Technical Development Document

Meta Spark AR - Gamified use of face tracking

[**Project Overview 1**](#_rvsz6g6ynsfp)

[Game title 1](#_50w4s46kwa4v)

[Team 2](#_ulh3oppmbp9f)

[Environment 2](#_cwzcdt5c1dzg)

[Alt. 2](#_oqcqsbceqpte)

[**Project Details 2**](#_526innzjkwc)

[Introduction 2](#_v5bl22fofnfm)

[Purpose 3](#_w4o3jbgu61gg)

[**Technical Architecture 3**](#_66fyf6jxbxlc)

[Flowchart 3](#_p6trs8c1ghab)

[State Management 3](#_7wqedg34ho0g)

[Scripts & Patches 4](#_d230eg368h8m)

[Additional Notes 9](#_u9mdzuc8tvwr)

[**Development Details 9**](#_sh6dune9ddsd)

[Current Tasks 9](#_8esl8fh6fwh9)

[Bugs & Issues 9](#_wqa61fso0dx8)

[**Test Plan 10**](#_icibbt57xdo0)

[Test Strategy 10](#_b41k2opvokvo)

[Test Environment 11](#_it22p8ecujqi)

[Test Cases 11](#_nn765e4l63rs)

[**Asset Management 11**](#_b122i8t2afbt)

[Art Assets 12](#_s0dt82bic267)

[**Appendices 12**](#_rbdgmpaj7kv7)

# Project Overview

## Game title

Step Up Right

## Team

Supervisor: Thomas Koh

Developer: Winston Chiu

## Environment

Meta Spark Studio -> Singleplayer Testing with Meta Spark Mobile Player

Visual Studio Code -> JavaScript & Reactive Code

Plugins: <https://spark.meta.com/learn/scripting/vs-code-extension>

Github: ARISENTU/meta-spark-Step-up-right

Note: Learn about reactive programming in order to continue with the project

<https://spark.meta.com/learn/scripting/reactive-programming>

<https://spark.meta.com/learn/patch-editor/bridging>

Note the project export file types used: <https://spark.meta.com/learn/articles/fundamentals/project-file-formats>

### Alt.

Meta sparks studio & meta spark ar is a software developed by meta as part of their social media engagement software suite. This software was made to create facebook, instagram & messenger compatible filters, en masse. This was then published for free for the general public to access & create their own filters, either personal, profit driven or engagement. Due to the need for ease of access & fast learning, it has 2 main methods of development, script based for developers & patch editor for visual scripting. They are also interchangeable & linked together via its accompanying plugin. It is for this reason that games & gamified filters can be created, & by many people nonetheless. With such attention & interest, Meta has released official [documentation & guides](https://spark.meta.com/learn/). These effects are compatible with a large audience & can be published to the meta servers under SkyLight. This means that publishing & deployment is easy & accessible by nearly everyone.

# Project Details

## Introduction

Step Up Right:

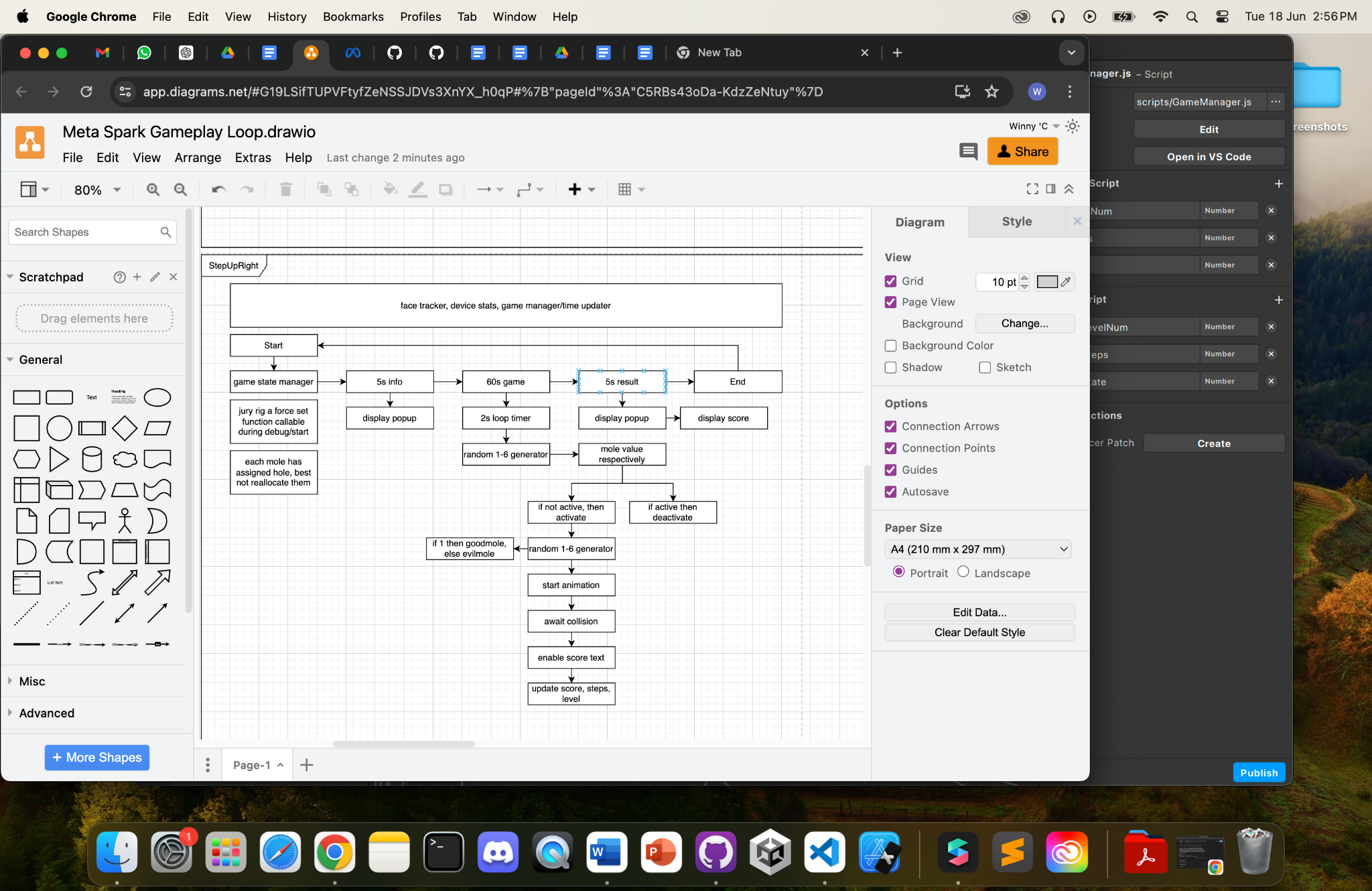
A short game about hitting evil moles with a hammer. Over time, moles will appear from holes around the screen. Good moles are cute & should not be hit, hitting them will lose score. Evil moles appear more frequently, hit them for more score. Interaction with the game relies on face tracking, specifically the nose, at the center of the face.

## Purpose

The project is a gamified filter with face tracking capabilities. The target audience is for all ages, with focus on the elderly interacting with the youth in video calls. This means that a software compatible with & easily published within a video call service is preferable.

# Technical Architecture

## Flowchart



## State Management

Effect start

State 0: Info, 5s, wait for user tap

State 1: Game, 60s

State 2: Result, 5s, wait for user tap

Effect ends

Generalized state change and detection via gamemanager.js. With no switch case suiting the needs of individual signal patches with primary boolean focus, opted to use comparison in an if-else context. Each comparison sends a boolean compared to a timer. This allows for a timed state change as well as repeating loop. Can adapt the timer in any state to be compared to screen taps or button presses.

