately in 3D space. I decided that I would alternate moving the background from left to right, and do the opposite with the playfield at the beginning of each section, slightly rotating each along the y-axis to mirror and emphasize this movement. This created a really satisfying effect that made it feel as though the playfield was floating on it's own away from the background, and by keeping the background and playfield on opposite sides of the screen, the screen still felt balanced overall and not like there was too much going on on one side or the other. Again, using predefined functions I was able to rotate the background and playfields, and created my own custom mod for the background to allow me to rotate it along the y-axis and a new custom mod for the playfield to allow me to rotate it along the x-axis. I also defined mods for both the background and playfields that allowed me to move them around the screen along the x-axis relative to the edges of the screen, such that a value of 100 would move it to the right edge and a value of -100 would move it to the left edge (creating additional mods to move the background along the y-axis and z-axis, with the y-axis also being relative to the edges of the screen). Once this was done, I ran into a problem with rotating the background. As it turns out, NotITG by default uses parallel projection on all elements except the playfield, and thus rotating the playfields just made them look squished. Changing the perspective to a project perspective was fairly easy, however, as it just required defining the fov of the background actor in the xml file. I also decided to move the background back in the z-direction away from the camera at each section, as I felt like it looked and felt more alive that way. This was done using the custom mod I mentioned earlier. I also ran into a small problem with this, as the default far clipping plane ended up being to close, such that when I moved the background to -1000 z for the final section, it ended up being clipped by the far clipping plane. To fix this, I had to overwrite the far clipping plane via the built in SetFarDist function provided by NotITG. At the end of the four sections as the small interlude in the drop began, I eased the background and playfields back into their original place. During each section, there are two synth notes of equal duration with a small amount of 16th notes between them. Again, inspired by another mod file I had seen, I decided to rotate the playfield along the x-axis, tilting it forwards during the first synth and resetting it back to normal during the second. Since this section wasn't any more difficult to read, I didn't make any special considerations when charting it. Instead, I simply made sure to keep rotating the playfield more drastically to slightly increase the challenge as the sections went on, making sure not to over-rotate the playfield and make it too difficult to read.

During the first four sections, I also decided to add some “bounciness” to the arrows themselves during the 16th note patterns, once again inspired by another mod file I had seen previously. I played around with many different mods but never quite found a combination I was truly happy with. In the end, I settled on using a combination of three predefined mods: “tipsy” which would move the columns slightly up and down relatively according to a wave function, “drunk” which would move the relative x position of the arrows of a column left and right according to a wave function, and “pulse”, which would change the relative size of the arrows according to a wave function. I played around with these mods and their different offsets until I found a few combinations I was happy enough with. On each note, I would quickly ease the effects to a given value, then quickly ease them to the opposite (negative) value on the next note, and so on for the duration of the 16th note section. Every set of four notes during the 16th note sections were the same, so for each set of four notes I used a different combination of values for the effects to make it feel more interesting than the song really was. For the last set of 16th notes in each segment, I also added the effect of having player 2’s mod values being the opposite (negative) of player 1’s, creating an interesting “seeing double” effect. The goal with these effects was to add a bit more flare and make the notes feel a bit more alive during a rather repetitive section of the song without overdoing any effect and making it difficult to read.

Figure 3: Rotated and Moved Playfield with Rotated and Moved Background Image

During the interlude between the two parts of the drop, I decided to make the background look like it was jumping and dropping out of frame. This was to make sure to have something to entertain the players during this small section of no notes, as well as to clear the background away for the effect I wanted to do in the next section. This was as simple as easing the background up using the movement mod I created earlier, and then back down past the edge of the screen. Getting these two eases to look good together and like there was a consistent pull of gravity on the background graphic turned out to be much harder than I expected, and I ended up just playing around with the eases until I got a bounce effect that I felt looked nice.

