PROTOTYPE DOCUMENTATION: SPACE.JAM

CASSIE SMITH
CART 351

PROJECT SUMMARY

SPACE.JAM is an explorable networked space aimed at the co-creation of new shared rituals of sound and movement. Users can choose from a variety of configurable virtual instruments, and open live "jam" sessions with their friends. The app uses machine learning to track users' poses and gestures, allowing them to play and interact with the instruments, and with each other, through dance-like movements that engage the whole body. SPACE.JAM engages collaborative musical creation and performance as an outlet for collective expression and the generation of shared meaning.

FEATURES

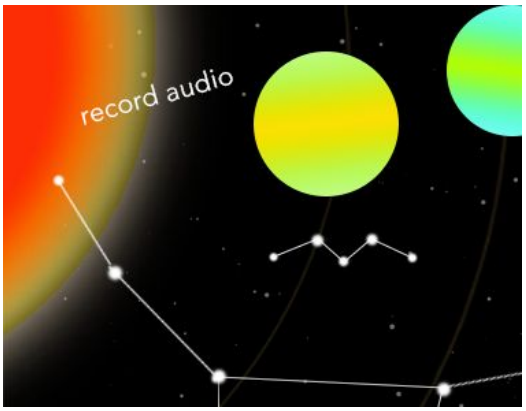
VIRTUAL INSTRUMENTS (SOUND)

Right now instruments are grouped by category, and users will also be able to view a list of all available instruments in alphabetical order. The current categories are:

- ➔ Keys and unpitched percussion instruments (such as piano and xylophone)
- ➔ Rhythm and unpitched percussion instruments (such as drums and cymbals)
- ➔ Strings
- ➔ Brass
- ➔ Oscillators
- ➔ A category that allows users to create custom samplers (possibly dependent on integration of persistent state features)

Sounds are generated using TONE.JS. The API's Sampler class can use a sound file with a single note as the basis for automatically generating other notes. For example, it will repitch a sound-file of the note C4 to generate additional notes with the same timbre as the original sound: D4, E4, etc. along the entire chromatic range. This saves load time, making it easy to synthesize a variety of instruments with a very minimal sound library. I am sourcing many sounds from the excellent Versilian Community Sample Library (CC0).

VIRTUAL INSTRUMENTS (GRAPHICS)

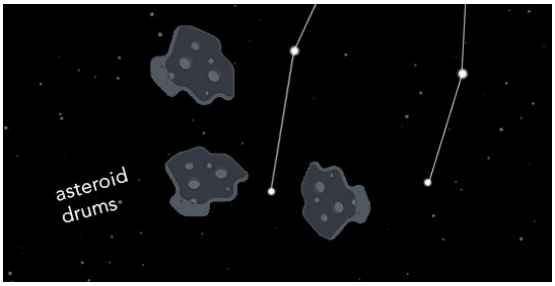


The theme is Outer Space, a tongue in cheek nod to the concept of an "explorable networked space". Each user is represented by a pose skeleton with ellipses marking each keypoint. The resulting visual is reminiscent of a constellation. Instruments are represented as celestial objects: asteroids, comets, planets orbiting a sun, etc. The sun is a RECORD BUTTON - users can tap with their wrist keypoint to record the audio output of the session.

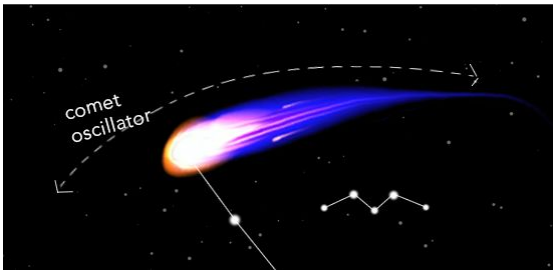
The goal is for each user's position to be transposed onto the same shared canvas, allowing two distinct users to navigate the same solar system. Each will have the ability to move around and play any of the instruments rendered in the space. In cases where the size of the display window is too restrictive. Users will exist in their own "sector" of the solar system, playing their own unique instrument. They will be able to see the other user represented in a thumbnail view.

VIRTUAL INSTRUMENTS (GESTURE INTERACTION)

Instruments are laid out as configurations of sound objects in the main app and sounds are triggered by different gesture interaction. Gestures vary based on the type of sound generated by each instrument, and triggers can correspond to different pose tracking keypoints.



