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 3**](#_526innzjkwc)

[Introduction 3](#_v5bl22fofnfm)

[Purpose 3](#_w4o3jbgu61gg)

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

[Flowchart 3](#_v0b7a1bcb0hs)

[State Management 4](#_m7j3rrqt688a)

[Scripts & Patches 4](#_rdm1urlobj27)

[Additional Notes 12](#_jstd3rkzvb19)

[**Development Details 12**](#_sh6dune9ddsd)

[Current Tasks 12](#_8esl8fh6fwh9)

[Bugs & Issues 12](#_wqa61fso0dx8)

[**Test Plan 12**](#_icibbt57xdo0)

[Test Strategy 13](#_b41k2opvokvo)

[Test Environment 13](#_it22p8ecujqi)

[Test Cases 13](#_pdbji9gac8wr)

[**Asset Management 14**](#_b122i8t2afbt)

[Art Assets 14](#_s0dt82bic267)

[**Appendices 14**](#_rbdgmpaj7kv7)

# Project Overview

## Game title

Beach Bomb

## 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-beach-ball

//note the name change, communication error. Game to be referred to as “Beach Bomb”, while github repository to be referred to as “Beach Ball”

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

Beach Bomb:

A short 3 level game about walking the plank and avoiding enemy attacks. Turning the head left and right to move forwards. Each level is categorized by a certain walking distance, while pirates will appear and shoot projectiles at you. Stop moving or walk faster to avoid the projectiles. Being hit will reduce points and stun the player.

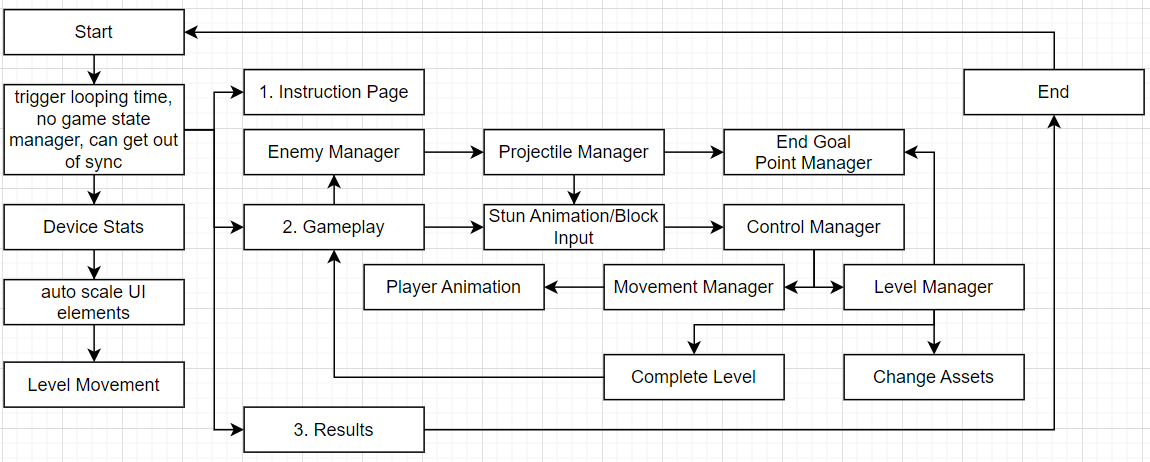
After ending the game(3 levels or 90s), you will receive stars, 0-49, 50-79, 80 onwards, as star distribution. The end goal is to get as many points as possible and get rewarded by stars.

## 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, 90s, or when the level ends

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

Effect ends

## Scripts & Patches

GameManager.Js

| **import Patches from 'Patches'; import Reactive from 'Reactive'; import Time from 'Time'; const Scene = require('Scene'); const Diagnostics = require('Diagnostics');  const targetFrames = 20;  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()); }  async function getGameInfo() {  let fWidth = (Patches.outputs.getScalarOrFallback('SetPlayerWidth'));  await Patches.inputs.setScalar('PlayerWidth', fWidth.pinLastValue());   let size = (Patches.outputs.getScalarOrFallback('SetProjSize'));  await Patches.inputs.setScalar('ProjSize', size.pinLastValue()); }  *//collision response, trigger stun* async function GotHit() {  let hit = (Patches.outputs.getBooleanOrFallback('GotHit'));  if (hit.pinLastValue() == true)  await Patches.inputs.setPulse('TriggerStun', Reactive.once()); }  *//fix this, remove game over state and replace with end game, ie complete all levels before time runs out* async function isGameOver(){  let isOver = (Patches.outputs.getBooleanOrFallback('gameOver'));  await Patches.inputs.setBoolean('b\_EndGame', isOver.pinLastValue());  if (isOver.pinLastValue() == true) { *//debug, delete when complete*  Diagnostics.log("stop game early, exit exception");  } }  *// General encapsulation of the various functions required to run every frame* async function update(deltaTime) { *// Main update function called every frame, does not run directly*  isGameOver();  getGameState();  getGameInfo();  GotHit(); }  *//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*** |
| --- |