For the final effect, I really wanted to do some sort of camera movement through a 3D matrix of playfields, as this was an idea I had had even before taking this computer graphics class in the event that I ever did start making mod files. I also felt that this movement really fit the final section of the song, so I went for it. The idea of the effect was to have the camera slowly move through a series of playfields evenly spaced out in the x, y, and z directions of a 3D space. However, to replicate this effect, I would need to have between 100 and 250 copies of the playfield active for the camera to pick up as it moves around, which is not computationally friendly to a game trying to run at 60 frames per second. In addition, the camera cannot move in the NotITG engine, and the effect would have to be done by instead moving the playfields around, which would make the computational problem significantly worse. However, I still really wanted to create this effect, so I looked into shaders. After a week and a half of learning how to write fragmentation shaders from scratch, I was able to cobble together a shader that would produce the desired effect. To fake the 3D movement of the playfields, I first divided the screen into a 10 by 10 grid of segments. I then centered the middle most segment to the center of the screen, and offset the x and y position of each pixel based on two variables passed in, wrapping around the edges of the screen. By changing these offsets, the grid could be effectively scrolled. Afterwards, I mapped the texture that the shader was being applied to to the grid, such that each section had a small scaled down version of the texture. Due to how NotITG works, shaders can only be applied to textures. The best way I found to make a playfield into a texture was to use an Actor Frame Texture to take the elements behind it and apply it to a quad as a texture. However, in doing this, it also picks up the background, which was not transparent. I added a chromakey effect to the shader at the end to key out anything close to the hex value #ff00ff color (which is magenta), and changed the background to be completely magenta. (I also added a way to toggle the chromakey to key out the color green, but this wasn't used). This allowed the background to be keyed out and turned transparent. Finally, I added a way to change the size of the grid via a passed in variable (though, again, this wasn't really used as the final grid size I used was still 10 by 10). NotITG also supplies a way to pass in and even change variables in shaders,

which allowed me to ease the x and y offset variables to create a smooth scrolling effect. I created five quads that were 10 times bigger than the screen and applied the playfield actor frame texture and scrolling shader to each. I position one at the center of the screen with no offset to the z position so that it would look like a normal playfield until the scrolling began, and positioned the other four quads behind the camera with equal spacing between them. Finally, I create a mod that would allow me to change the z positions of each on of the quads, and another set of mods that would allow me to change the scroll speed of each shader. For the final four sections of the song, I eased the scroll mods to move the grids down and to the right at differing speeds and with different ease function so that it felt more dynamic than just a simple scroll movement. I also eased each of the quads back away from the camera so they would pass in front of it and move backwards until all five were in view. Since they all stayed the same distance from each other, and each had the same scroll shader applied to it, this made it look like there was a 3D matrix of playfields scrolling past the camera, or the camera was moving through a 3D matrix of playfields. At the very end of the song, I also eased the five quads together in the z direction, returning them to the same place the initial quad started, thus making it look like the playfields were collapsing in on each other to return to normal. This effect turned out to be more difficult to read than I was anticipating, and thus I did not bother adding any more effects to this part of the song as they would just make it hard to read.

