PORTFOLIO

By Haotian Zheng

TABLE OF CONTENTS

Procedural Generation
Live Wallpaper Series
Node Editor
Other Works

NOTE

To view all (and newly added) projects with additional images, videos, and explaination texts, please visit the online portfolio at
https://portfolio.justzht.com/

PROCEDURAL GENERATION

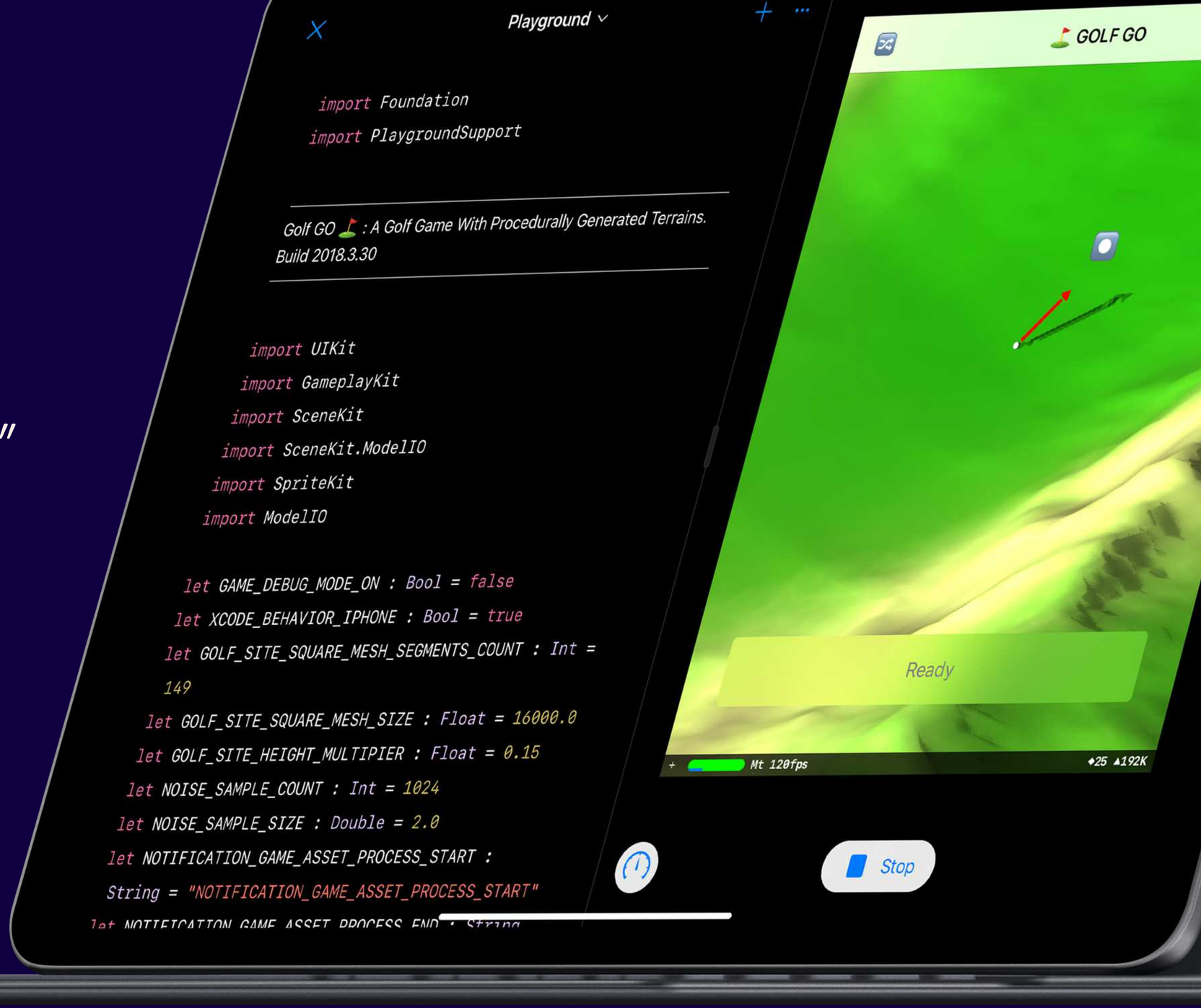
By Haotian Zheng

I always find procedural generations fascinating for its ability to create infinity and random worlds with limited resources. So you might wonder what a basic one would be like? Meet ***Golf GO****.*

GOLF GO

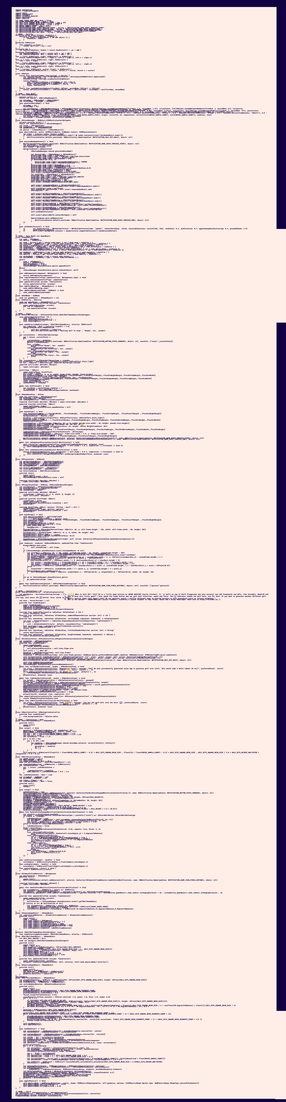
"APPLE WWDC 2018 SCHOLARSHIP WINNER"

Golf GO is a Swift Playground, one of 350 WWDC 2018 winner projects around the globe, and a casual game written ***within 1000 lines*** but offers ***thousands of*** different golf course maps.