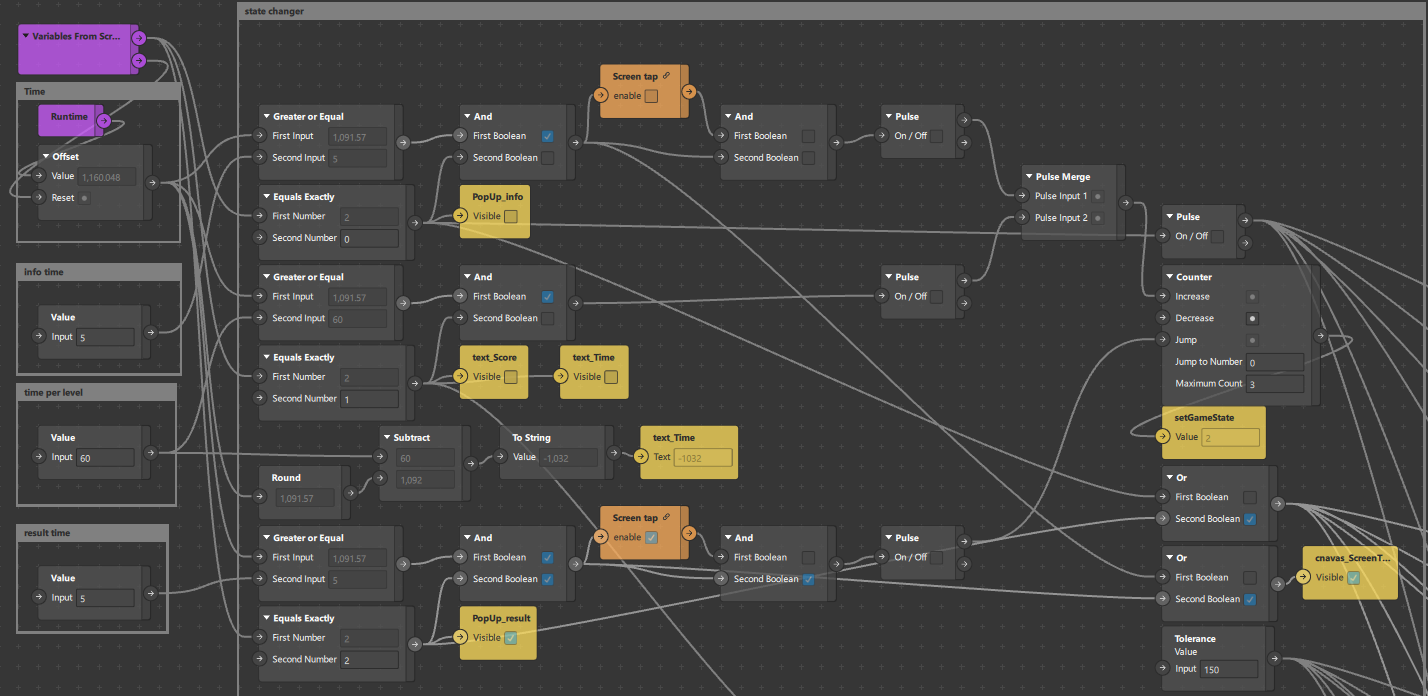
## Scripts & Patches

GameManager.Js

| **import** Reactive **from** 'Reactive'; *//handles react code structure* **import** Patches **from** 'Patches'; *//links with meta spark studio patches* **import** Diagnostics **from** 'Diagnostics'; *//console log and debug features* **import** Time **from** 'Time'; *//loops, time reference and runtime allowance*  *//initialize const variables* **const** Scene = require('Scene'); *//requests scene hierarchy* **const** targetFrames = 20;  *//init logic values from patch* **let** currState = 0; *//compare internal script state to the patch state, if different* **let** s\_gameState = Reactive.val(0);  **async** **function** **getGameState**() {  s\_gameState = (Patches.outputs.getScalarOrFallback('setGameState'));  *//if currstate not gamestate then send a pulse to the patch*  **if** (s\_gameState.pinLastValue() != currState) {  currState = s\_gameState.pinLastValue();  **await** Patches.inputs.setPulse('p\_ResetTime', Reactive.once()); *// Send a pulse to indicate the state change*  Diagnostics.log(`Game State: ${currState}`);  }  *//output the state regardless*  **await** Patches.inputs.setScalar('s\_GameState', s\_gameState.pinLastValue()); }  *// General encapsulation of the various functions required to run every frame* **function** **update**(deltaTime) { *// Main update function called every frame, does not run directly*   getGameState(); }  *//proper loop update* *// Set up a loop to call the update function every frame* **const** timeInterval = 1 / targetFrames; Time.setInterval(() => {  update(timeInterval); }, timeInterval \* 1000); *// Convert to milliseconds* |
| --- |

