Cat Golf Devlog

# 29/05/2023 – cat senses

We have figured out how we will implement cat senses.

Each cat will have a list of objects that it has in its notice. These are the things it has noticed are there. We have developed the system that the cats use to notice things. We will have a sensory update, where the notice calculations are run again after the delay is done.

A picture containing drawing, child art, circle, sketch

Description automatically generatedThis is a quick graphic of the cat’s senses. To start, vision. Cats have two types of vision: peripheral and main. If something is in its main vision, it will detect it based on sight alone. If its in peripheral vision, the sight detection probability will be multiplied by a function of the objects speed. Cats also detect sound all around them.

These calculations give a percentage chance that the cat will notice the object. Because of the consistent sensory update timing, even quite low chances are actually quite frequent. If the update happens every seconds, then it only takes 5 seconds for the cat to probably notice something with a 20% chance of getting noticed.

Cats have individual sense parameters that mess with the functions we have for calculating these chances. These parameters are:

certainSight: the sight distance where a cat is certain to notice an object

maxSight: the maximum sight a cat can see something

certainSpeed: the minimum speed of an object where speed doesn’t affect the percentage calculation any more.

Hearing: something or other.

Here is the sight distance calculation:

Here’s the velocity calculation:

Here’s the sound calculation. Note here that this uses the sound’s intensity as a part in the formula. The idea is that a cat’s hearing can reduce the intensity of a noise linearly.

In order to simulate things like sound occlusion, instead of developing a complex sound system our idea is that if there is an object in-between the sound source and the cat, we remove a constant from the intensity of the sound.

If an object is in the cat’s primary vision arc, the narrow one, then we use the distance calculation to determine the probability it will be seen. If its in the cat’s peripheral vision, then we multiply the distance calculation by the velocity calculation.

For sound, when a sound event is made it runs the calculation for all cats.

These fancy curves probably won’t matter when we actually do the game. Things will either get noticed within a boundary or they won’t. Anyway, that’s all.

# Cat Animation

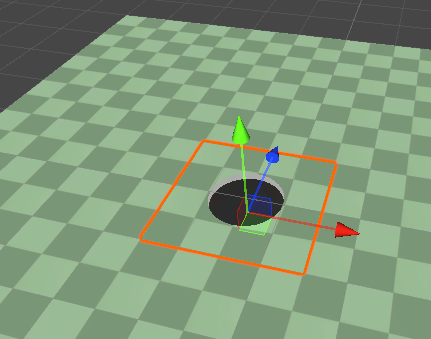
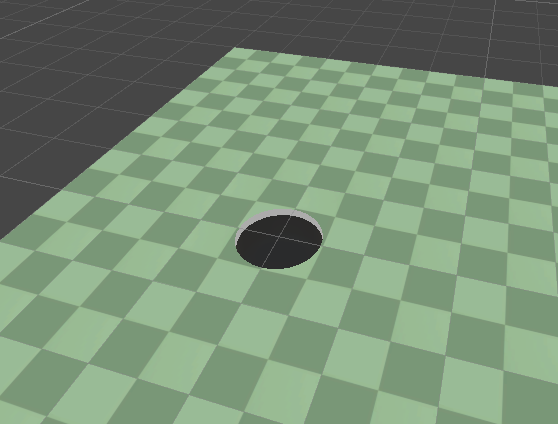
We have decided to handle the cat animations with in-game scripts rather than set animations. This is because its impossible amount of workload for us to have an animation for every turn, every look, every walk, and so forth.

The current model and rig for the cat is also going to be replaced for a more realistic one with useful bones for this style of animation.

There’s not much to say here, but the generic idea is to use inverse kinematic animation so that the body realistically follows the feet and the head.

There’s a lot more research to be done into procedurally handling cat animations, so these are mostly the early ideas. These animations are going to be intertwined with the game so much that we won’t be able to work on them until a decent portion of it has been made gameplay wise.

# Golf Cup



By using unity’s render feature to change the render order the cup can be inserted into the ground anywhere visually, without changing the ground mesh to prevent “z fighting”. The collision of each level will be made separately from the visual using primitive shapes as that’s more reliable for gameplay purposes.

# Networking

Basic network code has been written for connecting and sending data using TCP. Now to abstract the sending and receiving process, letter objects will be used to send formatted data easily. The letter object will have a header specifying how many more bytes there’s to come, so the receiver will wait for all the bytes to arrive before processing. The content will start with a byte specifying its content type. Each type will have its handler stored in a dictionary with the type enum as the key and the handler as the value.