WHAT'S IN 1000 LINES?

SOURCE CODE



FUNCTION

Static Defines *(23 lines)*
Helper Methods *(55 lines)*
Data Model *(22 lines)*

Scene Setup *(109 lines)*

Entity Component System *(73 lines)*

Button View *(76 lines)*
Menu View *(61 lines)*

Scene View *(122 lines)*

Controller Logic *(119 lines)*

Target & Arrow Logic *(144 lines)*

Camera Following Behavior *(59 lines)*

Terrain Mesh Modifier *(144 lines)*

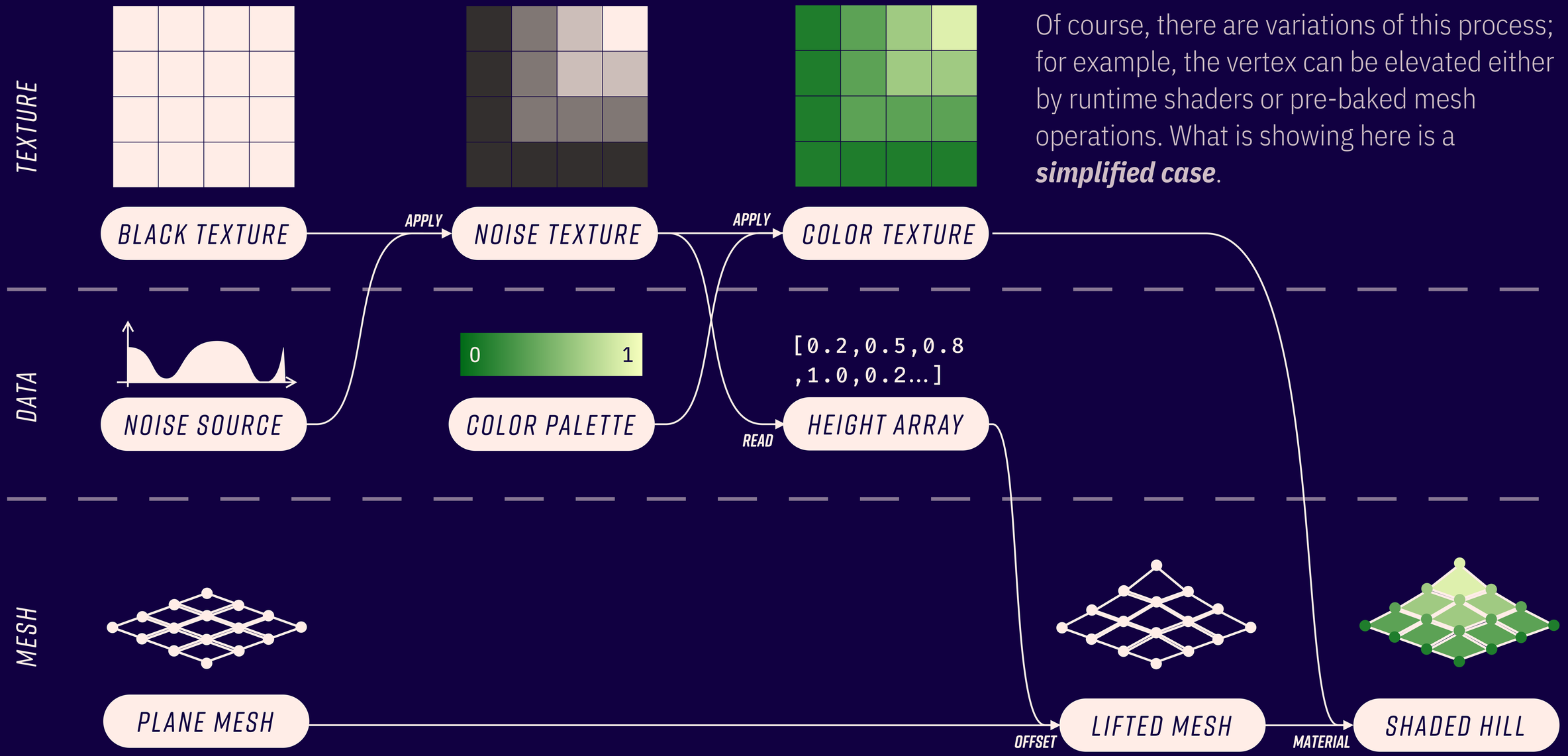
WHAT DO THEY DO?

Golf GO was written to showcase the power and creativity in Apple native frameworks, so there is no help from 3rd party dependencies in the game.

Entity-component-system (ECS) is a common architectural pattern used in game development. Apple provides such abstract layers in GamePlayKit with ***GKComponent*** and ***GKEntity***, but without a proper bridge to SceneGraph objects like ***SCNNode*** in SceneKit.
In Golf GO, I wrote a wrapper called ***JZGameObject*** that contains both references to the GamePlayKit and SceneKit and works in a way similar to Unity's MonoBehavior. For example, a GKComponent subclass called JZCameraFollowBehavior would have the render delegate of SceneKit dispatched to its update methods every frame from the mounting JZGameObject, in which it can access and modify the Transform property to make camera entity to follow the moving golf ball.

In Golf GO, golf course heightmaps are generated from noise generators like **GKBillowNoiseSource** from Apple GamePlayKit. Then the heightmap as a 2D array would be applied to a plane mesh, raising the z-axis of each vertex on the plane by the value in the corresponding 2D array. After the ***offset modification***, a MDLMesh from ModelIO framework would be used to generate normals based on the new terrain, giving it an accurate appearance when combined with a lighting model.

