CS 21 CanJam: Initial Design

Roger Burtonpatel, Cece Crumlish, Skylar Gilfeather, Tyler Thompson

Our proposed project, CanJam, is a collaborative music maker. It allows users to “paint” sounds over a colorful canvas to create combined melodies with peers across the web.

Overview

The Project

CanJam is an online, multiplayer melody maker that enables users to play musical beats with each other in real time. Inspired by the Chrome Music Lab Melody Maker, our goal is to build an intuitive, colorful, and whimsical visual interface for users to play with the pitch and timbre (the sharpness and tone) of different instruments. Furthermore, we want to enable remote users, separated by geographical distance, to play notes and create melodies with each other in real time.

To use CanJam, the user downloads and runs the CanJam python package, which spins up a colorful GUI that users can interact with via point-and-click. On startup, a user can choose to create a new canvas, to play music on their own; or, they can specify another existing CanJam user and join the canvas that they’re painting on. Each CanJam user is assigned a unique, 8-character code that identifies their device (corresponding to their IP address under the hood). By joining this other user, the current user is connected to a collaborative canvas with all of the other user’s currently connected collaborators. Uniquely, no one user owns the CanJam canvas: when the original creator leaves, all other users remain connected to each other and can continue playing music together. This decentralized approach frees users to dynamically form collaborative jam clusters, and has the potential to help CanJam scale performantly to a large user base.

In the collaborative canvas, the user is presented with a minimal, two- dimensional white grid of tiles. Each user is assigned a particular instrument and color; and, each tile on their individual view of the canvas represents that instrument being played at a given pitch and timbre. Timbre varies across the horizontal axis, and pitch varies across the horizontal axis. When a user clicks and holds their mouse over a particular tile, the unique sound is played for the duration of the mouse’s contact on the tile, and the tile itself glows with the user’s color. Within a connected CanJam cluster, user’s “painting” actions over the canvas are reflected onto every other user’s canvas: so, all users hear the sounds and see the trails of color left by their collaborators.

Design Decisions

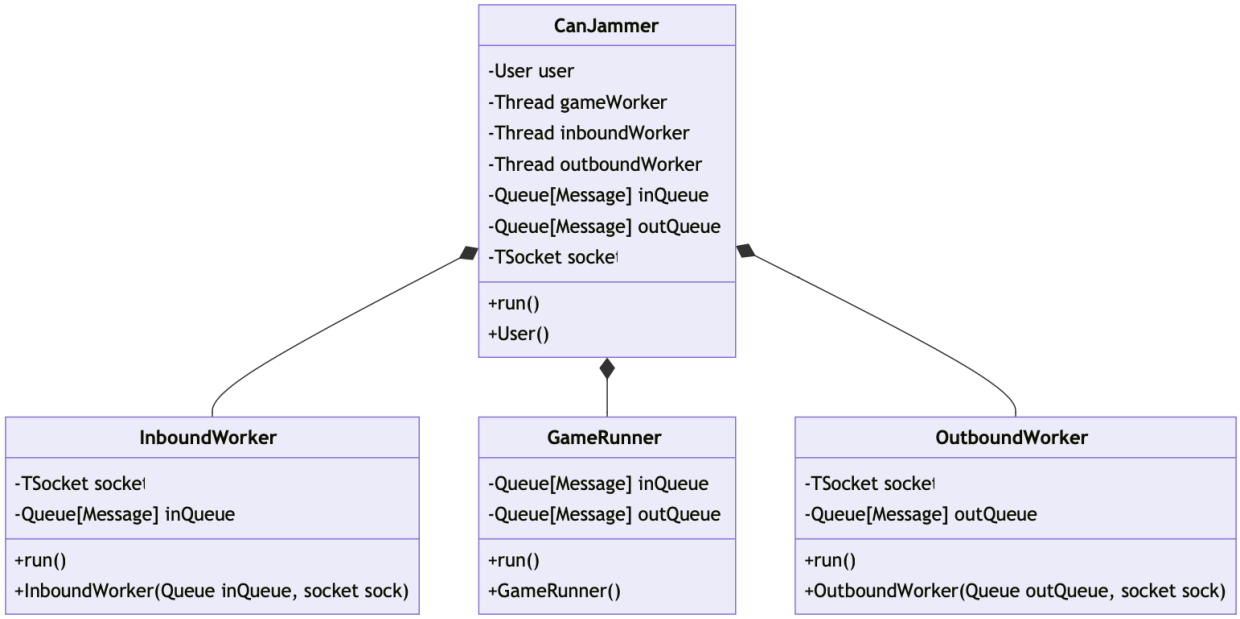
Throughout the design process, we explored several ways in which our user should generate sounds. Initially, we considered constraining users to metered music, similar to the Chrome Melody Maker. In this design, users would place notes at fixed points on an advancing musical “timeline” with a fixed tempo; then, wait to listen to their notes play out with others. However, we felt that this constrained users from spontaneously “painting” sounds across the canvas and hearing their music in real time. Alternatively, we also considered creating a music player that would translate a user’s command executions on a shared Linux server, such as Halligan, into musical notes. While this would be an interesting visualization of server activity, we decided that it would be more enjoyable for users to use a creative, interactive program. Ultimately, we opted to allow users to freely play sounds in freeform time, making the canvas an experimental, free-form space rather than a space to compose structured melodies.

