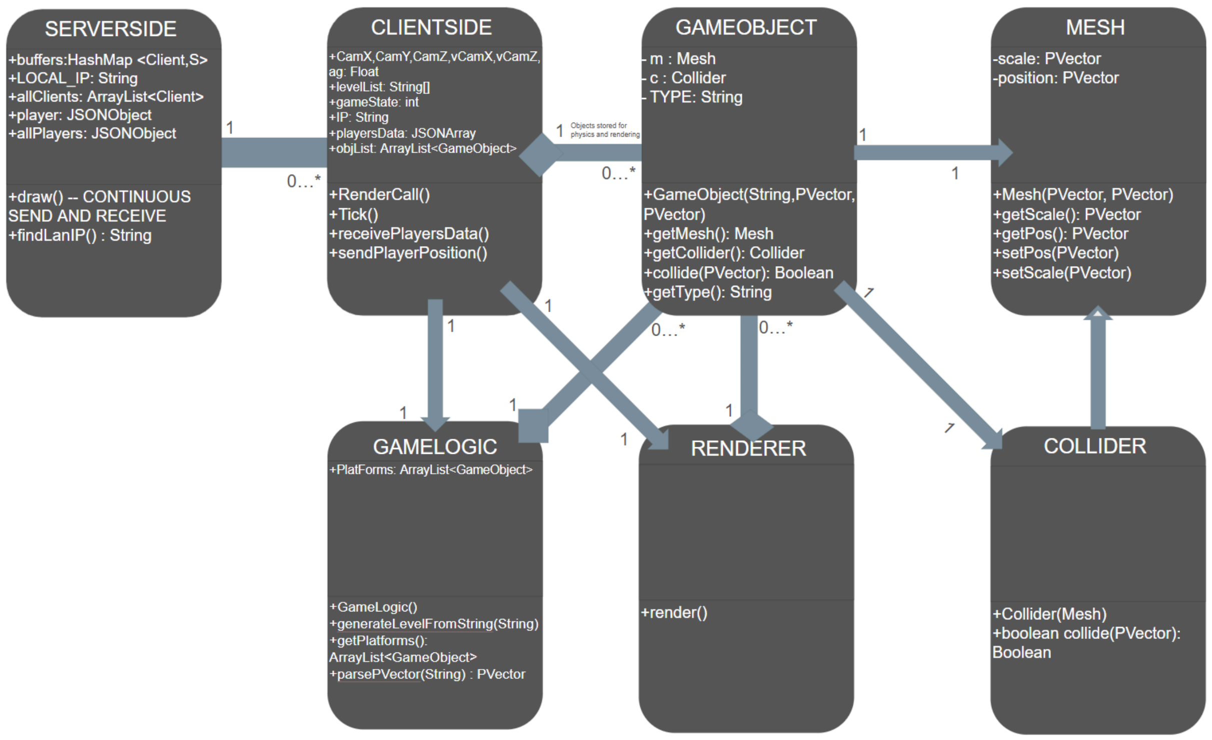
Period 2
Leo Titievsky
Certified Java Master
The BEST game

Short Description

I made a 3D multiplayer parkour/puzzle game. The game leverages Processing's Network, PVector, JSON, and P3D libraries to achieve these functionalities. Additionally, some Java Network as well as the robot library are used to facilitate the user experience. The game comes with a serverside and clientside program, and is designed for local and LAN use. As processing's P3D renderer is rather rudimentary, a large volume of the technical work of this project will go into making it into a usable game engine.