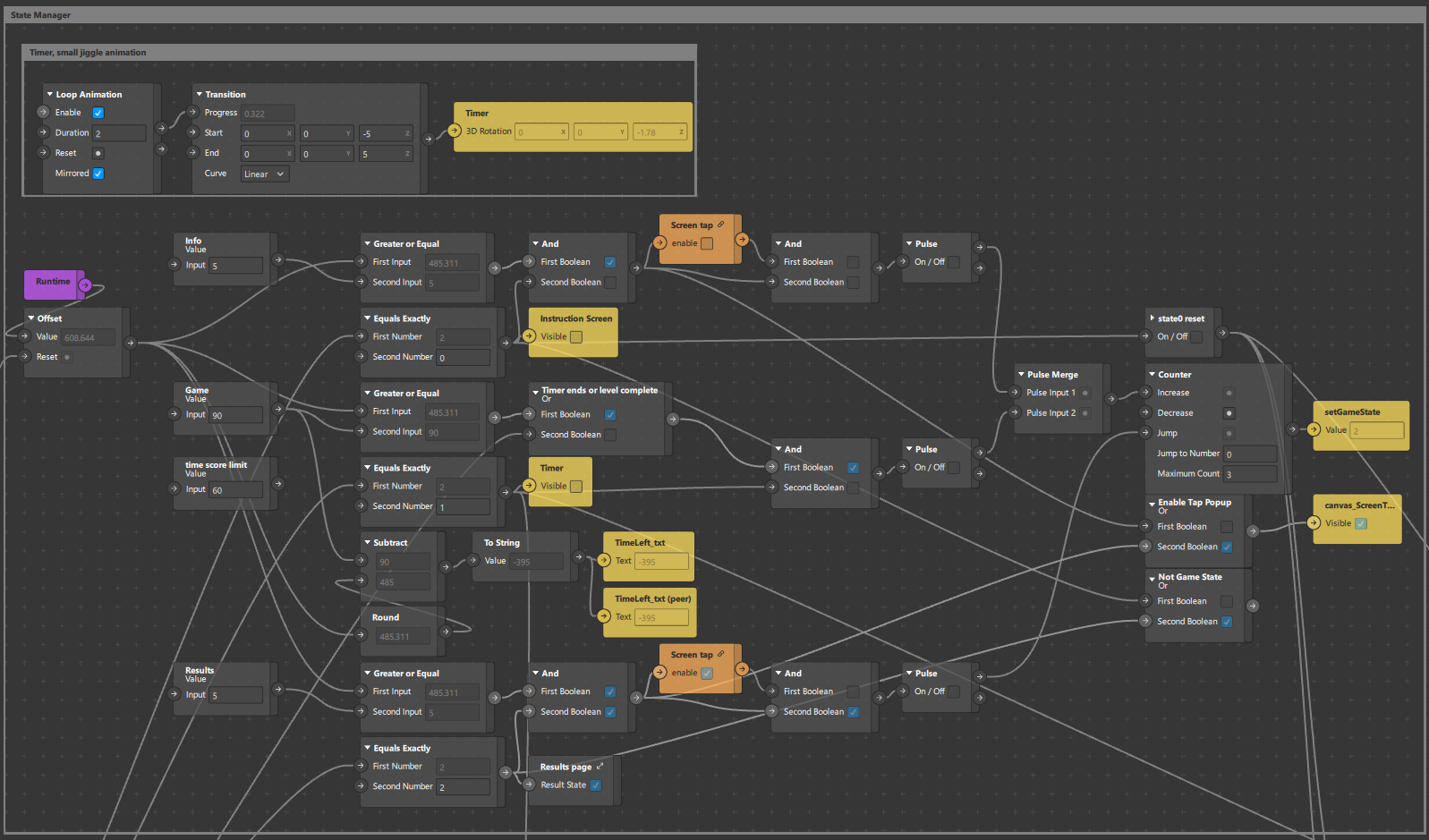
getGameState is a function that compares the internal value of gamestate to the external value found in the patch editor. If the values are not the same, it means that the value has been updated, and thus sends a single pulse to reset time. This means that each state has a maximum timer or an external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

getGameInfo is a simple getter function. Due to different device sizes, there is a sequence of patches that update the size of the player and projectiles, making it more consistent. This is then used for collision detection and making it much more stable.

