DJ Simulator: A Visually Authentic DJ Experience

COS426 Final Project by Matthew Fastow and Gregory Smith
Advised by Sahan Paliskara

0. Abstract

This report documents the design, implementation, and results of “DJ Simulator”, a COS426 project and ThreeJS web application in which the user plays a memory game inside a quasi-realistic nightclub setting. The number of dancers on the dance floor grows and shrinks in number depending on how well the player is doing in the game. Furthermore, many aspects of the nightclub scene itself may be altered by the user via the provided GUI, such as the strobe lights and the crowd composition. The implementation of the project’s core components span many course topics including mesh editing, scene topology, and raycasting. The result of this project is quite visually appealing and intuitive; future work may involve optimizing memory usage and expanding gameplay mechanics.

1. Introduction

1a. Goal

Each of our goals for DJ Simulator were classified as either an “MVP” (Minimum Viable Product) goal or a stretch goal. Our MVP goals included the rendering of a nightclub scene complete with a multi-faceted DJ mixing table, a dynamically lit dance floor, animated dancers, and speakers. In addition, the MVP would be an interactive game, i.e. it would provide a task for the user, listen for user input via event handlers, and convert that input into a score of some kind. Another baseline goal was to incorporate audio into the application to provide the user with an auditory indication of their current standing in the game. We successfully achieved all of these MVP goals at the time of writing (described in more detail in Section 2d-2g).

In addition to these MVP goals, we set stretch goals to enrich the project further. Dynamic spotlights moving about the scene would help to recreate the feel of a nightclub; we were able to implement this feature, even allowing users to alter the spotlights’ settings through a GUI (these features are described in 2h-2i).. Another stretch goal we defined for this project was to have the crowd dynamically respond to how well the player is performing in the game. As of now, additional crowd members will appear when the player completes a sequence correctly and a crowd member will leave when the player has failed to recreate the sequence. With additional time we may experiment with different arrival/departure animations; as of now, crowd members fade in and out when they appear and disappear.

1b. Previous Work

DJ-themed online games are quite popular; in fact, there exists an entire website dedicated to these types of games. Many of these games task the player with manipulating aspects of the DJ’s table and use auditory and visual cues from the audience to indicate how the player is doing. Some, but not all of these games focus on the auditory aspect of DJing, allowing the user to manipulate the music being produced using the DJ table’s controls. The

sheer volume of games in this niche indicates that this type of game is beneficial to a wide audience, which is a primary reason that we decided to pursue it for our project.

Outside of the project's theme, the implementation of several of our project's core components arose from researching and reviewing previous work. This includes Cat Dash (a COS426 project from Spring 2020 which created a floor similar to our club dance floor using dynamic shaders) as well as an array of ThreeJS examples for rendering walls, rendering text, applying animations to our meshes, and loading complex meshes GLTF meshes. We found that many of these examples did not modularize their code into separate `scene.js` and `app.js` files; for a project of our scope, we found that a more modular approach would allow for easier debugging and cleaner code.

1c. Approach

While the approaches taken in creating the nightclub scene, animating characters, and switching between scenes are relatively straightforward, decisions regarding the gameplay mechanics required additional thought. We chose to implement a memory-based game, where the user is presented with increasingly-long sequences of DJ table elements to remember and then repeat.

This game mechanic was among three potential gamification schemes that we discussed. Another option we considered was a "Whack-A-Mole" style game where the player's reaction time would be tested. Objects on the DJ table would light up and the user's score increase would be determined by the time between the object lighting up and the player clicking the object. User feedback indicated that the game mechanic was too boring.

Another option we considered an audio-mixing game: users would interact with the DJ table elements to apply effects to the music being played. While this game likely would have been similarly engaging, it presented a number of technical challenges that we were unable to tackle within the condensed time frame. For example, we would have had to construct our own DJ table using an array of meshes to allow the manipulation of each element directly; the available free meshes fitting this criteria were limited. Additionally, we would likely have had to dedicate a majority of our time on the project to dynamically modifying the audio being played; given that the course's focus is on Computer Graphics, we decided to opt for simpler gameplay mechanics which would allow us more time to enrich the visuals of the nightclub scene.