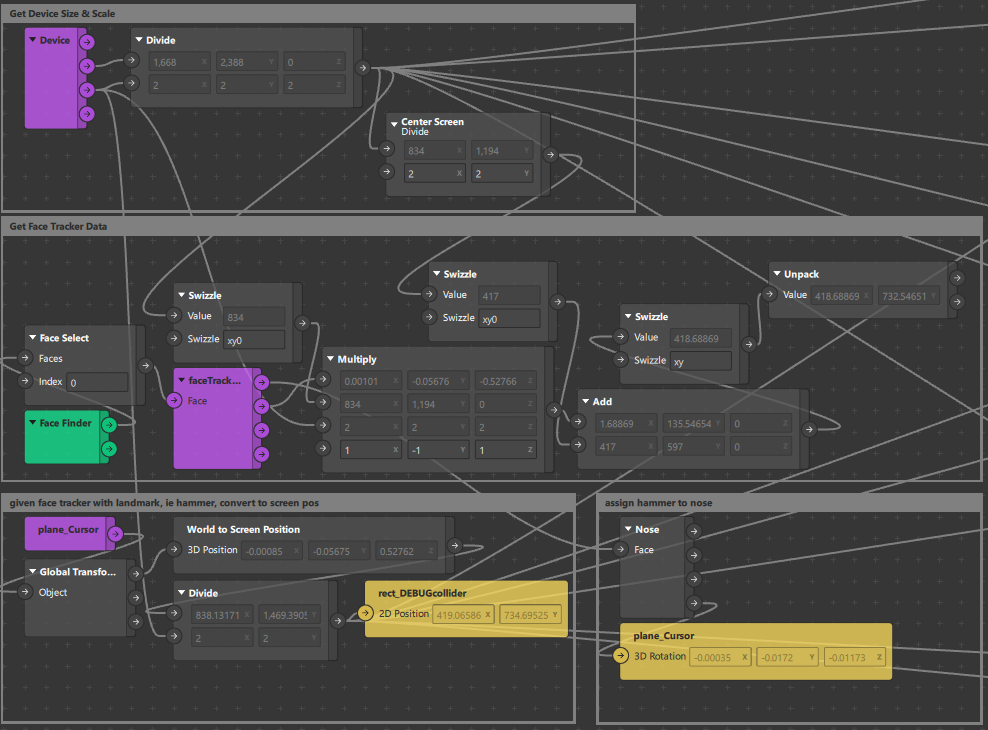
getGameState is a function that compares the internal value of gamestate to the external value found in the patch editor. If the values are not the same, it means that the value has been updated, and thus sends a single pulse to reset time. This means that each state has a maximum timer or an external condition to trigger state change. This allows for consistent, looping gameplay.

Game state manager:



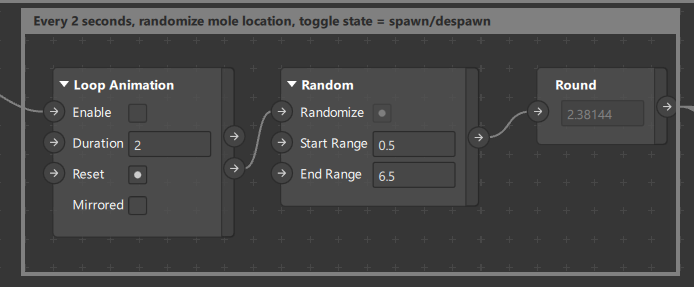
This handles game state with 2 requirements, runtime with offset to track the duration of each state and an optional screen tap to transition. This means that each state has a minimum duration and the user can be prepared to trigger the next state consciously. This allows for more preparation time and easier syncing of gameplay during competition scenarios. Each state is transitioned using a counter and a final reset trigger meaning a finite state machine with predominant loops. Additional states can be added easily.

