Technical Development Document

Meta Spark AR - Gamified use of face tracking

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

[Game title 1](#_50w4s46kwa4v)

[Team 2](#_t3amxc627tmq)

[Environment 2](#_x3a79x6u0bga)

[Alt. 2](#_oqcqsbceqpte)

[**Project Details 2**](#_osb6fdwox570)

[Introduction 2](#_v5bl22fofnfm)

[Purpose 3](#_w4o3jbgu61gg)

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

[Flowchart 3](#_6uhksogou4sa)

[State Management 3](#_lj7rt466zrk)

[Scripts & Patches 4](#_5fh1wksr5cwv)

[Additional Notes 8](#_kow639watvp3)

[**Development Details 9**](#_pbt1gupnbm5n)

[Current Tasks 9](#_p8y9g7504zjv)

[Bugs & Issues 9](#_fjvesghl4isk)

[**Test Plan 9**](#_icibbt57xdo0)

[Test Strategy 10](#_b41k2opvokvo)

[Test Environment 10](#_it22p8ecujqi)

[Test Cases 10](#_g2u9qqdb2fei)

[**Asset Management 11**](#_2ts8yjrtq2zw)

[Art Assets 11](#_s0dt82bic267)

[**Appendices 11**](#_zbshgjcgjpsx)

# Project Overview

## Game title

Apple Frenzy

## 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-Apple-Frenzy

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

Title: Apple Frenzy

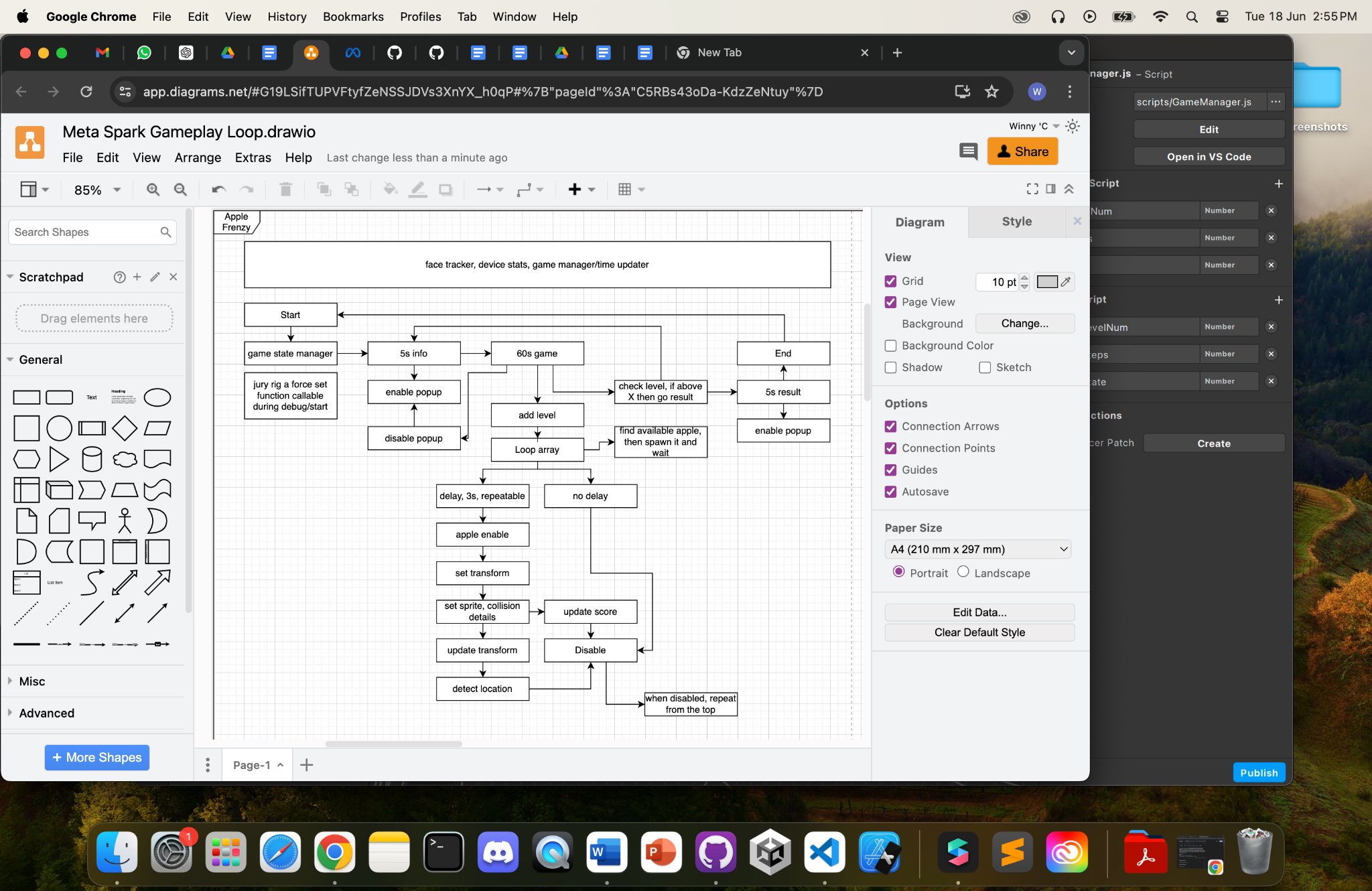
A short game about catching apples falling from above. Over time, apples will spawn & fall at a consistent pace. Good apples are red & have a faint glow, collecting them will give a score. Bad apples will fall as well, being gray & having a worm sticking out. Collecting them will result in a loss of 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