A GLIMPSE OF THE RANDOM GENERATION PROCESS



Of course, there are variations of this process; for example, the vertex can be elevated either by runtime shaders or pre-baked mesh operations. What is showing here is a ***simplified case***.

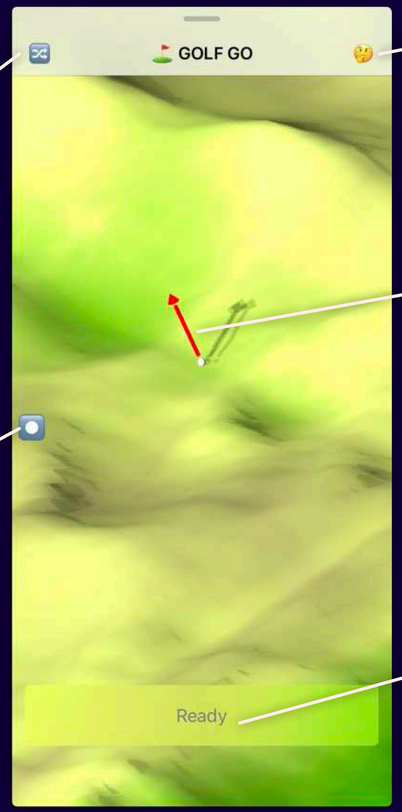
GAMEPLAY

RE-GENERATE BUTTON

Tap to generate a whole new golf course map. It takes about 2 seconds on an iPad Pro.

HOLE INDICATOR

The indicator functions as a SpriteKit based marker guiding you to aim. It works offscreen, just like ones in many games do.



MAIN INTERFACE

(under 300 lines)

HELP BUTTON

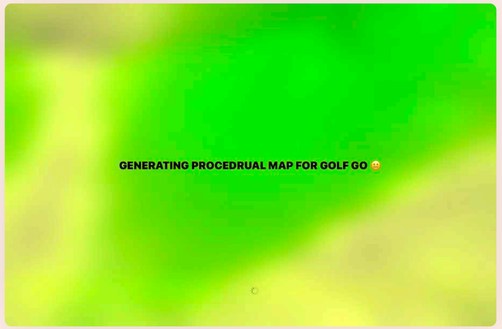
Press to show FAQ about the game and WWDC submission info.

AIMMING INDICATOR

Changes between 4 states:
- none: do not display at all
- yaw: 360-degree rotation on global XY-axis plane
- pitch: 90-degree rotation (back and forth) on local XZ-axis plane
- force: scale arrow from (1,1,1) to (1,1,5)

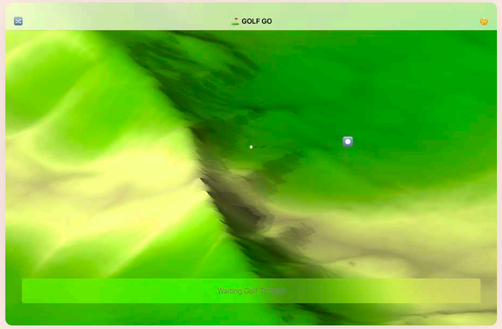
STATUS CHANGE BUTTON

Button disabled when the golf ball is moving. Button active when the golf ball is static and in one of 4 states:
- ready: ready to enter yaw mode
- yaw: press to confirm yaw angle
- pitch: press to confirm pitch angle
- force: press to hit the ball with current force.

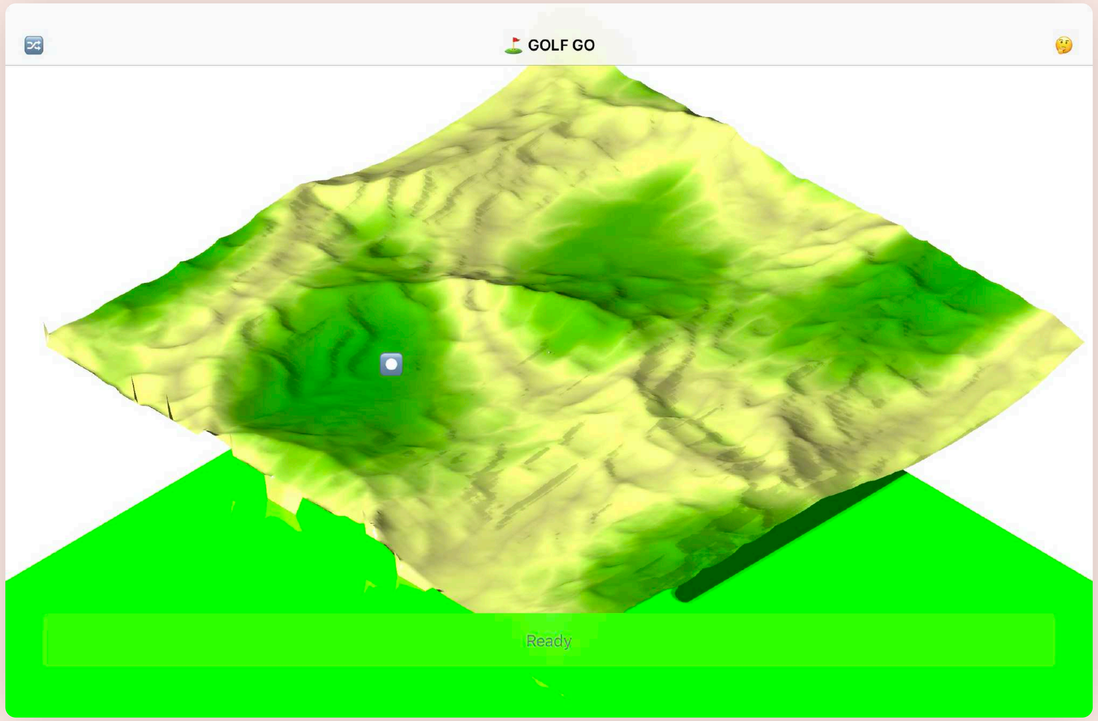


GENERATING

AIMING



HITTING



Here is a golf course map viewed from a distance in Golf GO.
What if we scale the golf world way up? Like, to the size of Earth?
Meet ***Lonely Planet***.