Screen size, face tracker and global coordinates:



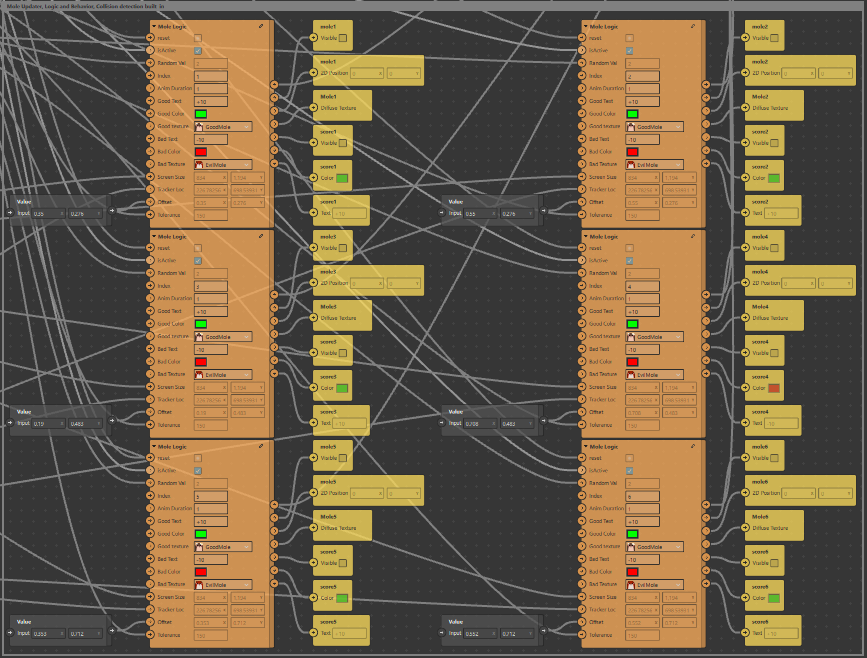
This requests the screen size of the device, then divides by the screen scale, this is done to remove antialiasing and smoothing of large screen devices such as ipads. Smaller devices such as iphones would have a screen scale of 1. Next is to assign the cursor on a facial landmark, in this case the nose. Next we convert the cursor’s location to screen space instead of on the face, flattening the 3d position to a 2d transform. This gives us the tracker location used for collision detection below.

Random Spawner:



This code only activates during gameplay, where every 2 seconds, it selects a random value from 1 to 6. This value TOGGLES the mole’s state. If the mole is not active, it triggers spawning animation and awaits collision. If the mole is active, it disables it, resets the animation and disables collision.