2. Methodology

2a. Project Architecture

DJ Simulator was built in JavaScript using the ThreeJS library and the project architecture closely resembles that of the starter code provided by the COS426 staff. The scenes, camera, audio, and event handlers, and render loop are all initialized in the `app.js` file. The game takes place in a nightclub, which is defined in `NightclubScene.js`. Each of the game's screens (main menu, instructions, game over screen, and the game itself) are Scene objects which inherit from `NightclubScene.js`; using this architecture, we were able to make

changes to the nightclub scene in a single file and have these changes reflected across all of the scenes in which the nightclub is rendered at once.

Other components of the application include textures, objects, lights, and audio sources, each of which was assigned a separate folder within the `/components` directory. The objects in the game include text objects, advanced 3D meshes found on SketchFab, and planes manually created to represent the nightclub's walls. We chose to store the various textures associated with the more complicated objects' meshes in their own subfolders for clarity. A visualization of DJ Simulator's project architecture is provided in **Figure 1** below.

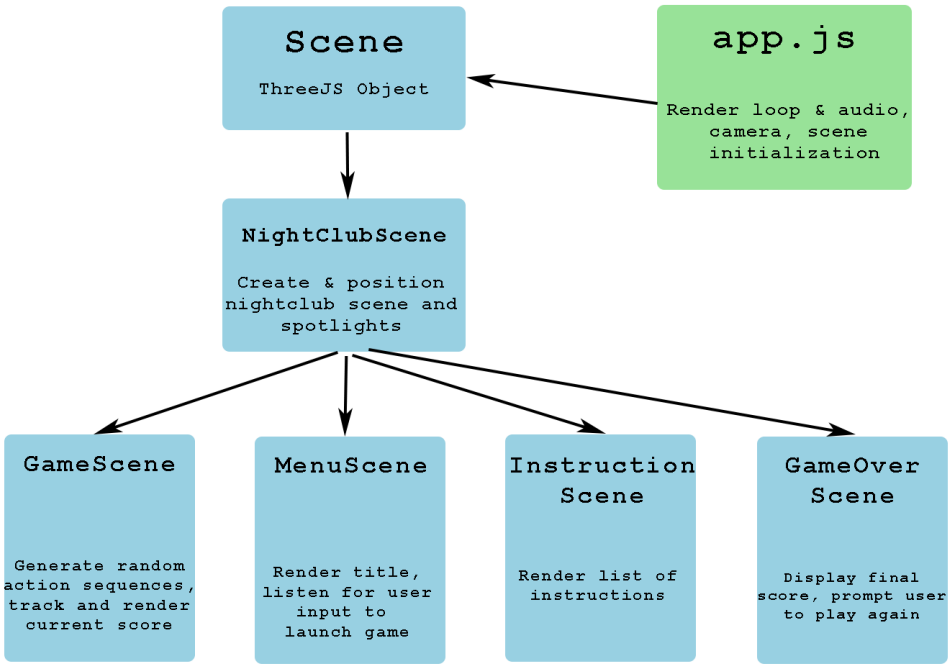

Figure 1: DJ Simulator Project Architecture

2b. Text Objects

A goal of ours when implementing this project was to render all text required for the game (title screen, score, instructions) as 3D in the scene. This was accomplished by defining a `Text` object which takes as parameters its scene, text data, position, and size. Using these parameters, ThreeJS `TextGeometry` and `MeshLambertMaterial` objects are initialized and used to generate a `Mesh` which is then added to the parent scene. This modular text object abstraction allows us to easily generate text for the game without creating individual objects for each instance of text, which was our initial approach.

One implementation hurdle we encountered with regards to text stemmed from the fact that `TextGeometry` objects are initialized asynchronously and therefore our `Text` object's

properties cannot be changed easily in the Scene file in which they are created. Because of this detail, we initially could not update the player's score after a sequence was completed. We were able to get around this by creating functions which dispose of a given text object and create a new one in its place for score updates, instructions, and performance indicators.