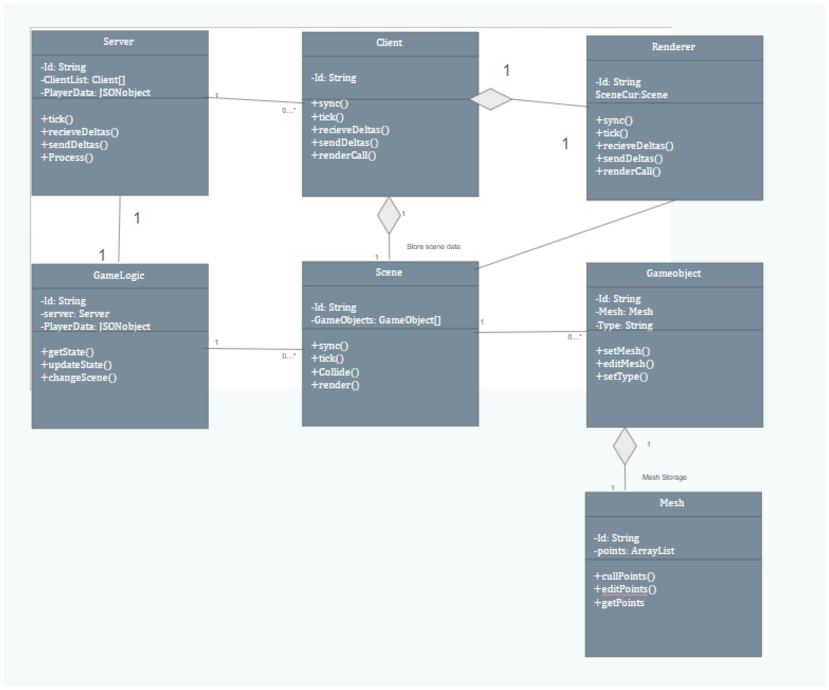
Further details on the project further down…

FINAL UML DIAGRAM



Methods with no specified return are voids. Manyminor/redundant methods and fields are left out

LEGACY UML Diagram



User Guide

FOR OFFLINE USE:

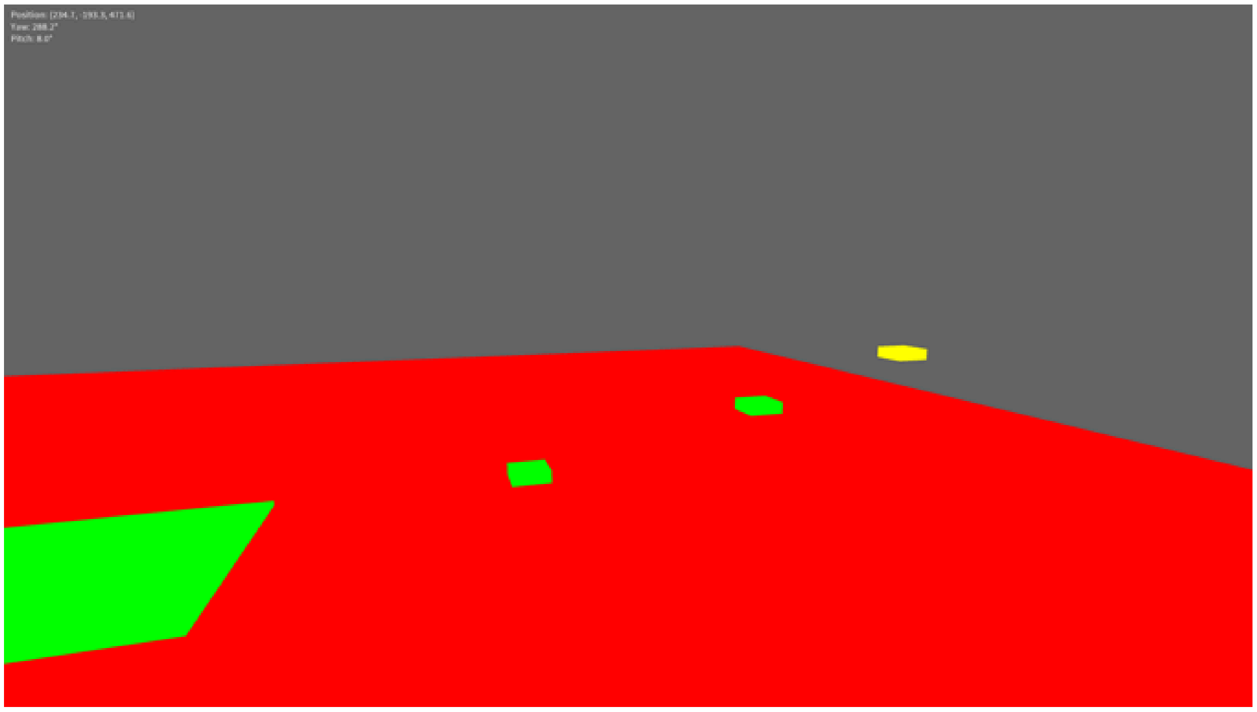
Launch the clientSide main script. Press enter to bypass the connection menu. Use the W, A, S, and D keys to move, and the space key to jump (you should have caps lock disabled). You will advance to the next level if you reach the yellow goal tiles. You will be reset if you hit a red object. Proceed through the levels as you can.