For example, drums are triggered by "kicking" the sound object (when the ankle keypoint crosses the position of the object). Properties such as amplitude may be tied to velocity.



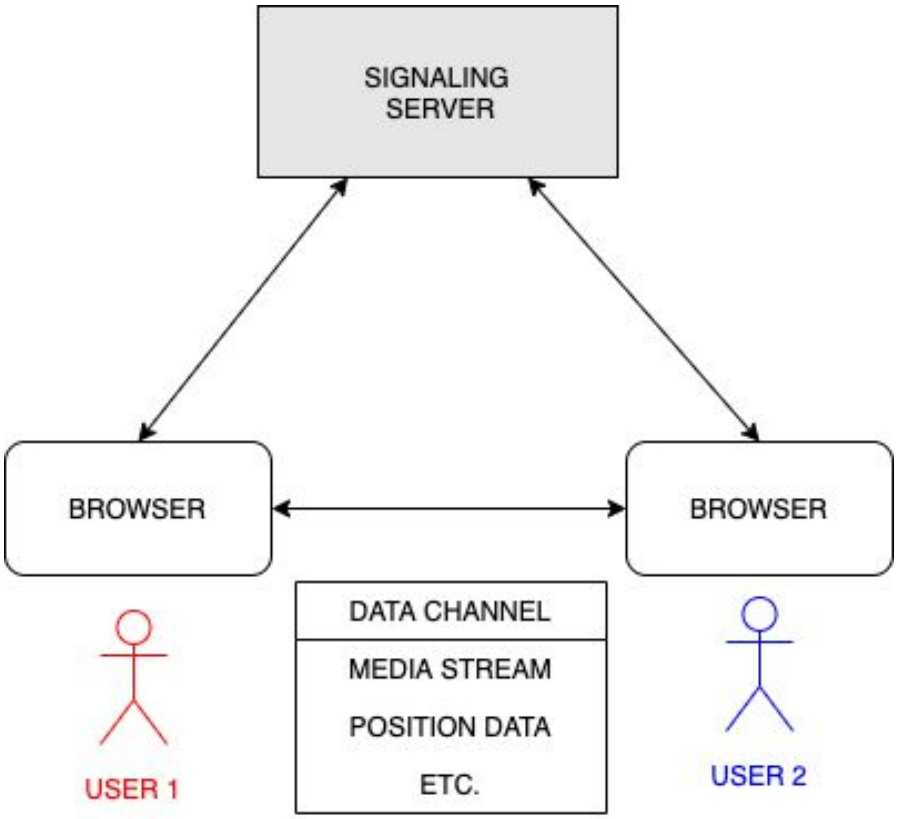
In the case of oscillators - or other instruments that emit continuous (or legato) rather than discrete (or staccato) sounds - gestures include dragging the sound object along a path to modulate the pitch or control the length of the note.

Originally, my goal was to have users step back from the camera, so that their entire body was in the frame. This offers more possibility for interaction via distinct trigger keypoints and encourages users to move with their full body interacting with the instruments in a kind of ritual dance.