2c. Complex Mesh Objects & Wall Objects

The majority of objects rendered in DJ Simulator are quasi-realistic 3D meshes discovered online and downloaded in the .gltf format. These objects include the DJ table itself, the trusses in each of the four corners of the nightclub and the speakers mounted atop them, and the various dancers in the night club. Each of these objects is defined by a `.gltf` file, a `.bin` file, a `.js` file (stored in `components/objects/{object_name}`, and textures in the .jpg format (stored in `components/textures/{object_name}`). Each object's `.js` defines the constructor for the object, loads in the associated `.gltf` file, and provides an `update()` function (which allows the dancers' animations to play and the DJ table's disc to spin). These objects are exported for use in the game's scenes.

The walls of the DJ's nightclub are defined in their respective `.js` files using the ThreeJS library's `PlaneGeometry`, `MeshStandardMaterial`, and `Mesh` objects. The wall's properties such as location, rotation, and color are initialized in this file such that they form a four-walled room when created and added in `NightClubScene.js`.

2d. Game Logic & Scoring

The implementation of the game's logic resides primarily in `GameScene.js`. The game scene has three states: `demonstration`, `attempt`, and `gameOver`. During the `demonstration` state, arrows hover over the elements that the user is supposed to click, in chronological order. User clicks are not intercepted when the scene is in this state. Once the demonstration has completed, the scene's state is changed to `attempt`, where user clicks are listened for and handled. The user must repeat the sequence in the exact order that it was presented to them.

The scene remains in the `attempt` state until one of two outcomes occur: (1) the user completes the sequence successfully, or (2) the user clicks an incorrect table element. In either case, `score` is adjusted accordingly, and the scene returns to demonstration state. If the user was successful in replicating the sequence, a new sequence with a length that has been incremented by one is generated, the scene's state is switched back to `demonstration`, and this new lengthened sequence is shown. Otherwise, the state is switched to `demonstration`, and the same sequence is repeated. This cycle repeats until the song completes, at which point the scene's state is changed to `gameOver` and the user's final score is displayed along with a prompt to play again.

2e. Event Handlers

To add interactivity to the game, we intercept user keystrokes and clicks. We implemented this interception using JavaScript event handlers paired with the ThreeJS

`Raycaster` class. Initially, the user is presented with the title screen, which welcomes them to the game. To progress beyond the title to the instructions, they are told to press the space bar, an event which is listened for and handled in `app.js`. If the user is detected to have pressed the space key, the scene to be rendered is assigned to an `InstructionScene()` object, rather than a `MenuScene()` one. This same process allows the user to progress from the instruction page to the game itself.

Once the user is playing the game, their clicks must be intercepted to allow them to mimic the randomized sequences. This interception is performed in the following way. First, everytime a `mousedown` event occurs, the appropriate event handler in `app.js` is triggered. This event handler reads both the mouse location and the browser window size to determine the click location in 2D scene coordinates. It then uses the ThreeJS `Raycaster` library to cast a ray into the scene at this location, and return whichever object (if any) the ray intercepts. If the ray does intersect an object, this object is passed to the `GameScene` by modifying the `scene.state.selected` field. The scene continually checks this field in its update loop, and responds appropriately.

2f. Dynamic Shaders

The dynamic fragment shader we utilized for the dance floor comes from the graphical artist Daedelus[citation needed] and has been used in previous COS426 projects. While we did not create the shader ourselves, we initialized the required ThreeJS `TextureLoader`, `PlaneGeometry`, and `Mesh` objects to apply this shader to the floor and position it correctly. We were quite satisfied with the result of this shader, which brightens the scene dynamically and mimics the effect that a disco dance floor might have visually.

2g. Audio

Our audio sources are .mp3 files discovered via YouTube and are stored in the `components/audio` directory. Each of these sources are loaded and made playable in the `app.js` file using ThreeJS's built in `AudioLoader()`, `AudioListener()`, and `Audio()` objects. Each `Audio()` object, once loaded, is placed into a JS object which is passed into each scene's state to allow for sounds to be played and paused within the scene.

An issue we encountered regarding audio was that certain web browsers (such as Google Chrome) do not allow for applications to play sound until the user has interacted with the page. For this reason, the background music which plays on the main menu screen is only played on browsers which have not placed this restriction, such as Safari. As of now, the menu music will begin to play once the user has tapped the space bar to continue to the instructions screen when the game is played on Google Chrome.