FOR ONLINE USE (LOCAL NETWORK ONLY)

Instantiate a server. Using either the written-in IP debug, or some other ipconfig tool, determine the ip of the machine your server is running on. Launch the client. Enter the IP when prompted, then press enter. Play proceeds normally from here, with the exception that as many people can join the game on the local network as possible. Players can ride each other like mobile platforms, and are encouraged to utilize this feature as a problem solving tool. A fellow player will appear as a purple moving platform on your screen.

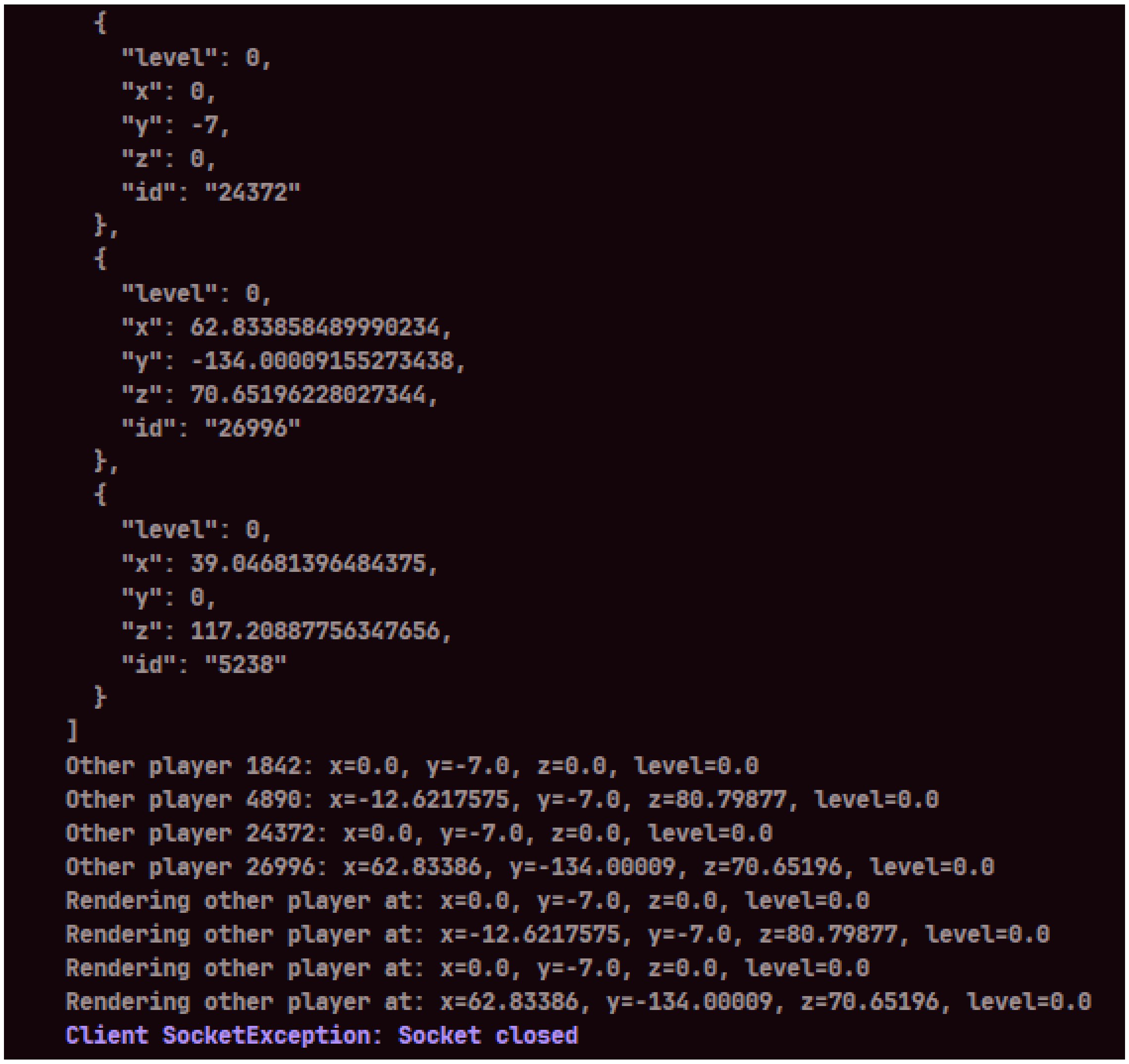
Please follow these instructions exactly -- failure to properly initialize the server before launching the game or failure to properly configure the ip has resulted in some nasty crashes.