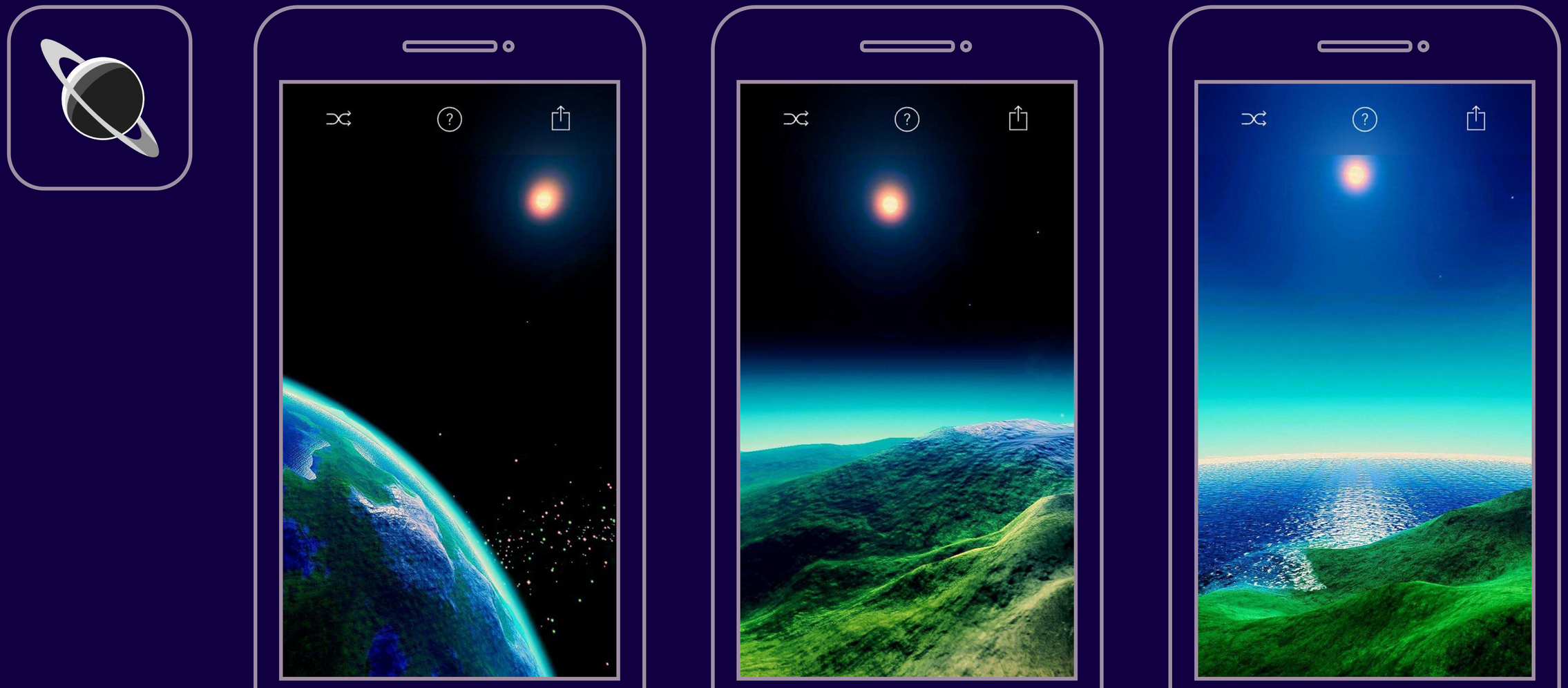
LONELY PLANET TECH DEMO

"MILLIONS OF PLANETS ON YOUR PALM"