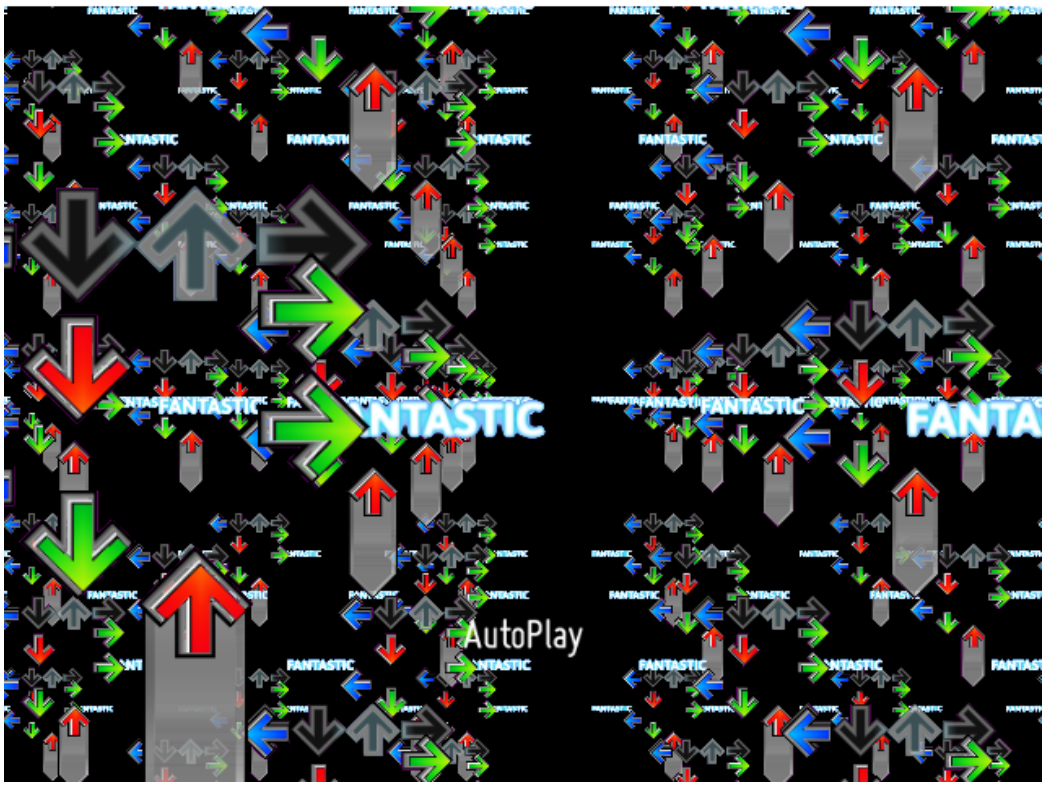

Figure 4: Layered Quads with Grid Shader Applied creating 3D Matrix Effect

Lessons Learnt:

I learned many things from this project, from the basics of step charting to how to write NotITG mod files, and from how to write basic lua to how to write GLSL fragmentation shaders in C++. I learned how to overcome interesting challenges and work around odd quirks of an engine to create the visual effects I wanted. I got to interact with the NotITG community, and even became good friends with one of the members. However, in addition to the skills I picked up during this project, I also learned a few lessons.

Well-written, complete documentation is very important when making a library or other such framework for others to use. When trying to create the mod file, there were many times where I could not find what I was looking for in the NotITG documentation (only to find that a more complete documentation was hidden elsewhere on a completely different site to the official documentation page). I also had difficulty with The Mirin Template's documentation as it simply was not completed in places and lacking in quality in others. The online resource I was using to write shaders was also not completed and seemingly abandoned by its creator, making the later stages of shader work much more difficult. Without the help of the NotITG community, especially "SheepyPeka", this project would not have been completed.

I also learned the importance of time management and project planning. While I did start working on this project very early, I was still pressed for time near the end as I simply had not done enough soon enough (this report included). I was always rushing to pull everything together at each deadline. This was also partly due to not fully having a plan for what I was going to do and what effects I wanted to make, so I didn't know what to expect in terms of how much work I was going to need to put into this project in the beginning. It also meant I wasn't able to incorporate all the things I wanted to in this project near the end as I started getting more and more ideas. Not planning out the project also caused me to have to try and learn how to write a shader that could be used in NotITG from scratch in just under a week, which really was not ideal.

I also learned just how much easier working on a bigger project is when always trying to write clean and modular code. While I didn't focus too much on writing modular code, I put in extra effort to write cleaner code than I usually do, which was really a blessing when I had to go in and change a whole series of effects on only one player specifically or throw out an entire series of elements for an effect to reimplement it a completely different way. While, granted, the code I wrote isn't super clean (especially when it comes to the shader I wrote), it was way easier to work with and modify than what I have normally done in the past which helped greatly, especially as I was pressed for time.