GAME ENGINE
Major changes are upon the best Java game ever! Rendering has been reworked, as well as collisions, now utilizing Axis-Aligned bounding boxes to represent each mesh. While spartan on the surface, composite meshes can be formed of virtually infinite resolution. Collision, in short, works by projecting a point, adjusted by an offset, to projections of each side of the collider box, and checking if it is inside each of them. This system is also very optimized for a simple processing game like this one, thus it can handle 120 network ticks and 60 physics ticks in a second with minimal stuttering, with multiple moving players -- a massive bound over previous versions stuttering, largely in part due to the efficient legacy renderer from the voxel game. The game engine is probably the crowning achievement of this project, as it stands on its own from the game and can be repurposed in its own right for many 3D projects.


Test capture, offline.

NETWORK

The network system, although the same in structure, has been rewritten. As always, the server maintains a JSON object to which clients read and write relevant fields. Getting everything to sync smoothly was a nightmare, but those issues have been resolved to my knowledge.



```
    {
      "level": 0,
      "x": 0,
      "y": -7,
      "z": 0,
      "id": "24372"
    },
    {
      "level": 0,
      "x": 62.833858489990234,
      "y": -134.00009155273438,
      "z": 70.65196228027344,
      "id": "26996"
    },
    {
      "level": 0,
      "x": 39.04681396484375,
      "y": 0,
      "z": 117.20887756347656,
      "id": "5238"
    }
  ]
  Other player 1842: x=0.0, y=-7.0, z=0.0, level=0.0
  Other player 4890: x=-12.6217575, y=-7.0, z=80.79877, level=0.0
  Other player 24372: x=0.0, y=-7.0, z=0.0, level=0.0
  Other player 26996: x=62.83386, y=-134.00009, z=70.65196, level=0.0
  Rendering other player at: x=0.0, y=-7.0, z=0.0, level=0.0
  Rendering other player at: x=-12.6217575, y=-7.0, z=80.79877, level=0.0
  Rendering other player at: x=0.0, y=-7.0, z=0.0, level=0.0
  Rendering other player at: x=62.83386, y=-134.00009, z=70.65196, level=0.0
  Client SocketException: Socket closed
```

Beautiful client network debug data, display received and sent JSON data and other player data

GAMEPLAY

The gameplay has been on a rollercoaster lately, with many near additions. Perhaps one of the most costly misadventures: server maintained conditional platforms, in which players could interact with levers (through a ray cast system) to globally enable or disable GameObjects. Ultimately, this worked locally but required a massive increase in server complexity, with the final prototype of this system being twice as large as the final server code. Unfortunately, the server side code never worked entirely, and this was scrapped for the alternative method of collaboration -- multiplayer physics. Instead of opening doors for each other, players now have the option to boost their compatriots' jumps. Considering that if one player advances, the whole server does, this is a valid strategy for collaboration. You can also block and kill your other players if you feel evil.

ISSUES AND STRUGGLES

The biggest issue of this project, overall, is narrowing the scope and understanding my limits. In the end, I have this massive technical project with bare visuals and levels that took a ton of time. I fear it may have been wiser to settle on something more fun and easier.

More specifically, syncing server-client interactions was the greatest challenge, forcing me to abandon some awesome features (like conditional platforms/buttons)

LOG

I am solo and everything is done by me.