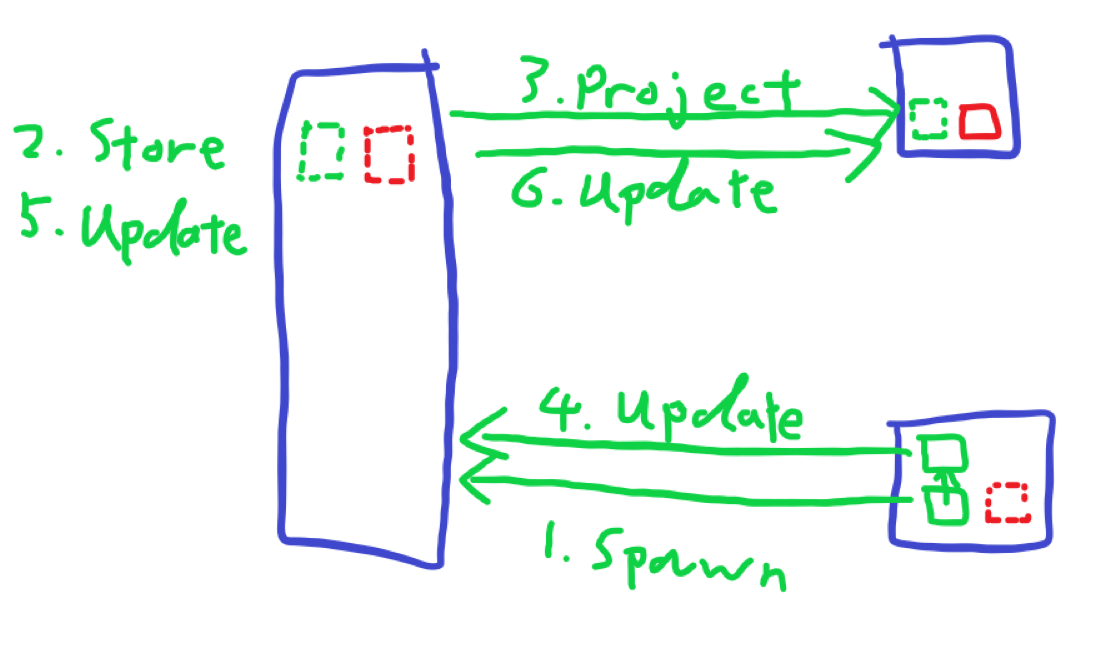
Furthermore, for performance the letter objects will be pooled to reduce instancing.

# Client-Server Interactions

Currently, the architecture works like this. There’s the client class *Resident*, server class *Post Office* and socket TCP wrapper class *Postbox*. The connection code execution path is as followed: Server starts, binds to port 25569 and listens for connections on its server socket; Client starts and when player click connect it sends a connect request via its postbox to the server; When the server accepts the request, generate a UID for the client during this session and create a client record object *ResidentRecord* to store its details and socket; The server then sends a packet to the client to signify connection successful and alongside the client’s UID; The Client when received its UID sends a packet containing its username; The server then stores the username in the client’s resident record.

To detection disconnection, every frame when the socket is polled its checked that its still connected; when the server detects the client is no longer connects it closes the socket and removes the client’s records. Or when the client detects its no longer connected it clears its record and socket too, but also invokes the *onDisconnect* event to signify UI elements or other systems to reset back to menu.

# Object Synchronisation

The object synchronisation will follow the chart’s steps. The objects to be synchronised will be represented with a *Hologram* object it will store the data that object need to be synchronised, also it will handle serialising and formatting the data into bytes to be sent by a *Hologram Transmitter*. Which will be received by the server and stored and used by a new client that connects to catch up with the rest. Then the server will pass on the *Hologram Data* to the other clients apart from the one that sent them, so they can project the data over their mirror of the actual object. Interpolation might have to be done for motion data in the projector. Extrapolation is also on the scope if needed with the help of numerical integration methods.

## Updated Plan

The plan is roughly the same with the exception of two new classes: Hologram Database and Hologram System. Hologram Database is server side and each town gets an instance, it caches object data for the new players who joins the game midway through. Hologram System is a client side singleton that orchestrate and handles object synchronisation packets that would’ve otherwise been handled by the transmitters themselves, this should make debugging easier and it would be easier to manage which transmitter sends on which frame to lower bandwidth usage.