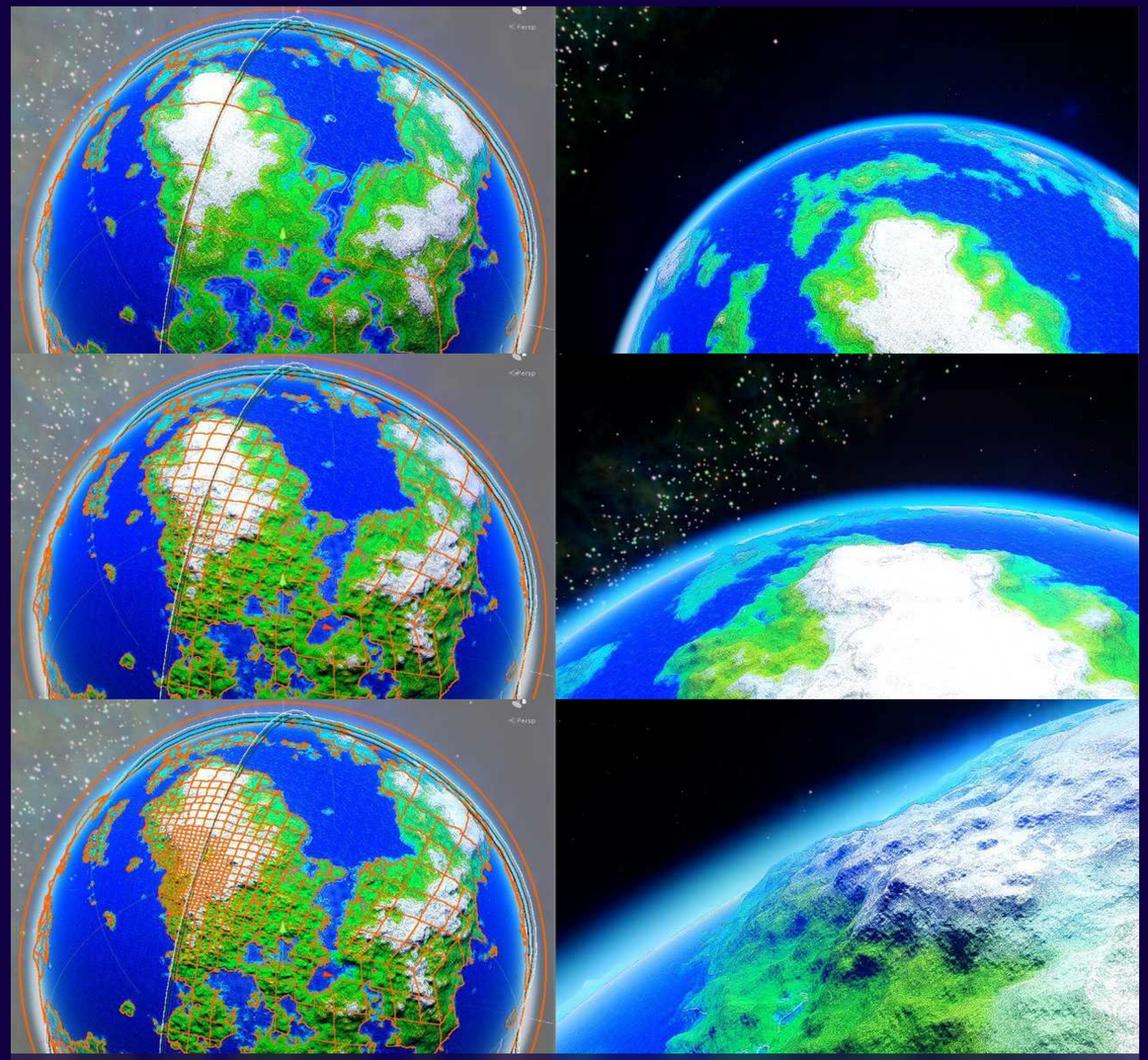
Built with Unity in Sept 2016, ***Lonely Planet (Epoch Core)*** utilizes my own cg port of the originally C++ written noise framework LibNoise to generate heightmaps on GPU, which shortened the time for a perlin noise texture ***from 12 seconds to 0.15 seconds*** while offering ***millions of different*** terrestrial planets.

A STEP FORWARD IN GRAPHICS

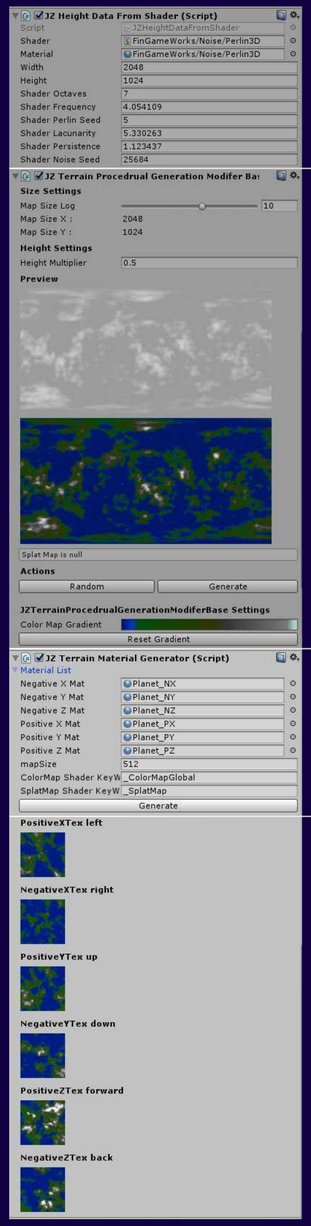
Unity as a game engine is far more versatile than Apple SceneKit, and thus it enables the gorgeous graphics of Lonely Planet. In it, I used various techniques like ***quadtrees***, ***triplanar-projection terrain shaders*** as well as the help from some 3rd party assets, including the ***atmospheric scattering*** solution.



PROCEDURAL GENERATION WORKFLOW IN UNITY



*The generation is **distance-aware** for a seamless transition from outer-space to the ground. As the player approaches sea level, terrain chunks near the player would be divided into small pieces, as shown on the left.*



1

*I ported the **perlin noise** module and some helper methods from LibNoise into a shader lib (.cginclude) to **generate heightmap on GPU** via RenderTexture (As in 2016 the Unity compute shader support on mobile wasn't complete)*

2

*The generated gray-scale map would then go through **another pass for coloring** with a predefined gradient (0.0 for sea / 1.0 for mountain). For a 2k resolution map with stacking noises, this whole process would cost about 150 ms on an iPhone 6S.*

3

*With quadtree, the planet would be split into 6 chunks: X+, X-, Y+, Y-, Z+, Z- in local spaces. Think about **a cube** with **each side** being a quadtree as well as each vertex elevated to a fixed radius. This would efficiently create a sphere. Based on the sphere, Lonely Planet applies another radius offset based on the heightmap texture, which would give terrain its shape.*

4

*The final part would be applying 6 colormaps to each side. The planet viewed from **far away** would directly display these colormaps for **performance** but uses a triplanar mapping with ground textures and colormaps as splatmap references when viewed **closeby** for more **detail**.*