Program Architecture

The CanJam architecture consists of three primary components, each running on a unique thread: the **GameRunner**, **InboundWorker**, and **OutboundWorker**. These components communicate by producing and consuming **Message** objects via shared thread-safe queues. These threads are managed by a driving **CanJammer** component, which maintains a shared UDP socket for the **InboundWorker** and **OutboundWorker**.

Class Diagrams

Modules



CanJammer: Driver component that facilitates communication between the InboundWorker, GameRunner, and OutboundWorker component threads.

- ➔ Maintains shared queues of **Message**s.
- ➔ Provides thread-safe manipulation of the **Message Queue**s. The **Queue** is Python's synchronized FIFO queue, **queue.Queue**.

Inbound Worker: Communication component that receives incoming packets from other User peers on the shared **TSocket**., and deserializes them into **Message** objects

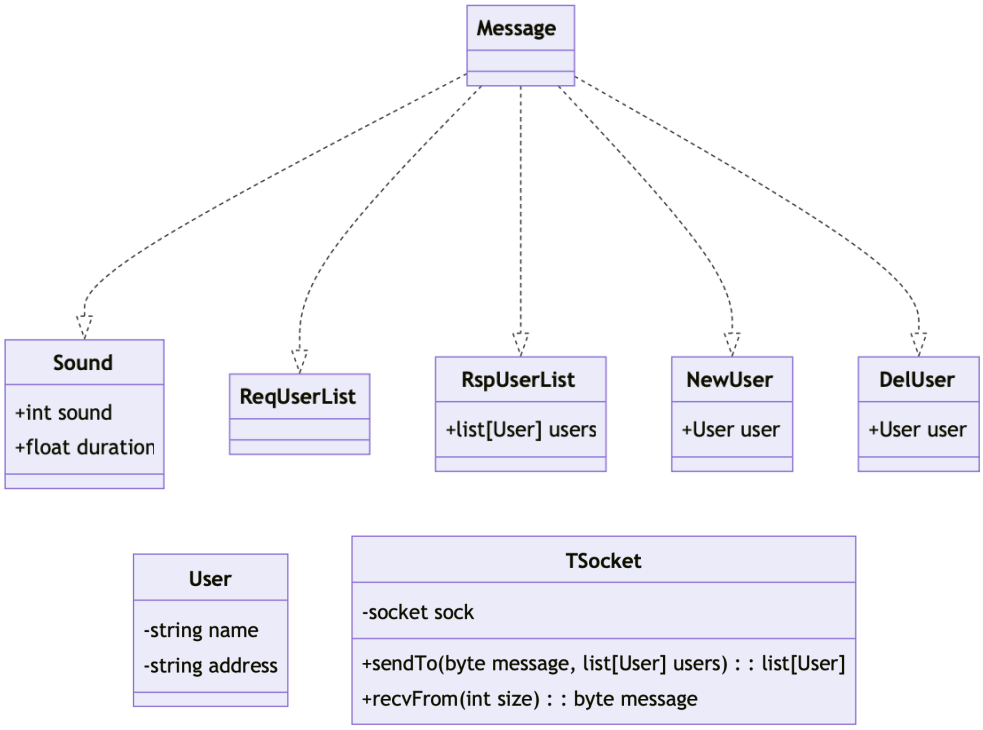
- ➔ Puts **Sound Message**s on the inQueue for the **GameRunner** to process and play.
- ➔ Handles **NewUser** and **DelUser** packets *directly* by updating internal **UserList**, and putting the message on the outQueue for the OutboundWorker to propagate / broadcast to all of the connected **User** peers.