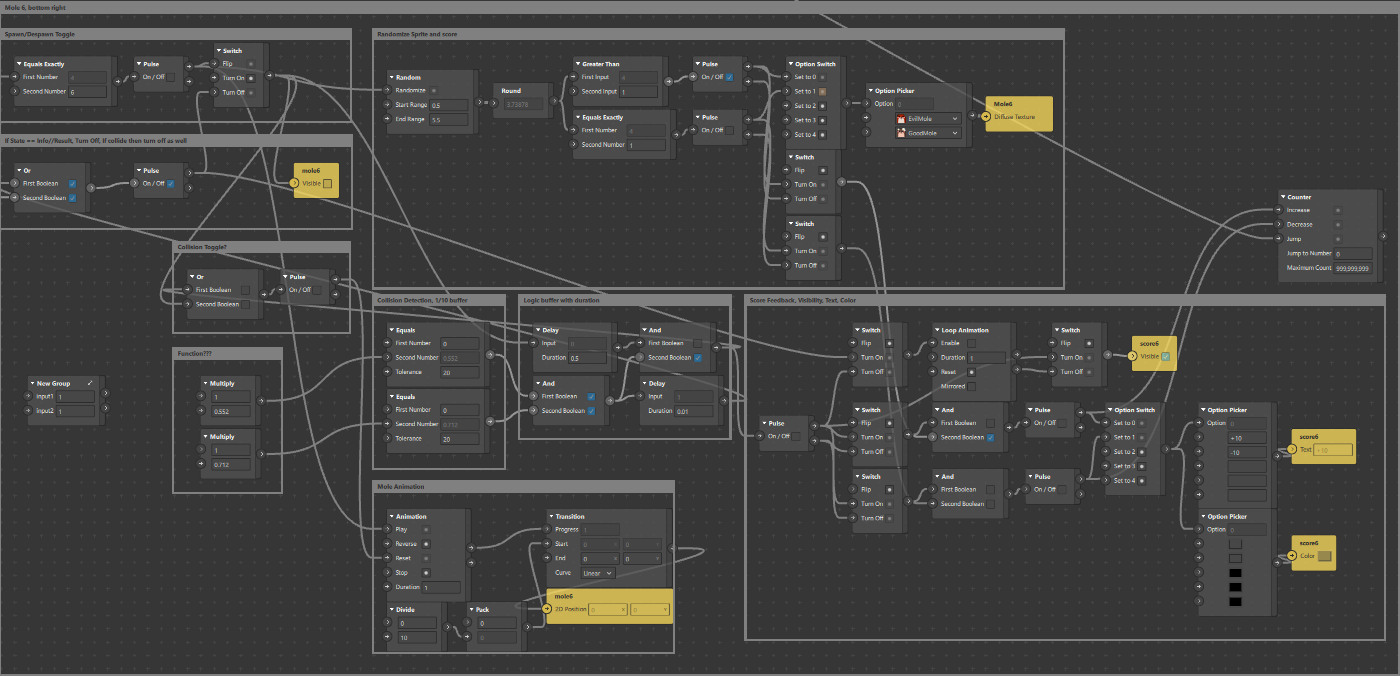
Mole Manager:



Through the mass of wires, the mole manager is a collated array of self updating moles. The triggers and inputs are as follows,

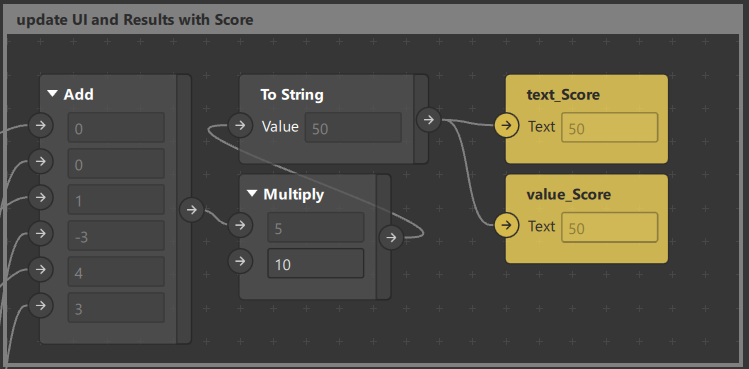
1. Pulse reset, to force reset all values and states
2. Boolean isActive, to enable behavior only on game state 1
3. Number Random Val, refer above^
4. Un-wired inputs such as:
   1. Number index, to compare to the random value
   2. Number Anim Duration, not necessary, but ensure more than 1s, handles spawn animation and score popup duration
   3. Score text and score color should not be changed, provides player feedback
   4. Mole texture, good mole and bad mole texture, updates the diffuse texture during runtime.
5. Vector 2 Screen Size, refer above^, handles collision detection
6. Vector 2 Tracker Loc, handles collision detection
7. Vector 2 Offset, legacy value used to calculate relative screen location of the mole, only change when changing the map layout.
8. Number Tolerance, a pixel limit between the magnitude of the tracker and the mole, handles collision detection “smoothness”

Inside the Mole Updater patch:



Commented frames. Each portion of the code handles 1 core responsibility, such as the entity switch, randomized sprites, spawn animation, collision detection and finally collision response. Scores are being tracked on the counter, which is output below. Values are reset when pulse reset is triggered.