GotHit is a function that handles collision response, taking in the collision and returning a stun trigger. Being stunned will prevent movement of the player as well as display a rotating animation of stars spinning, similar to cartoons.

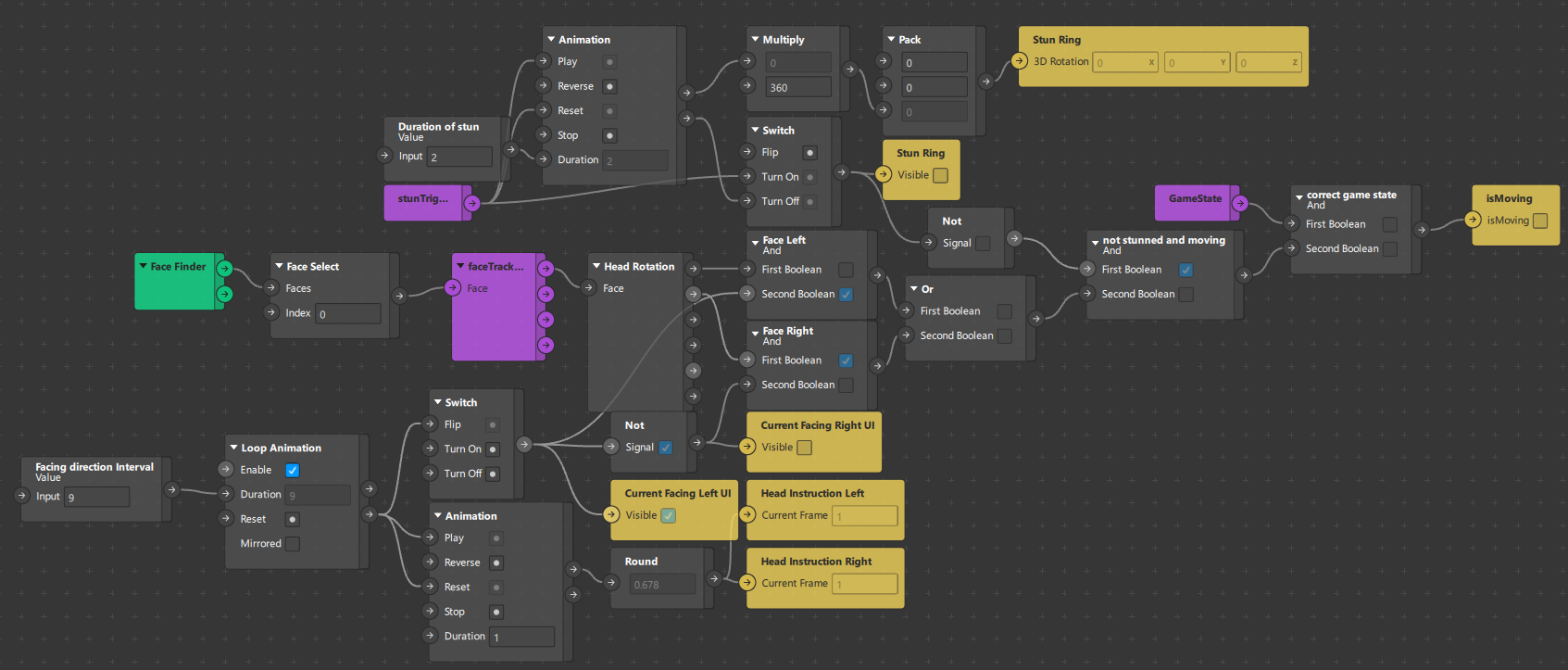
isGameOver is a similar function that makes game over a global value and thus updates all the relevant code, thus similar to a broadcast function.