2h. Spotlight Abstraction

As part of the nightclub aesthetic, we utilized ThreeJS' SpotLight library to create a higher-level spotlight abstraction, which resembles strobe lights that one might see in an actual nightclub. The `StrobeLight` API contains several configurable components, most of which are

available to the user in the scene GUI. These configurable components are listed in **Figure 2** below.

Figure 2: Properties of `StrobeLight` API

Property	Type	Description
`blinking`	`boolean`	Determines whether the light blinks or remains 'on'
`strobeSpeed`	`int`	Determines how fast the strobe light blinks
`color`	`string`	Color of the light as a hex value in the RGB basis
`spinning`	`boolean`	Determines if the light is moving in a circle or stationary

In each scene, both `StrobeLight` (defined in `StrobeLight.js`) and `LightTarget` (defined in `LightTarget.js`) objects are created. In the scene constructor, the `StrobeLight` object is then pointed at the `LightTarget`, which has the ability to move dynamically throughout the scene. The `StrobeLight` then adds various of its state fields (listed above) to the parent GUI, and continually checks this state in its `update()` function. If it detects that some of its state has changed, it updates accordingly. We experimented briefly with alternative implementations available through ThreeJS (`HemisphereLight`, `DirectionalLight`), but these alternatives were not nearly as realistic as the `SpotLight` abstraction.

2i. GUI Options

There are two spotlights moving about the scene while the game is being played. Each of these spotlights can be modified directly by the player in the GUI on the left-hand side of the screen. These options include b`linking` (which determines whether the lights blink to produce a strobe effect), `strobeSpeed` (which, when `blinking` is enabled, determines how quickly the lights blink), and `color` (which determines the color of the light). Once the user modifies these values in the GUI, their changes are instantly reflected in the game. Before the final deadline, we would also like to add an option to change the crowd's composition (by default, it is a random assortment of aliens, Shreks, men, and storm troopers).

3. Results

We user-tested the game with six users, collecting their feedback at various points throughout the development process. We have summarized the main highlights of this feedback below.

There were several aspects of the aesthetics and gameplay that users appreciated. First, they thought that the nightclub scene was relatively convincing, and similar to nightclubs that they themselves had visited in the past. They also found the audience makeup and general premise somewhat humorous, which is comforting to hear as the primary goal of the game is to

entertain the user. Finally, they found the game's mechanics straightforward, intuitive, and easy to learn on the first play-through.

However, there were several ways that they suggested we might improve the game. The most common suggested improvement was to make the gameplay more intricate. While intuitive, most users agreed that the premise is still simplistic, and they would have appreciated being challenged more or having the opportunity to play different "mini-games" within the same nightclub setting. Additionally, some users indicated that the game and load screen became somewhat laggy depending on the browser they played it in.

We attempted to perform a quantitative assessment of our game - in addition to this qualitative assessment - by capturing the FPS and load screen time across various browsers and computing devices. However, the FPS module from previous assignments was not easily portable to our NodeJS application, and thus we were unable to gather the necessary data.

4. Discussion

4a. Follow Up Work

If we had more time to work on this project, there are several initial changes that we would make. First, we would hope to implement optimizations that make the game more portable to different browsers and computing environments. Several optimization ideas are as follows. First, we would focus more on removing meshes from the scene the moment that they are no longer needed (rather than making them invisible, as we do with the dancers and the arrows). Second, we would focus on "preloading" many of the meshes that we might later need, such as possible "scores" that the user will achieve, rather than constructing them during gameplay execution. And third, we would focus on combining the menu screen, instruction screen, and game environment into the same "scene" object, such that switches between different scenes (and thus unnecessary reconstruction of the nightclub) is eliminated.

Along with optimizations, we would like to expand gameplay mechanics in several ways. While the clicking-based gameplay is a reasonable first attempt at a browser game, it becomes somewhat easy after several run-throughs. We believe that we could make the memory sequence more challenging by combining clicks with keystrokes and other tasks (e.g., dragging/dropping items), raising the challenge ceiling of the game. Additionally, we would like to lean into the DJ aspect of the game more, and actually offer the user the ability to mix audio as a DJ might.