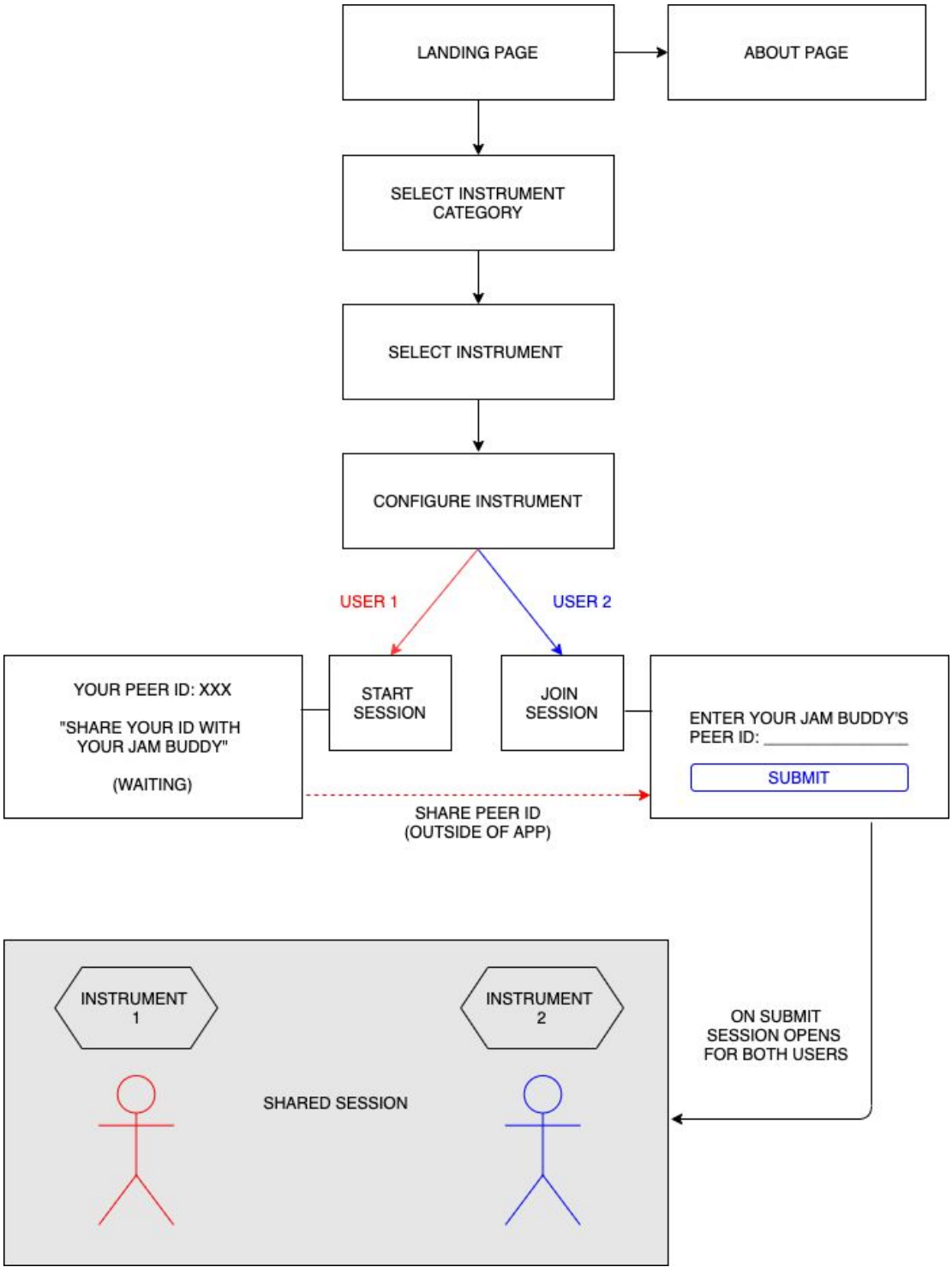
However, in running tests, the pose tracking seems to lose some fidelity when the user is so far back from the camera. I also realized that not every user will necessarily have access to a space, large enough to stand at such a distance from their computer. Still, I want to avoid a situation where users were just sitting at their desk (as this removes the element of dance, which was the basis for my interactive vision). I compromise by having the user stand and step back slightly so that their entire torso is in the frame.

EXAMPLE STANCE AND CAMERA DISTANCE FOR CALIBRATION

NETWORKING (WEB RTC, WEB SOCKETS FOR SIGNALLING)



USER FLOWS


LANDING PAGE
ABOUT PAGE
SELECT INSTRUMENT CATEGORY
SELECT INSTRUMENT
CONFIGURE INSTRUMENT
USER 1
USER 2
YOUR PEER ID: XXX
"SHARE YOUR ID WITH YOUR JAM BUDDY"
(WAITING)
START SESSION
JOIN SESSION
ENTER YOUR JAM BUDDY'S PEER ID: _____
SUBMIT
SHARE PEER ID (OUTSIDE OF APP)
INSTRUMENT 1
INSTRUMENT 2
SHARED SESSION
ON SUBMIT SESSION OPENS FOR BOTH USERS

COMPONENTS (API SELECTION)

OVERVIEW:

GRAPHICAL INTERFACE P5.JS
SOUND TONE.JS
POSE TRACKING POSENET
P2P & BACK-END PEER.JS, NODE (EXPRESS, REST API)

DETAILS:

GRAPHICAL INTERFACE

The graphical interface has two main components:

1. The wrapper (website), which includes the landing page and settings for the main app. Interaction occurs via traditional mouse and keyboard controls.
2. The main app, which runs in the canvas and is principally controlled through the webcam using gesture tracking.