Game State Manager:



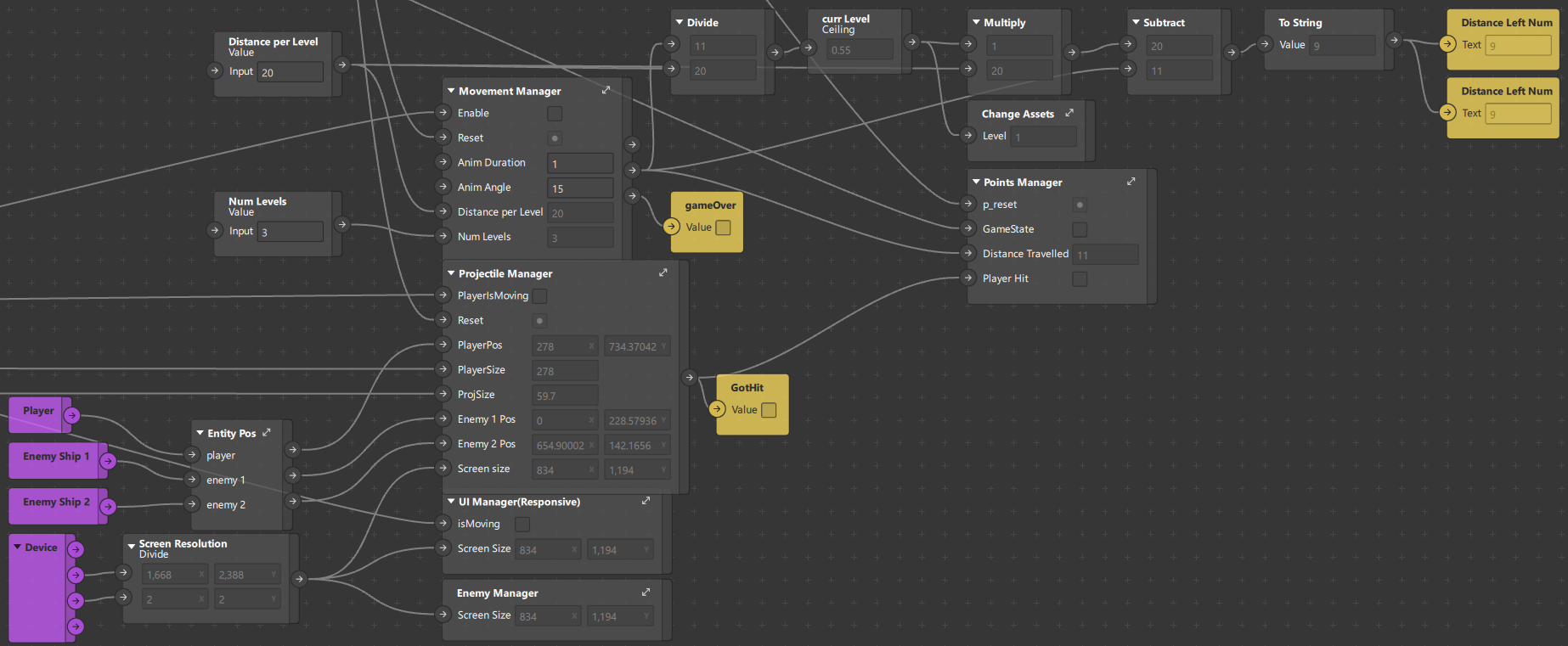
This handles game state with an early exception when the player completes the game. The transition includes a timer for a minimum duration and a screen tap popup, which the user can be prepared and choose to trigger the next state. This allows for more preparation time and easier syncing of gameplay 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.

Control input, face detection & stun trigger:



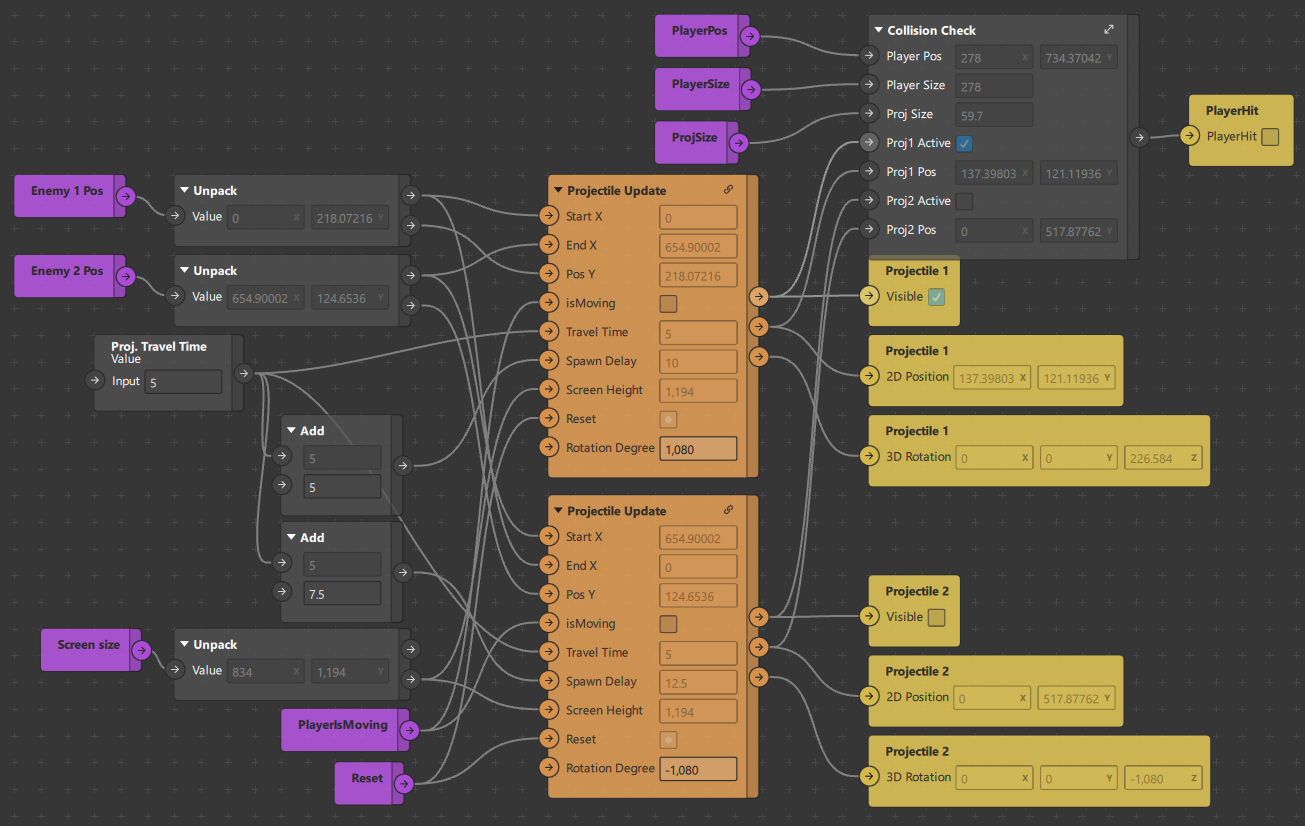
Different from the other games, the input uses the face’s direction instead of position. When the head turns left and right, the player character moves. This also takes into account getting stunned, which triggers an animation and prevents movement for the duration.

General Behavior:



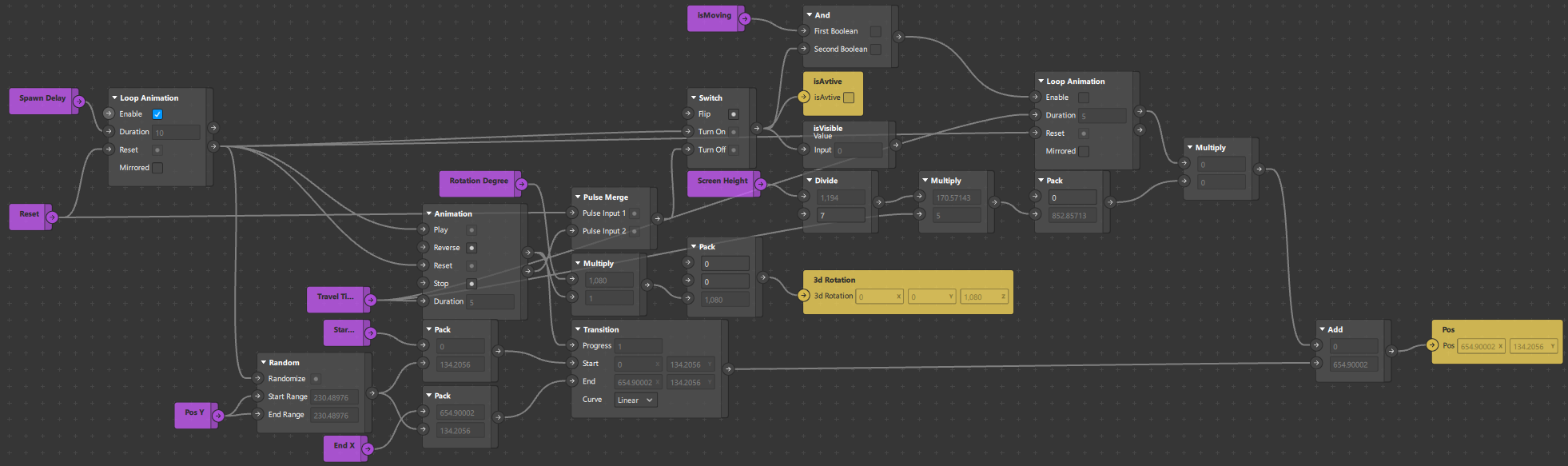
Tracking various states such as player position, enemy 1 & 2 position to spawn projectiles. Using an input manager to trigger movement and animations. Lastly the outputs of collision and distance to update level counter and score tracker.