Possible Future Improvements:

The current mod file has a few problems that I would like to fix before releasing. At the current moment, the shader used at the end of the song has a problem with the chromakey that leaves a thin magenta border around every element in the playfields. This is due to the chromakey threshold not being lenient enough to fully remove the magenta coloring as it starts to get blended with the outer edges of other elements. However, if the threshold for the chromakey is too lenient, it will start to key out parts of the playfield. Work has already begun on fine tuning the chromakey by experimenting with different thresholds and background colors (as the magenta used is fairly close to some of the red coloring used to color the arrows in the playfield).

Due to how the playfields were aligned to the top left corner of the scene before being moved back to the center with the use of proxies, the perspective on the playfields is slightly off. This can be seen by how the left side of the arrows are much more visible than the right side. This has already been fixed for the release of the chart by centering the playfields to the center of the scene, using a proxy to move them to the left side of the scene (so the proxy can be used with the rotation effects), and then using another proxy to move the playfields back to the center of the scene. This has the side effect of changing how much the playfields get rotated during the drop section, which is currently being fixed for release.

The shader used at the end of the song has another problem. The playfields move too quickly to realistically be parsed correctly by the player, and thus some fine tuning is needed to ensure that the playfields move slowly enough to be read but still quickly enough to be challenging, all while remaining dynamic enough to feel satisfying.

Aside from the column swaps, the beginning section is quite bare in terms of effects. I would like to add a few more small movement effects for flare to the beginning section to make it more satisfying, while being careful to not cause the section to become too much more challenging, as the column swaps are already fairly challenging by themselves.
NotITG also supports adding custom 3D models into the scene. I originally intended on adding 3D text to the middle of the song that said “IT’S ALL ABOUT HUMANITY” (timed to appear when the same is said in the song), however due to time, I was not able to start implementing this. Ideally, the models can be created in Blender, an open source 3D modeling software and converted to a model file the NotITG Engine can use through a 3rd party tool someone created for this exact purpose. I do not fully know how to use Blender to create models, however, nor how to implement a model in the NotITG Engine, and thus I would have to learn how to do both.

Finally, I believe that the final step of the chart should feel significantly more impactful than it currently does. Ideally, in addition to collapsing the playfields back down to normal (as it is already implemented in the file), I would like to also have some sort of effect that causes the playfields/receptors to dissipate in a satisfying way. I have not quite figured out what I want to do to accomplish this, but I would like to have something there before releasing the file.

Conclusion:

This project was born out of a two year long desire to create a working mod chart in NotITG. For the longest time, I felt like I simply did not know enough to start working on one, and never felt motivated enough to learn the basics. Because of this, I decided to use this project as an opportunity to learn how to create a NotITG mod file while also practicing what I have learned in class. Although this project was very challenging, especially as I was working alone, I had a lot of fun and I feel like I made an enjoyable mod file. As of right now, I want to fix a few issues and add a bit more to it first. Once it’s finished, I plan on releasing the file such that it can be downloaded and played on the current version of NotITG.

Glossary:

The following terms have been used throughout this paper, though their meaning may not quite be understood by the reader. For that reason, I have gathered a list of the more ITG/NotITG specific terms and have given them definitions (or, in most cases, used the definitions provided by DDRGuide [13]).

Chart (Step chart): “A gameplay sequence for a particular song … During gameplay, the chart lays out where notes (and other gameplay elements like [mines] …) are placed.”

Simfile: “A song bundled with charts and metadata for use in StepMania [and forks]. Simfile is short for ‘simulation file’.”

Mod File (Mod chart): A special simfile for the game where different mods and other effects are applied throughout gameplay.

Mod (Effect): “A configurable aspect of gameplay.” Mods can affect the visual aspect of notes such as visibility and scroll speed, among other things.

Sync: “A measure of how precisely the notes match a song's audio.”