4b. What We Learned

Overall, the main constraint to the breadth of this project was the availability of free meshes/animations via open-source online repositories. Many elements of the game were built around the constraints imposed by available meshes: audience members were forced to "teleport in" rather than walk in, because walking animations often were not available; and DJ table discs/dials could not be moved, because the underlying mesh did not make them configurable elements. In some ways, these barriers encouraged us to be more creative with the game premise and environment, but we would also hope, in future animation projects, to

control the mesh/animation production stage as well, thus giving us more creative autonomy over the project.

5. Conclusion

DJ Simulator is an in-browser, ThreeJS-based game that immerses the user in a quasi-realistic virtual nightclub. The user is tasked with remembering a randomized sequence of DJ table elements that they must click in order to increase their score.

During gameplay, the user will receive positive/negative feedback based on their performance in several ways. First, messages will appear at the top of the screen, congratulating the user when they do well and criticizing them when they do not. Second, the audience size will fluctuate based on performance: audience members will show up if the nightclub gets lit or leave if it does not. And third, the audience will either cheer or boo at the end of the song based on overall performance. The nightclub is configurable to the user in several ways. They can change the color of the strobe lights, as well as whether they strobe and, if so, how fast. They can also change the composition of the dancers in the crowd.

The project implementation incorporates many key topics from the COS 426 curriculum. The nightclub itself is a collection of 3D mesh objects, for which we perform several mesh editing operations, such as scaling, translating, and smoothing. The clicking mechanic is achieved via raycasting, where the first scene object that a ray intersects is considered “selected” by the user. And finally, scene hierarchy/topology, camera mechanics, and dynamic shaders are all utilized to make the nightclub appear as realistic as possible.

6. Contributions

6a. Matthew’s Contributions

Matthew’s main contributions to this project included constructing the original nightclub scene (meshes, spotlights, and dancers) and implementing the gameplay logic.

6b. Greg’s Contributions

Greg’s main contributions to this project included implementation of the menu, instructions, and game over scenes, audio components, modular text objects, and responsive elements (positive/negative feedback, audience members coming/going).

7. Works Cited

Online Tutorials

Floor Shader (Cat Dash): https://github.com/littleCatEvelyn/Cat-Dash

Raycasting: https://threejs.org/docs/#api/en/core/Raycaster

3D Models and Animations

DJ Table: https://sketchfab.com/3d-models/dj-roomv01-31fe3b72d76842b1bcd0660a02f8ac09

Speakers: https://sketchfab.com/3d-models/speaker-ba43293ea8504d3698fbe38002253c48

Truss: https://sketchfab.com/3d-models/truss-74759f3b36f741ec82803efe4435f31d

Arrow: https://sketchfab.com/3d-models/cc0-arrow-5-f38febed165240b9aea5289c215c6b65

Dancer 1 (Regular Man):
https://sketchfab.com/3d-models/hip-hop-dancing-4274cabfd36147739a08197486d4f63b

Dancer 2 (Stormtrooper):
https://sketchfab.com/3d-models/dancing-stormtrooper-12bd08d66fe04a84be446e583d6663ac

Dancer 3 (Alien):
https://sketchfab.com/3d-models/alien-hip-hop-dancing-fac7a402b9f940aa955da87f8eb0619c

Dancer 4 (Shrek):
https://sketchfab.com/3d-models/shrek-hip-hop-dance-b0ad224b063041d0ab5419a054f6f646

Audio Files

Main Music (*Animals* - Martin Garrix): https://www.youtube.com/watch?v=gCYcHz2k5x0

Intro Music (*CLUB* - Andrew Huang): https://www.youtube.com/watch?v=pWI7b_A8UbU

Crowd Boo: https://www.youtube.com/watch?v=y1U6g-kJ5og

Crowd Cheer: https://www.youtube.com/watch?v=HkUb5nFGWjs

Action Failure: https://www.youtube.com/watch?v=9M3cjtS68Jg

Action Success: https://www.youtube.com/watch?v=0On57DooNI4

Button Press: https://www.youtube.com/watch?v=h8y0JMVwdmM