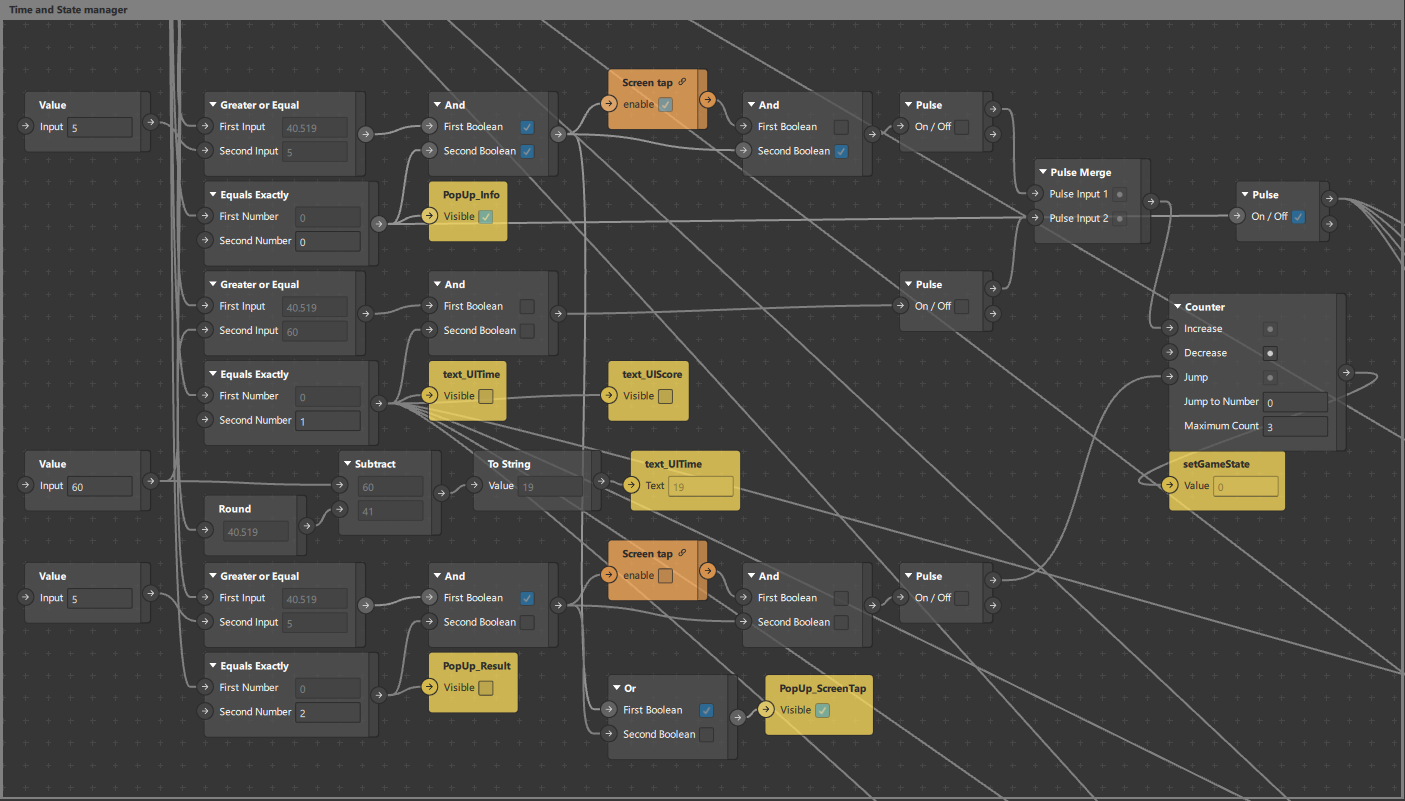
GameManager.Js

| **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()); }  *//collision result* **async** **function** **handleCollision**() {  **let** collided = Patches.outputs.getBooleanOrFallback('collisionTrigger1');  **if** (collided.pinLastValue() == true) {  Patches.inputs.setPulse('p\_hit1', Reactive.once());  Diagnostics.log(`Apple 1: Hit`);  };  **let** collided2 = Patches.outputs.getBooleanOrFallback('collisionTrigger2');  **if** (collided2.pinLastValue() == true) {  Patches.inputs.setPulse('p\_hit2', Reactive.once());  Diagnostics.log(`Apple 2: Hit`);  };  **let** collided3 = Patches.outputs.getBooleanOrFallback('collisionTrigger3');  **if** (collided3.pinLastValue() == true) {  Patches.inputs.setPulse('p\_hit3', Reactive.once());  Diagnostics.log(`Apple 3: Hit`);  };  **let** collided4 = Patches.outputs.getBooleanOrFallback('collisionTrigger4');  **if** (collided4.pinLastValue() == true) {  Patches.inputs.setPulse('p\_hit4', Reactive.once());  Diagnostics.log(`Apple 4: Hit`);  }; }  *// 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();  handleCollision(); }  *//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.

handleCollisions is a function that receives the hit condition of each apple, in terms of booleans, which is more consistent and has a larger window of time for response. This boolean value is then used to output a single pulse to trigger collision responses in other parts of the code. This is a primitive method to create subscription based events.

State Manager:

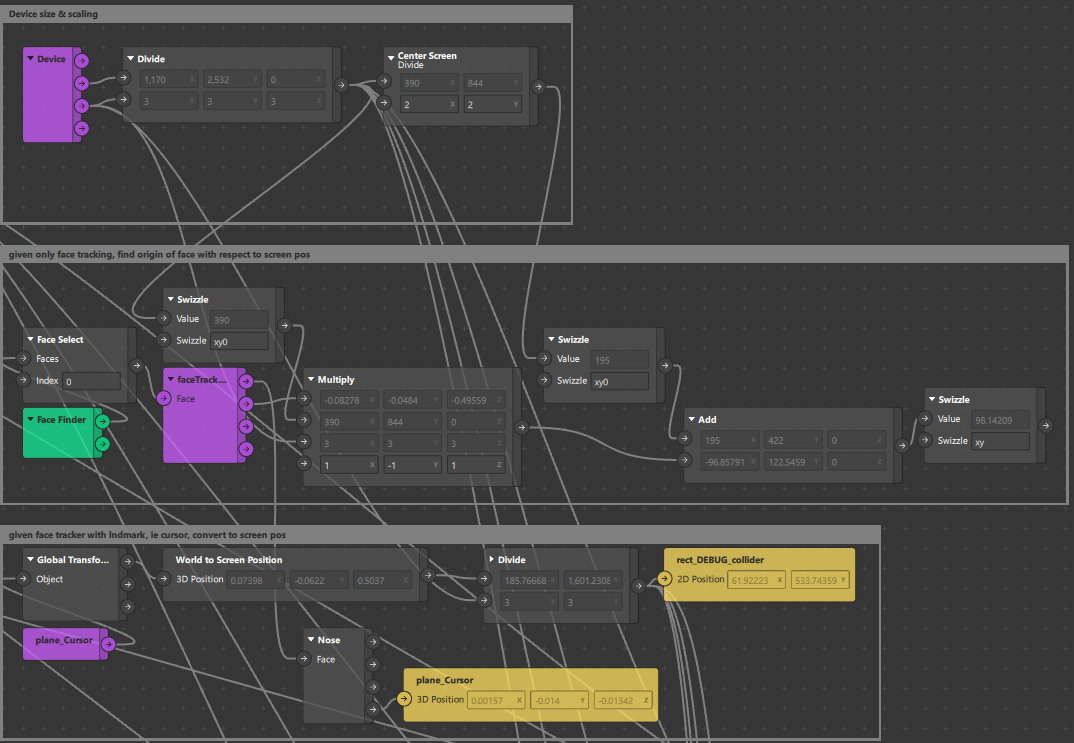


Given a finite number of states to switch between, each switch uses either a time AND screen tap to transition, except for state 1, where it is the game state, solely using a timer, this will be subject to change between the other games, such as time running out OR completing a level.

Orange patch is a patch asset, a callable function that can be exported between projects, and is synced to all existing patches, thus useful for general functions or commonly called functions.

The above patch is a runtime patch with an offset linked to a state change detector. Using a hybrid of Javascript to detect the change globally, then sending a pulse when it has been changed, this recreates an event subscriber system.

Screen data & Face tracking:

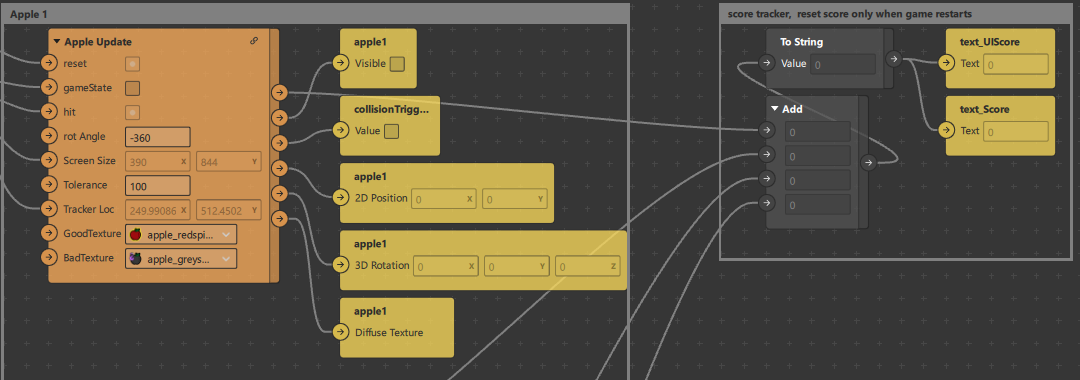


This gets the screen size and scale, then divides it to find the actual screen size. This is due to larger devices using 2x2, 3x3 pixels to smoothen graphics, allowing for a similarly made chip to increase the screen size used, but for logical detection, dividing by the screen size will make the sequential math more accurate.