The website component is built with CSS, JAVASCRIPT and JQUERY.
The main app is principally built with P5.JS.

CONSIDERATIONS

I started prototyping in P5, because it was fast and easy, but considered and ran experiments with a few different APIs for final implementation of the main app. These include:

- ➔ HTML5 CANVAS API
- ➔ PIXI.JS (an open-source 2D WebGL renderer)
  An open-source 2D WebGl renderer.
- ➔ PAPER.js
  An open-source vector graphics scripting framework (uses PaperScript).

I considered using the HTML5 CANVAS API to get more comfortable building from the ground up rather than relying on external libraries. It also integrates easily with getUserMedia() and other APIs that are part of the WebRTC framework.

PIXI.JS is quite comprehensive and uses WebGL, which would allow me to create more impressive graphics. However it comes with a larger payload and I am already concerned about the pose tracking slowing down the app.

PAPER.JS is lightweight and makes it easy to work with vectors. However PaperScript works with its own DOM and has some interoperability issues with the JavaScript DOM.

CONCLUSION

Ultimately I decided to stick with P5 since it meets my basic needs and I'm already familiar with it. I have many other elements to incorporate into this project, which are essential to the core functionality, and felt I should not prioritize learning a new graphics library or API.

SOUND

I am using TONE.JS, a Web Audio framework that makes it easy to generate and manipulate sound in the browser. It's useful for simple projects but offers enough functionality to support more complex creations. TONE.JS permits quantization and provides other features for synchronizing and scheduling events. These features will come in handy when I implement the networked aspects of this project.

POSE TRACKING (GESTURE-BASED INTERFACE)

Pose / gesture tracking is done using POSENET, a machine learning model and API built on TENSORFLOW.JS. The API tracks 17 different keypoints:

0	nose	6	rightShoulder	12	rightHip
1	leftEye	7	leftElbow	13	leftKnee
2	rightEye	8	rightElbow	14	rightKnee
3	leftEar	9	leftWrist	15	leftAnkle
4	rightEar	10	rightWrist	16	rightAnkle
5	leftShoulder	11	leftHip		

P2P & BACK-END

For the real-time interaction I am using PEER.JS, a peer-to-peer API built on top of WebRTC, supporting both data channels and media streams. The API uses WebSockets for signaling. I have integrated it with an Express app in Node. If there is time I will build a REST API to handle persistent state features (saving and loading custom instrument presets).

PROGRESS OVERVIEW

DONE

- ➔ Setup canvas (main app) and stream webcam video to canvas
- ➔ Integrate POSENET tracking

- ➔ Basic smoothing of POSENET keypoints using lerp (also researched Kalman filtering and other algorithms)
- ➔ Test of different graphics APIs (decided on P5)
- ➔ Generate background of twinkling stars
- ➔ Write class for the construction of Sound Objects, including methods to:
    - ◆ Draw objects onto canvas (currently drawn as ellipses, with some basic styling, to test)
    - ◆ Drag to change position on canvas
    - ◆ Compare position of specific POSENET keypoints and play sound (using TONE.JS) when position overlaps with object
- ➔ Write Instrument class to generate simple synth (array of Sound Objects according to a diatonic scale passed as an arg). Includes methods to layout array of Sound Objects within bounding box:
    - ◆ Vertically
    - ◆ Horizontally
    - ◆ Along a bezier curve
- ➔ Test interaction with Sound Objects in different configurations: an 8th note is triggered when wrist keypoint overlaps.
- ➔ Setup PeerServer in Node combined with Express app

TO DO (PRIORITIES HIGHLIGHTED)

- ➔ Finish implementing networking
    - ◆ Add SSL certs
    - ◆ Integrate PEER.JS on client side
- ➔ Landing page and instrument selection/configuration
- ➔ Tweak graphics and instrument layout
- ➔ Include option to record audio output
- ➔ Different gestures for triggering sound
- ➔ Guided instructions / feedback for calibration step
- ➔ User testing (within the next couple of days)

IF THERE'S TIME / FUTURE DEVELOPMENT

- ➔ Include persistent state features to allow users to save custom instrument presets
    - ◆ Database
    - ◆ REST API
- ➔ Implement a more sophisticated smoothing algorithm for pose tracking
- ➔ Instrument search feature