Score tracker:



Uses a counter within the moles to track scores, then multiply by a factor of 10 for the final score. This assumes that hitting good moles have the same weight as hitting bad moles. This is balanced by a 1 in 6 spawn rate of good moles.

### Additional Notes

Current implementations of scripts are used as global variables allowing for unofficial loops, thus suppressing loop errors and preventing recursive calling. The scripts use get and set functions to pull data from existing patches or set the values. This is an effective but primitive way to handle data, especially when there is no added logic or transformation of the data within the code. I have taken to rename more functions as well as document them for further use with a clearer understanding in the future.

# Development Details

## Current Tasks

Such empty

## Bugs & Issues

Such empty

# Test Plan

Instructions for setup & deployment. Use of video call & screen record for data collection. Only 1 iPad should screen record, preferably the host.

1. Set up 2 ipads with gmail within the messenger app, gmail should not be the same.
2. Add the ipad to a chat group, if not done prior.
3. Use 1 of the ipad as a host & video call the chat. This will call all participants within the group.
4. Enable screen share
   1. Within the messenger video call, there is a sharing button at the bottom of the screen.
   2. Recommend features include games & watch parties. Screen sharing is located on the rightmost button.
   3. Select screen share option & follow instructions given.
5. The host of the call will enable screen recording
   1. The screen record function can be found within settings, control center.
   2. Pull down the notification center & long hold the record button. This will expose the detail menu, click the microphone to enable audio recording, this will make the microphone red.
   3. Clicking through the record button will now record the screen & audio of all applications.
6. On both ipads, open the mobile player for meta sparks
   1. Click the hamburger menu located on the top left or top right of the app & select the most recent version of the target game.

Notes: the ipad can record a limited amount of videos, make sure to export the video in between play sessions. This is unlikely to present an issue if alternating ipads record the video. Name the video in concise terms as soon as possible as the name will be generated to the date-time of recording.

## Test Strategy

Session Count: 3(1 for each of the games)

Session Duration: 30 ~ 40 minutes each

Participants: 10 elderly with differing demographics, no particular specification

Alt: (Refer to Differing levels of technical proficiency) Provide explanation and tutorial phases before playing to reduce confusion, frustration and emotional fatigue towards gameplay. (not survey)

## Test Environment

Testing Facility: Elderly Activity Centers/Active Aging Centers

Technical Support: ARISE@NTU, Remote & Physical Presence needed

Equipment: iPad with preloaded Meta Spark AR games, Meta Spark Mobile Player, Messenger, Screen Record(Built-In)

## Test Cases

Collection of Data:

1. 1st measurement, before the start of the 1st session
   1. Demographic questionnaire
   2. Intergroup anxiety
   3. Intergroup attitude
   4. Loneliness (older adult only)
   5. Well-being (older adult only)
   6. Ageism
   7. Quality of interaction
2. 2nd measurement, after the 2nd session
   1. Identical as 1st measurement, except for demographic questionnaire
3. 3rd measurement, after the 3rd session
   1. Identical as 1st measurement, except for demographic questionnaire
   2. Post-test interview on feedback on exergames and intergenerational bonding

Expected Outcome and Response: hope to see a decrease in intergroup anxiety, a better intergroup attitude, lower loneliness (in older adults), higher wellbeing (in older adults), lower ageism, and positive feedback towards exergaming.

End Goal: To examine the effect of video-mediated communication and simultaneous gameplay on intergenerational communication and bonding.

# Asset Management

Assets are split between textures & materials. Most materials are generated by the software when a texture is used. Most scene entities will not accept textures, but require material instead. This means there are 2 layers to compression to reduce the size of the final product at the cost of slightly harder development. When deleting a texture, the attached material will turn into a checkerboard pattern. When deleting a material, there will be a separate prompt to delete the attached texture. If the prompt does not show up, then manually go into the root project to delete it. This is due to the software no longer referencing the asset, thus unable to clean it despite prior deletion.

Temporary assets are used to be able to change the sprite/texture during runtime, such as the moles being good or evil. These textures are assigned randomly while the rest of the game does not interact with the textures.

## Art Assets

Previous batch made them

# Appendices