With a functional procedural planet system, why not generate millions of them and place them randomly in the universe? Meet ***Epoch****.*

$\times \infty = ?$

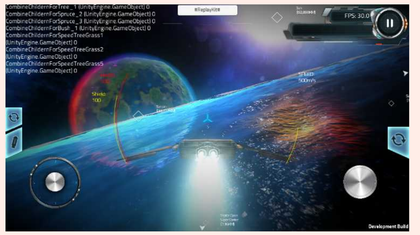
EPOCH (WIP)

"FULLY PROCEDURAL AND PERSISTENT UNIVERSE"

Epoch is the project behind Lonely Planet Tech Demo. Started from 2016 as an ambitious game, Epoch would combine previously tested procedural planet system with star systems, battleships and fighters, factions and stories ... all things that would make it a full-fledged game when finished.

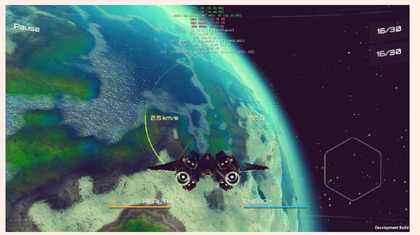
DEVELOPMENT STATUS

Epoch has been paused enormous times due to my personal affairs like courseworks, internships, and full-time jobs. There is a playable demo on ***itch.io*** which demonstrates its progress in Nov 2016.
With a future development plan including procedurally generated cosmic phenomenon, improved flight control, AI system, and an infinity star map, it is now estimated to be finished in Q1 2024.



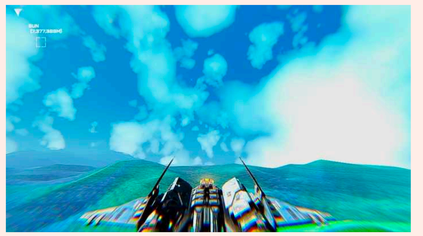
JAN 2016
INITIAL COMMIT

MAR 2016
VEGETATION



SEPT 2016
TOTAL REWRITE

NOV 2016
COMBAT SYSTEM



SEPT 2017
VOLUMETRIC CLOUDS

END

Procedural Generation By Haotian Zheng

Additional Info: https://portfolio.justzht.com

LIVE WALLPAPER SERIES

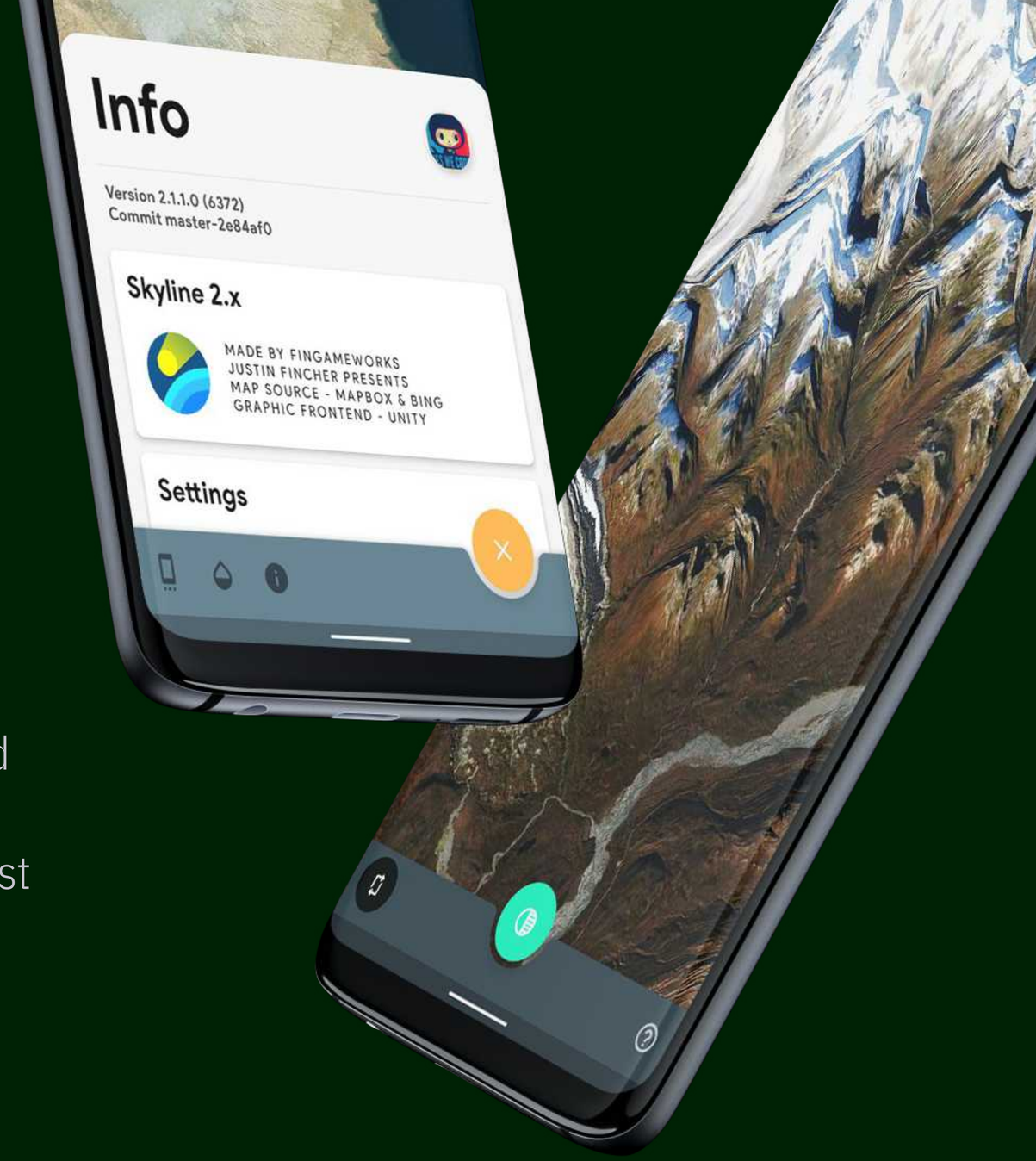
By Haotian Zheng

I love tossing around with frameworks and SDKs as if they were LEGO bricks. Every once in a while, I would find ultimate combinations that turn specific things into a wild card. You might think it ordinary but I call it a hack. Meet ***Skyline****.*

