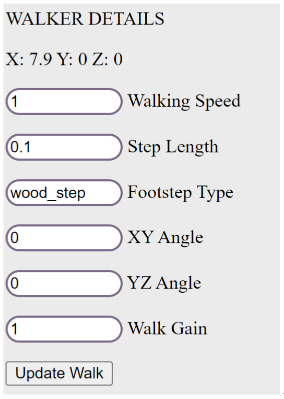
Felix Goes on A Walk: Writeup

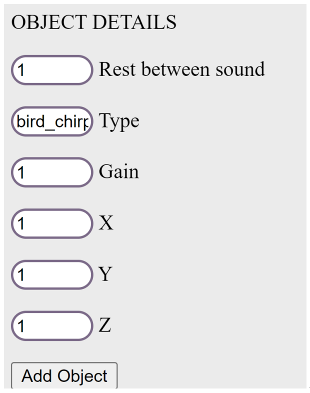
In the application “Felix Goes on a Walk”, I sought to create a live programming toolkit that allowed users to simulate sound and movement in a 3d space. Users are able to set a walking speed, pace, footstep type, gain, and a direction in degrees (as seen in *Figure 1*). The direction is divided into two fields, one being the xy-plane, and the other being the yz-plane. The footsteps are customizable, allowing for different footstep samples to be played as well as adjusting the gain and amplitude of the footsteps. The walking begins upon booting up the program, and can be updated at any time using the update walk button.


WALKER DETAILS

X: 7.9 Y: 0 Z: 0

1	Walking Speed
0.1	Step Length
wood_step	Footstep Type
0	XY Angle
0	YZ Angle
1	Walk Gain

Figure 1

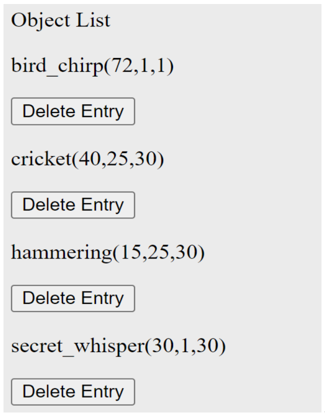

OBJECT DETAILS

1	Rest between sound
bird_chirp	Type
1	Gain
1	X
1	Y
1	Z

Figure 2

The user is also able to create stationary objects with an x-, y-, and z-value (as seen in *Figure 2*). These values are the object's placement in space in relation to the walker. As the user “walks” the objects’ relational position will correspondingly change, sounding closer or farther away from the user. The objects can be further altered in their creation with the fields “Rest”, “Type”, and “Gain”, which respectively determine the space between each sample play, the sample played, and the gain of the sample.

The details of the object are also stored with their position (as seen in *Figure 3*). I decided to not allow the objects to move primarily for reasons of feature bloat. The code I created allows for the changing position of sound objects, but I wanted the user to focus on the movement of a user through an area of static objects. The object detail storage additionally allows users to delete sound-objects created, erasing both their sound and visualization.


Object List

bird_chirp(72,1,1)

cricket(40,25,30)

hammering(15,25,30)

secret_whisper(30,1,30)

Figure 3

There is also a simple visual representation of the relative position of created objects to the user. The visualization takes place along the x-, y-, and z-axis, (as seen in *Figure 3*). Ideally, in a future expansion of the project, there would be a 3-dimensional representation of the walk, rendered in Blender, WebGL, or another 3-dimensional visualizer. However, for this project I primarily wanted to visualize the moment that the user passes an object, and on which axis they pass the object. The moment of the cross-over can be seen in *Figure 4* and *Figure 5*, a visual which should hopefully aid the user in understanding how they are hearing the current sound.

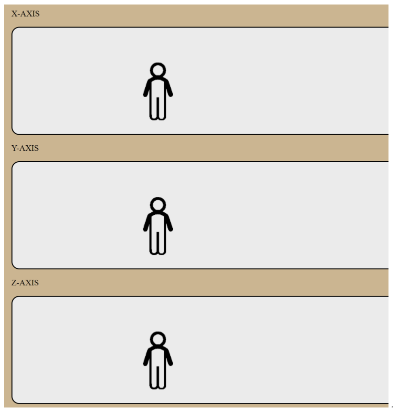


Figure 4

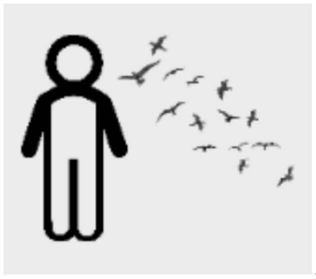


Figure 5

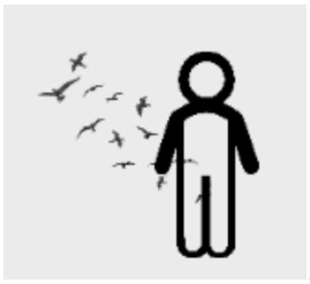


Figure 6

In coding this project I used WebAudio, specifically WebAudio's PannerNodes. Upon arriving at the webpage, the code generates a Walker object, a simple dictionary which holds all of the details listed in *Figure 1*. When clicking "Update Walker", the Walker dictionary is updated with all the data currently listed in the fields under "WALKER DETAILS". Once every "Walking Speed * second" the x-, y-, and z- position of the walker is updated, also stored in a dictionary. Correspondingly, the html walker position text is updated, and the relative position of the walker to each sound object is checked, and if they are within 20 units (Panner positions, unsure of the formal units), then they are rendered corresponding to their distance from the user in each plane.

When a new sound object is created, the sound's data is stored as a dictionary in a dictionary of dictionaries "activeSounds". The unique soundID is then passed to a recursive function, which calls itself according to the "Rest between sounds" parameter. Each time the sound object is called it first updates the PannerNode by subtracting the walker position from the soundData position. Then, it plays the sound. By storing the audioBuffer of the sound in the soundID's dictionary, it means that the function does not have to await loading the sample, saving some time. I was wondering for a while if I should update the sound's position in the “playSound”, or “playWalker” function, but eventually decided on “playWalker” because the created sounds are unable to move, and if I updated based on the sound object’s replay rate the movement would be staggered and jagged.

In visualizing the objects, I used an html canvas and png images. I also included a sound-to-images map, which maps the name of the samples to the name of the images, so that to update each canvas I could just iterate over the “activeSounds” map, check relative position, and then, if within 20 units of the walker, display the corresponding png.

I really enjoyed working on this project, and I think that it is something I would like to pursue further— I think VR would be a particularly fun and salient space to play with sound spatialization. Thank you for a wonderful semester, and I look forward to working with you next semester!

Nicholas Denton-Cheng