Projectile manager:



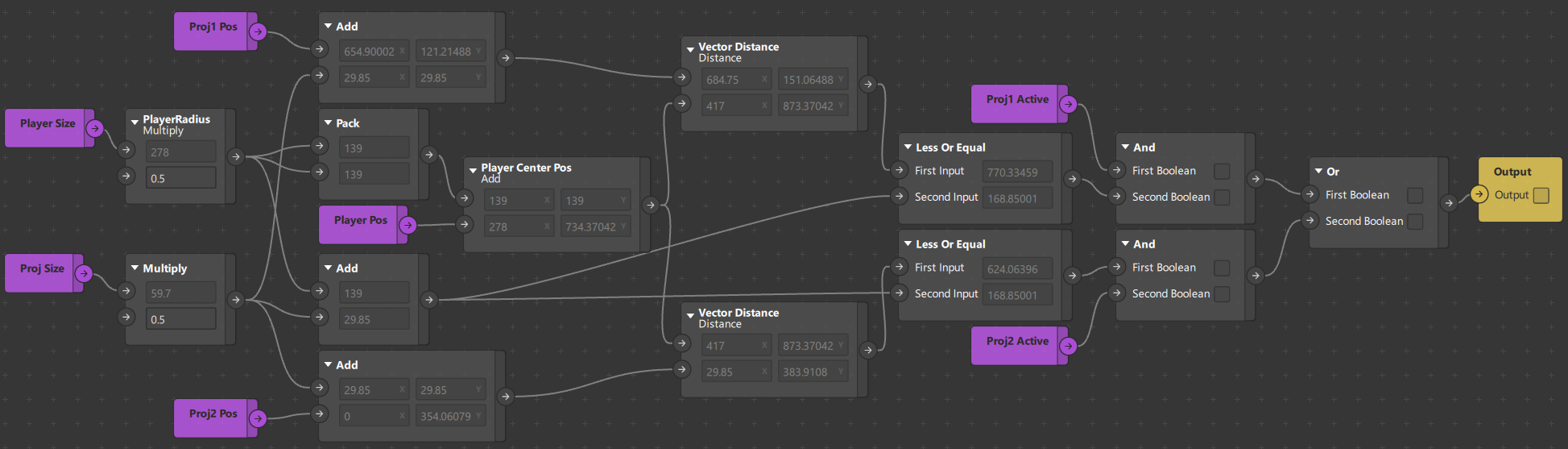
This handles projectile spawning from each of the enemy boats. Projectie 1 spawns from enemy 1 and projectile spawns from enemy 2. The projectile update and collision check will be explained below. Projectiles have a travel time of 5s, which is added to a spawn delay to alternate/stagger spawns.

Projectile update:



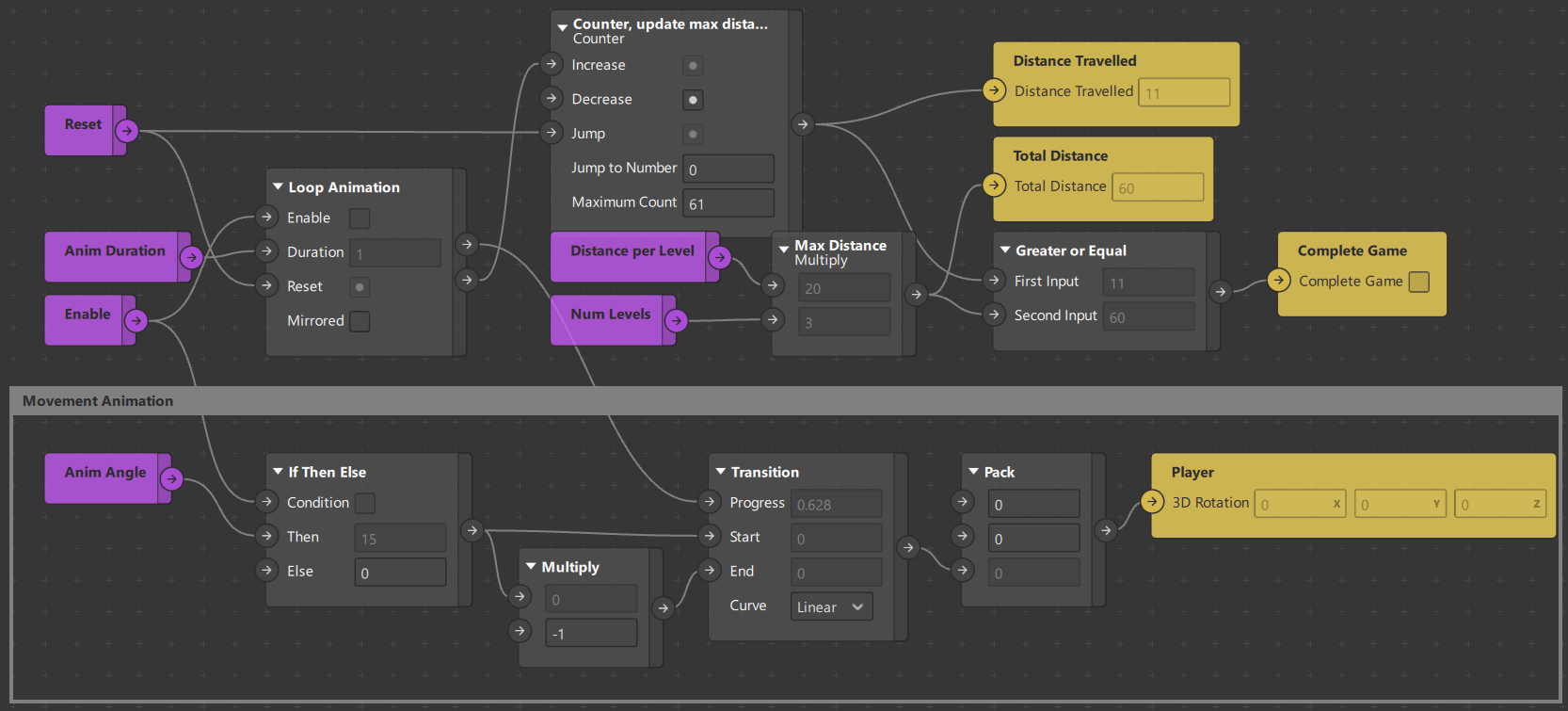
It spawns a projectile from the enemy ship, then travels across the screen for 5s(travel time), playing a rotating animation. Animation, travel direction and yPos is based on the enemy position.

Collision check:



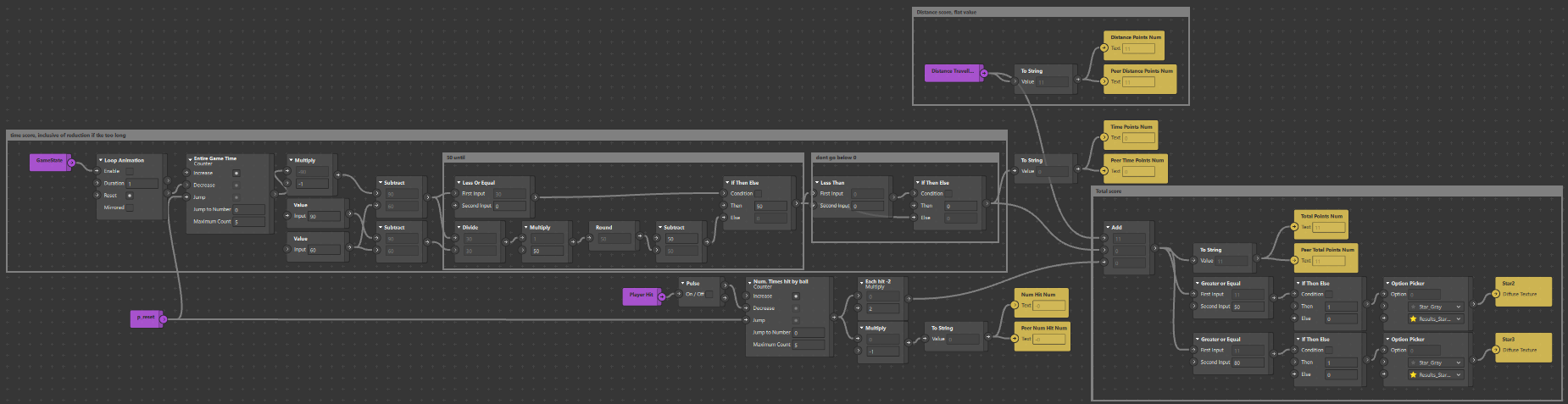
Using projectile 1 and projectile 2 position, compared to the player’s position, by vector magnitude. If the magnitude is smaller than a given tolerance, the player is considered to be “hit”.

Movement update & Animation:



Given a distance to travel, as long as the player moves, it triggers a movement animation, and updates a counter to track distance traveled. It outputs the total distance required, distance traveled and lastly checks whether the player completed the game.