SKYLINE

"ONCE THE MOST PAID APP ON GOOGLE PLAY STORE*"

Built with Unity and my own Android wrapper in Oct 2017, ***Skyline Live Wallpaper*** was ***featured by The Verge***, LifeHacker, The Next Web, and Android Authority for its bold implementation in pushing ***3D satellite terrains*** onto users' home screens. It maintained in the top 5 in most paid app list for straight a week and sold $15k in one month.



** Data Source: App Annie, Date: 2018.1.21, Google Play US Region*
*** 'JustinFincher' on the screenshot is an alias ID I used on GitHub & Internet*

THE IDEA

Google has a similar product pre-installed exclusively on its Pixel phones, called ***Pixel Live Wallpaper***. With the resources of Google, you would expect it to be a Google Earth on home screens, but it doesn't, ***offering merely 6 pre-defined locations***.

Skyline, on the contrary, could load 3D terrains ***anywhere in the world as well as on any Android phones***.

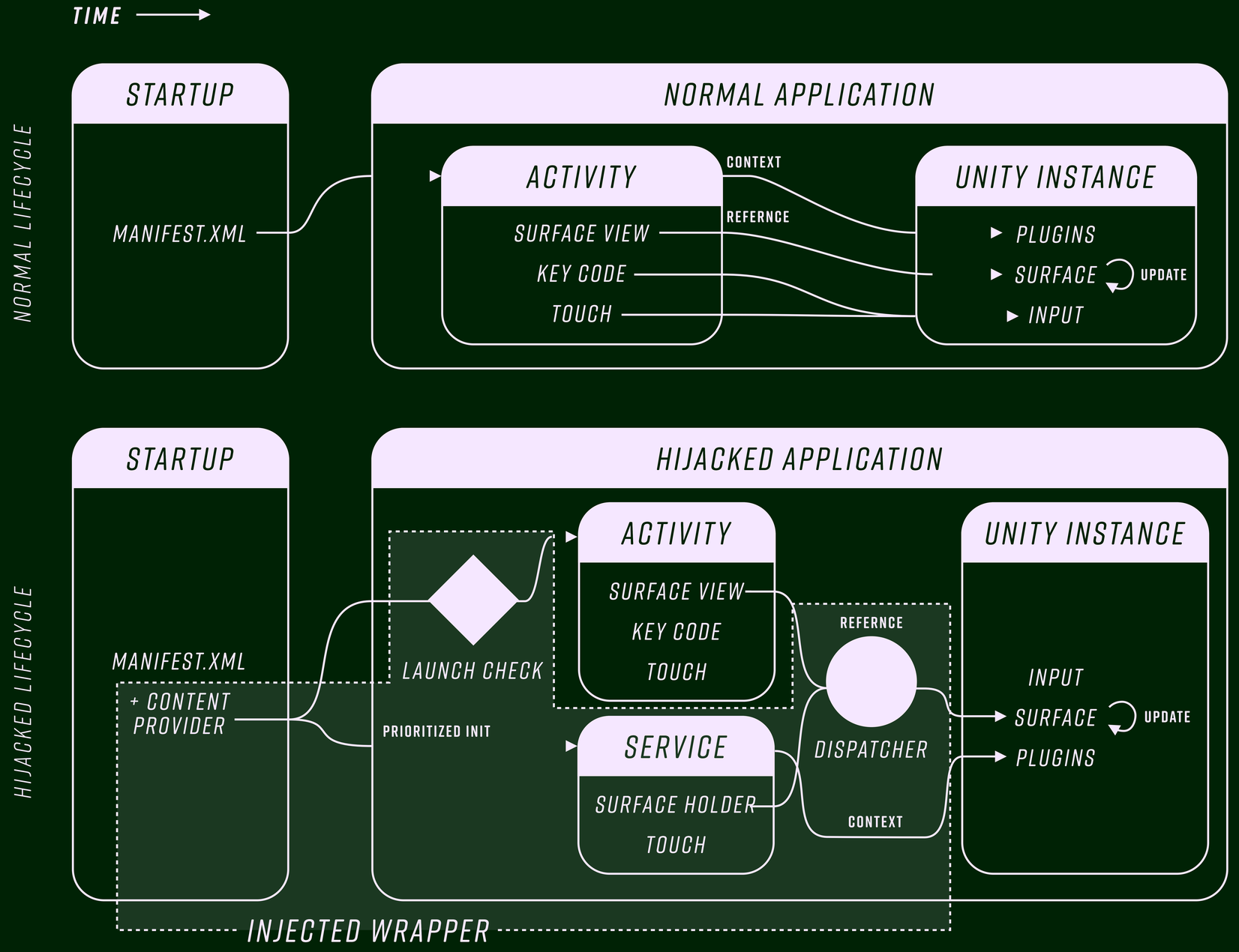
The wild imagination of transforming your phone screen into a window of International Space Station made it ***unique***, and the inclusive natural for all devices made it commercially ***popular***.