GameRunner: Functional game component that runs the **Pygame** GUI and handles user input events.

OutboundWorker: Communication component that serializes **Messages** into packets, and broadcasts them to all connected User peers on the shared **TSocket**.

- ➔ Consumes **Sound Message**s from the outQueue and broadcasts them to all connected User peers.
- ➔ Handles **NewUser** and **DelUser** packets *directly* by updating internal **UserList**, and putting the message on the outQueue for the OutboundWorker to propagate / broadcast to all of the connected **User** peers.

Data Types



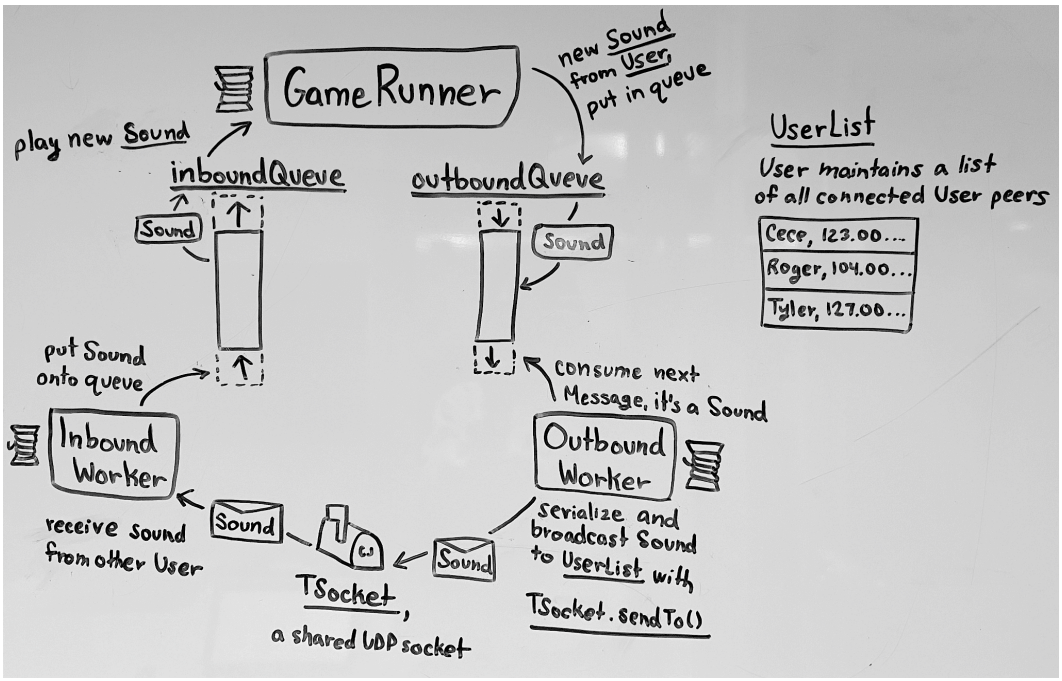
User: Represents a CanJam user, by their name and unique IP address.