This also gets the attached face tracker within the scene and attaches a cursor on it. The cursor can be on any part of the face, and with any texture.

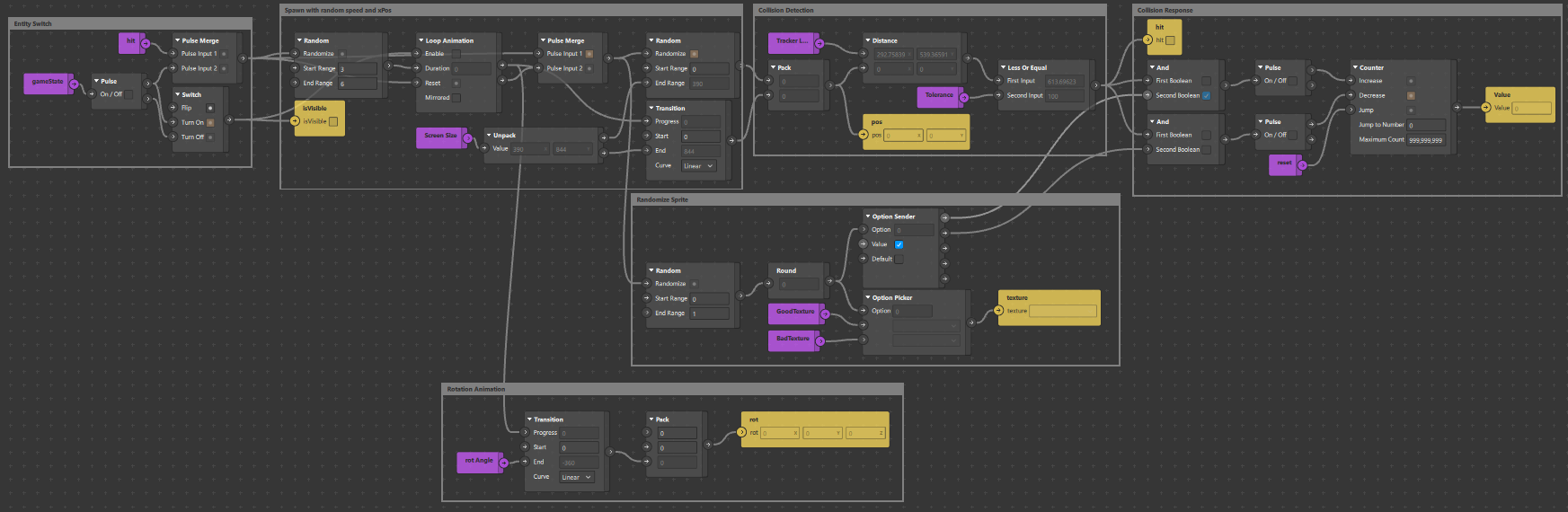
The middle code is a generalized math patch to find the origin of the face, namely the chin, where values may not be used in the actual game. The bottom code has a debug collider on it, this is just a placeholder graphic. The actual vector 2 position used is the debug collider as the cursor is placed on the face, with respect to the face origin. This means that the cursor is not on the screen and thus requires math conversion to get the actual screen position of the tracker.

Apple Spawner & Updater:



Following the previous batches’ work, I have decided to condense the apple spawner and updater to the following patch asset, this would make it much easier to update the logic and behavior of the code.

Within the Apple Update Patch Asset:



Each segment of the patch asset has been labeled, such as the entity switch that triggers the rest of the patches, but also instantly despawns all apples when the state changes. Next there is the randomizer, which spawns the apple at a random X position across the top of the screen and randomizes the fall rate. Next, it randomizes the sprite used, good apples for points and bad apples for point deduction. Next is an attached rotating animation linked to the fall rate. The other patches check for collisions and give a collision response.

### 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 apples falling with red or gray. These textures are assigned randomly while the rest of the game does not interact with the textures.

## Art Assets

Previous batch made them

# Appendices