Receptors: “The visual element typically seen at the top of the screen used by players to determine when to hit each note. Each panel being used in the current gameplay mode has a corresponding arrow-shaped receptor towards which notes will scroll. The receptors flash in sync with the music on each quarter note.”

Note: “The main scoring element during gameplay, represented visually by an arrow scrolling towards the receptors. Players try to step on the corresponding panel as close as possible to the moment when the arrow coincides with its receptor; the game assigns them a judgement based on how precisely they timed their step.”

Jump: “Two different notes on the same horizontal row, typically hit by jumping with both feet onto the corresponding panels.”

Hold: “A special note where the player must keep the corresponding panel pressed until the "tail" of the [hold] note has passed.”

Roll: A special note where the player must keep repeatedly pressing the corresponding panel until the "tail" of the roll note has passed.

Mine: “A special gameplay element … Players must avoid standing on the corresponding panels when it passes in order to avoid triggering [the mine], which causes a miss.”

Pad (Dance pad, Stage): “The physical box that houses the panels and connects to the main game cabinet. Players typically stand on this stage during gameplay.”

Panel: “One of the primary gameplay inputs, typically activated by a player stepping onto it. There are four panels per side, for a total of eight panels on a standard dance stage.”

Pattern: “A particular sequence of notes, typically categorized by the techniques required to hit the notes rather than the specific arrows involved.”

Measure: “Typically four beats of a song; sometimes fewer, sometimes more.”

Beat: “The most commonly understood subdivision of a song according to its tempo, which is accordingly measured in beats per minute (BPM). One beat is equivalent to a quarter note.”

References:

[1] *Not In The Groove.* (v4.2.0), HeySora. [Online]. Available: https://www.noti.tg/

[2] “XeriOl”. “The Mirin Template.” Mirin Template | The Mirin Template. https://xerool.github.io/notitg-mirin/ (accessed Feb. 21, 2022).

[3] *GNU Image Manipulation Program.* (v2.10.10), The GIMP Team. [Online]. Available: https://www.gimp.org/

[4] *REAPER.* (v6.32), Cockos Incorporated. [Online]. Available: https://www.reaper.fm/

[5] *ArrowVortex.* (2017-02-25), Bram ‘Fietsemaker’ van de Wetering. [Online]. Available: https://arrowvortex.ddrnl.com/

[6] Max Brhon. *Humanity.* (Jun. 26, 2019). Accessed: Mar. 15, 2022. [Online Audio]. Available: https://ncs.io/Humanity

[7] “HeySora”. “Lua API for OpenITG / NotITG.” Lua for oITG / NotITG. https://sm.heysora.net/doc/ (accessed Feb. 20, 2022).

[8] “Sphinx”. “Welcome to NotITG’s (unofficial) documentation!” NotITG 4.2.0 documentation. https://craftedcart.gitlab.io/notitg_docs/index.html (accessed Feb. 21, 2022).

[9] “Lua Documentation.” Lua: documentation. https://www.lua.org/docs.html (accessed Feb 21. 2022).

[10] P. G. Vivo and J. Lowe. “The Book of Shaders.” The Book of Shaders. https://thebookofshaders.com/ (accessed Apr. 5, 2022).

[11] “SheepyPeka”, private communication, Feb, Apr, 2022.

[12] “0b5vr”. “Flip-Invert.” 0b5vr.com. https://0b5vr.com/flip-invert/ (accessed Feb. 21, 2022).

[13] “Glossary.” DDRGuide. https://ddrguide.com/glossary/ (accessed May 5, 2022).

[14] *Glitchy colours.* (2017), jojjesv. [Online]. Available: https://www.shadertoy.com/view/MlVSD3

[15] *Shadertoy.* (2022), Beautypi. [Online]. Available: https://www.shadertoy.com/

[16] *GLSL Sandbox.* (2022), mrdoob. [Online]. Available: https://glslsandbox.com/