TSocket: A monitor wrapper for Python's socket.socket, which provides a thread-safe shared UDP socket for sending Messages.

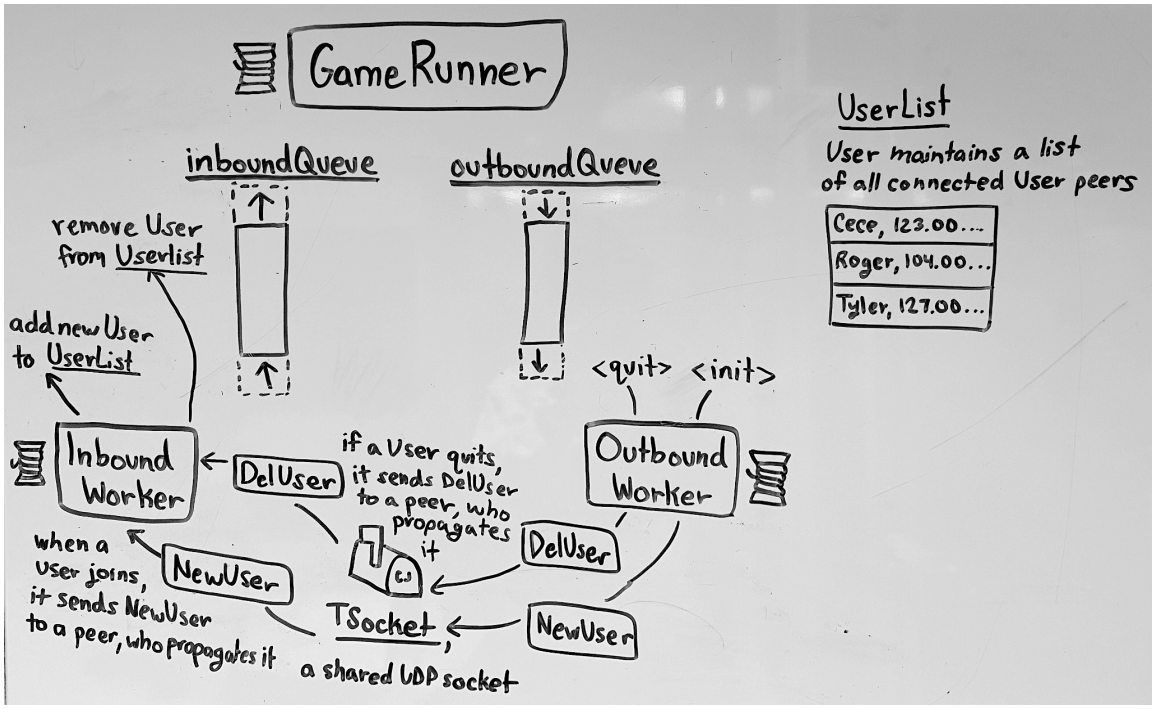
Message: Sound | ReqUserList | RspUserList | NewUser | DelUser

Object Diagrams

Sound Data Flow



Distributed User Connection Flow



Packages and PoC

We will be developing CanJam fully in Python, using a variety of libraries to implement network communications, render and animate the canvas GUI, and enforce safe communications and shared data manipulation between threads.

Important Python Libraries

threading: Thread creation and synchronization primitives, for thread-safe Queue and TSocket manipulation.
sockets: Low-level UDP socket creation, for CanJammer to send and receive Message datagrams from CanJam instances at any given IP.
queue: Main

External Packages

pygame: Main game engine in GameRunner, used to animate the GameRunner canvas and handle user input actions.

Docs: https://www.pygame.org, version 2.5.2.

fluidsynth: Synth library that supports soundfonts and synth channels. Used in GameRunner to generate variety of musical sounds.