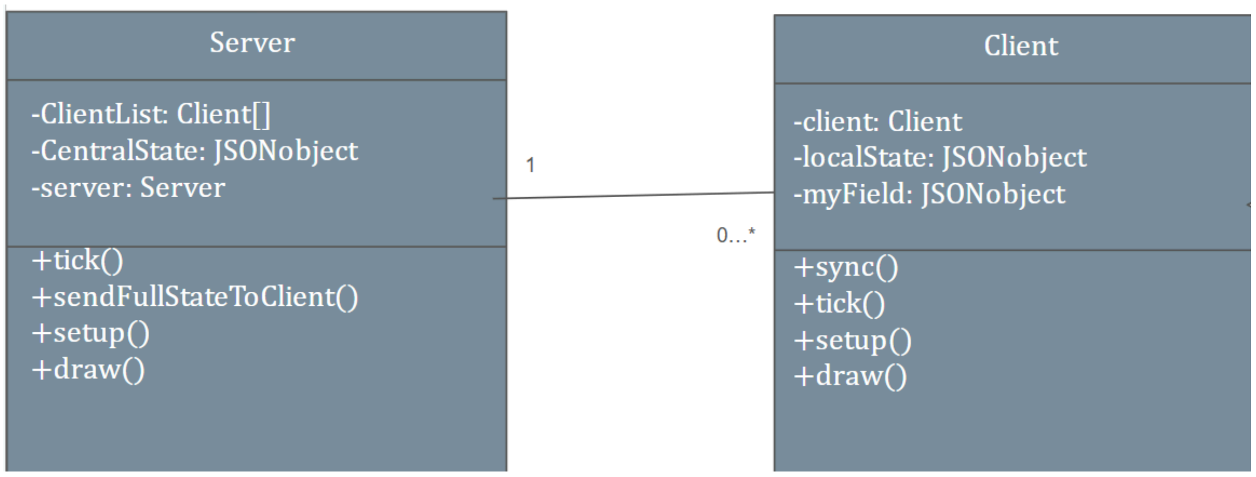
LEGACY INFO FROM PREVIOUS EDITIONS

Functionalities/Issues
WEEK OF 5/27
Current functionalities on main: exchange json between client and server.
Current functionalities on game engine branch: fully functional mesh renderer running a minecraft style voxel demo "game"

Issues: server support limited to 1 client. Not consistent/fast enough to support smooth multiplayer gameplay. Voxel game needs to be erased and the core renderer needs to be stripped out. Gamelogic and levels need to be created. The game engine and networking need to be integrated

Changes to the UML:

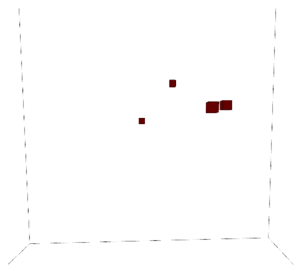
The server/client relationship has been updated:



Since each is run independently, they maintain their own draw and setup methods. The deltas system has been scrapped for a simpler system of assigning a json field to each client to maintain its state, then using the server to distribute the global state + the state of each player to each client. Since each of these will be a driver for any aggregate classes, minimal methods are used as future additions will delegate more and more to other classes called through server/client tick methods. If performance in later versions is poor, the more optimal deltas system will return, but for now I can not find a method to make it work well with processings network setup.

WEEK OF 6/3
Current functionalities include previous network capabilities as well as a new game engine implementation - this is complete with a camera/player controller, some example meshes, a placeholder raycasting system, some basic physics, and some basic collision. Network and game engine features are not currently combined yet, and exist as separate scripts.

A test rendering scene from the current version of the game engine



Issues: aside from the lack of integration between the game engine and the networking script, the main issue lies in the lack of performance seen with the renderer - it sees long start up times and after 1 or 2 thousand polygons it begins to lack. This will hopefully be solved soon with a series of occlusion algorithms to selectively draw points based on camera direction and position relative to other points. The camera script was previously bugged, but redoing the math that let to it corrected this.

By next meeting, all aspects currently existing shall be integrated, a series of performance algorithms shall be implemented, and a game logic system with playable levels should be implemented as well.