Point counter & Result page stats:

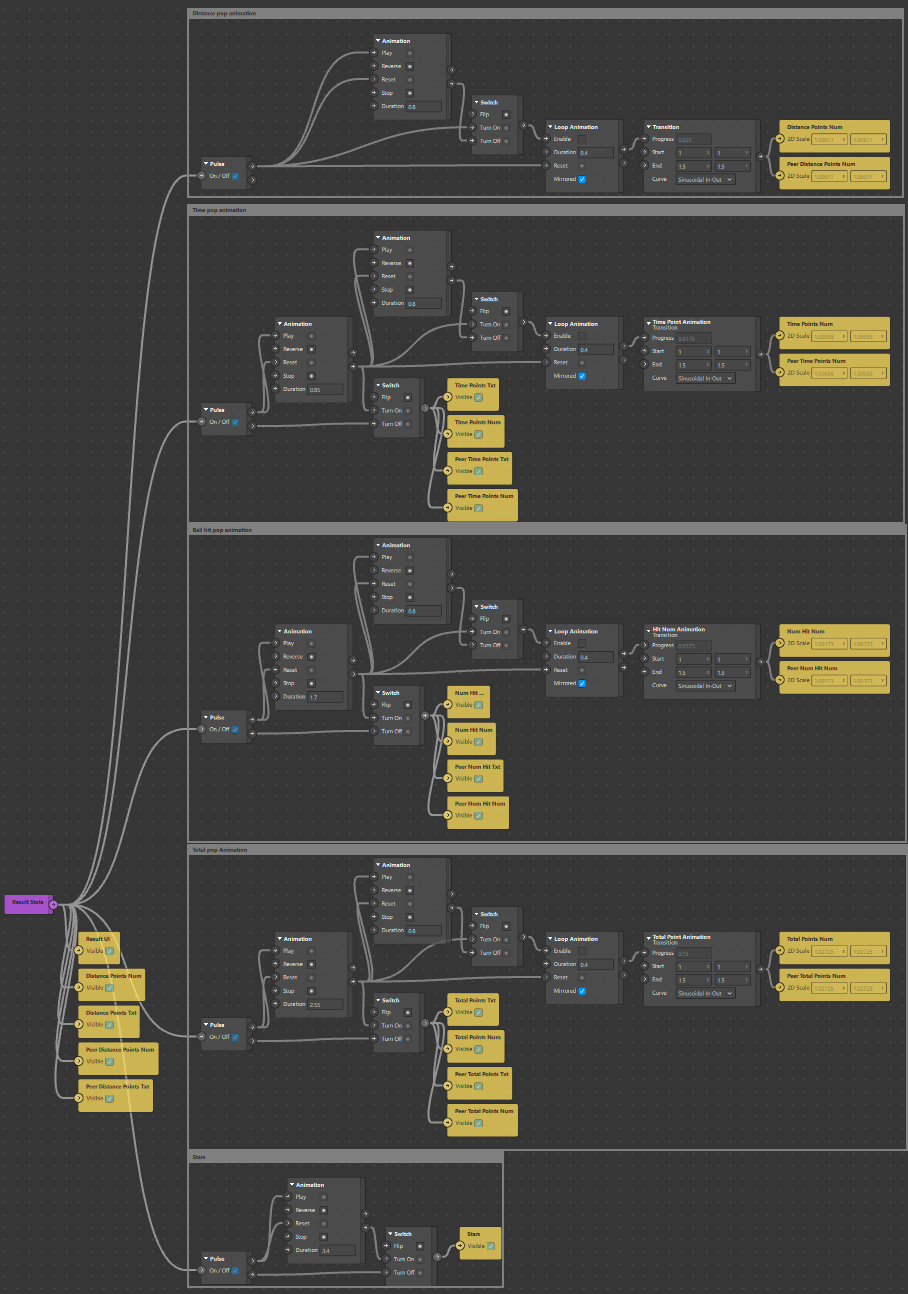


Has an internal timer on the left to track game time without external global variables. This is to counteract the constant global time.

Tracks the total score via an add patch, adding distance score, time score and reductions from getting hit.

It updates the texture of the stars, 1 being always yellow, then 2 and 3 appearing when their respective point threshold has been reached.

Result page animation:



A large animation trigger that updates the various stats on the results page, such as the bubble pop animation when values show up.

### 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 the 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 stars being gray or lit up. These textures are assigned by score while the rest of the game does not interact with the textures.

## Art Assets

Previous batch made them

# Appendices