Docs: https://www.fluidsynth.org, version 2.3.4.

pyfluidsynth: Fluidsynth bindings for **fluidsynth** to use.

Docs: https://github.com/nwhitehead/pyfluidsynth, version 0.2.

numpy: Math processing library required for pyfluidsynth's frames.

Docs: https://numpy.org, version 1.26.0.

pyaudio: Audio library to create audio channels for **pyfluidsynth** to use.

Docs: https://pypi.org/project/PyAudio/, version 0.2.14.

Boilerplate and PoC

Our project is hosted on GitHub at the official CanJam Repo. There, the demo folder contains fluid-test.py, a sample program that modulates and plays out synth sounds using the **fluidsynth** library.

Development Timeline

Our team will meet on Sundays from 3:00pm to 4:00pm in the JCC. We'll check in with each other asynchronously about progress on Fridays, and get started on the next week's tasks during the following Sunday meeting.

Timeline

1. **Module, Class, Method, Stubs and Final Modified Architecture**
    a. Fri March 29th → Fri April 5th
2. **Module Implementations: GameRunner, InboundWorker / OutboundWorker, Network Interface**
    a. Fri April 5th → Fri April 12th
3. **CanJammer (Driver), End-to-End, Plumbing, Finish Rough MVP**
    a. Fri April 12th → Fri April 19th
4. **Testing and Touchup**
    a. Fri April 19th → Mon April 22nd
5. **Final Codebase Submission, Presentations**
    a. Mon April 22nd (TBD Presentation Date)
6. **Final Report, Reflections**
    a. Mon April 22nd → Mon April 29th

D1: Music Player GUI

Steps

1. Basic Sound Modulation / Playback via PyAudio
2. Canvas: Single Tile, Mouse-Over Activation and Playback
3. Canvas: Multiple Tile GUI, Activation and Playback
4. Start Screen (User Configuration), Canvas Options Menu (Exit)
5. Plumbing to CanJammer Message Queues for Producing / Consuming Messages

MVP

- ➔ Piano-style, horizontal axis of sound tiles varying in pitch (no vertical volume or timbre axis).
- ➔ Single instrument sound type available.
- ➔ Users are assigned a default color, and a user's active tile is highlighted with a translucent layer of their color. Two users on the same tile produce a mixed color!
- ➔ Users play peer sounds as soon as they arrive: no conception of preserving accurate time delay independent from network delay.

D2: Shared Message Producer / Consumer Work Queues

Steps

1. InboundWorker, OutboundWork interaction with shared queue.QUeue object, functionality testing
2. (Also for D3): Basic TSocket Implementation, Serialization / Deserialization of packets into Messages
3. Message Consumption and Processing from TSocket

MVP

D3: Distributed, Peer-to-Peer User Cluster System

Steps

1. (Also for D2): Basic TSocket Implementation, Thread Safety
2. Basic TSocket Implementation, Thread Safety and
4. Message Consumption and Processing from TSocket

MVP

Anticipated Challenges

Integrating FluidSynth Into the Python Build System:

Although there's a variety of ways to get noise to work in pygame midi and not midi, most are either a bit dated or require funds (tragically). Due to this, we will be using fluidsynth a program for digital synthesizers as it supports soundfont, and thus many digital instruments Given that fluidsynth is not python we will also be using the pyfluidsynth package, for fluidsynth bindings. To get our synths and sounds. For a demo see fluid-test.py examples.

One challenge we have already faced is library and version difficulty with correctly importing the pyfluidsynth library and fluidsynth, a tragic bug we encountered was getting both to work in a conda virtual environment, we referenced these we will be using to play noises for the game. After referencing https://github.com/nwhitehead/pyfluidsynth/issues/40, for solutions failed, we opted to try just installing both onto our computers sans venv's and it works that way. Ultimately we hope to get the venv thing figured out for this as it is so annoying for linking, but alas, this is the best we can manage.