WHAT'S THE CATCH?

Google uses pre-defined locations for a reason: the terrain mesh can be ***pre-baked and optimized*** ahead of time, which is battery-friendly, especially with the native java graphics library LibGDX.

Skyline 1.x was using ***Mapbox Unity SDK***, the only worldwide terrain streaming solution for indie developers at the time being.

However, Unity, as a full-screen game engine, isn't designed for live wallpapers. It expects its instance staying at ***foremost***, receiving user inputs, and ***killed*** right after playing, yet a live wallpaper needs to rest at the ***bottom*** without any interference with user inputs, and most importantly, ***keeps*** running for weeks. One must reverse engineer Unity to ***bypass these limitations*** and that was exactly what I did.

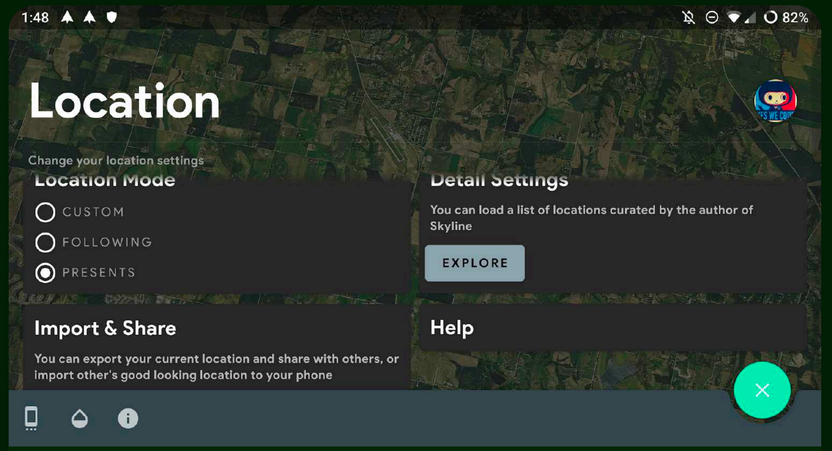


From inspection of the decompiled Unity Android .jar file, the official startup process involves an ***Activity with a SurfaceHolder*** that connects with the Unity instance.

Though it is pretty straightforward, the ***over-reliance on the Activity*** is prone to raise issues, as the Unity instance uses it as the context of the initializer as well as of plugins. If the ***Activity was killed***, which is possible in the live wallpaper scenario, then Unity would throw ***null reference errors***.

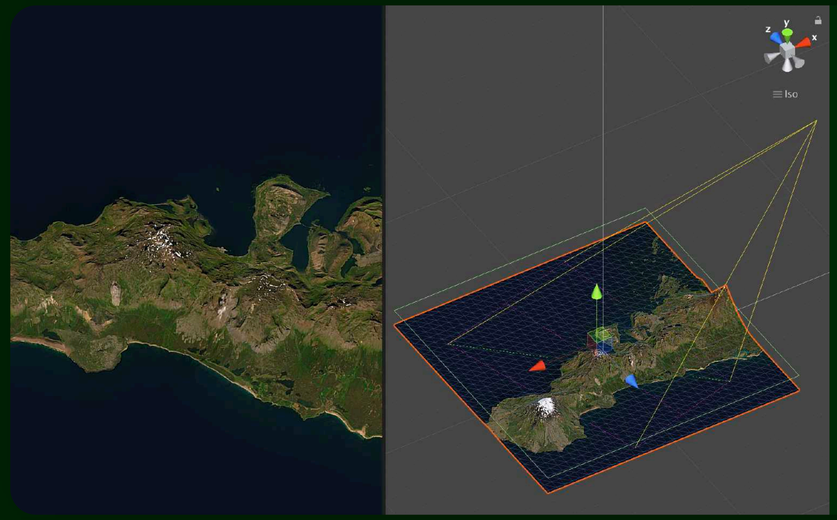
The solution was to inject a ***ContentProvider*** for ***early initialization*** of ***Service*** even before Application callbacks. The Service then would take all context-related roles from the Activity, and only ***switch SurfaceHolders*** when necessary.

Through multiple releases to improve the injection solution, I had more understanding in Android architecture, and decided to rewrite the GUI using native componments, hence Skyline 2.x, a major release emphasizing native-level performance to really outrun the Pixel Live Wallpaper.



NATIVE FEATURES

Skyline 2.x brings always on display (AOD) support through undocumented Android Open Source Project (AOSP) methods, which was previously only available in Pixel Live Wallpapers. With Slice support, Android X, Material Design 2.0, and Dark Mode support, Skyline 2.x always ***aligns*** with the stock experience.



CAMERA RIG IMPROVEMENT

The camera in Skyline 2.0 needs to handle different screen ratios, parallax angle offsets and its own rotations, which makes ***boundary management and clipping*** of the map difficult. With a runtime camera rig controller that raycasts to the ground and automactically adjusts camera parameters, Skyline would maintain the maximum view areas with the ***smallest loading cost*** of map tiles.

```java
public enum TerrainTextureProviderMode
{
    Mapbox, ArcGIS, Here, Yandex, Bing, Google
}

@JsonProperty(value = "Unity.Visual.TextureProviderMode",defaultV
public TerrainTextureProviderMode CurrentTerrainTexturePr
```

JAVA

```c#
[NativeMapping(JSONPath = "Unity.Visual.TextureProviderMode")]
JustZht +1
public string CurrentTerrainTextureProviderModeString
{
    get => Enum.GetName(typeof(Enums.TerrainTextureProvic
    set
```

C#

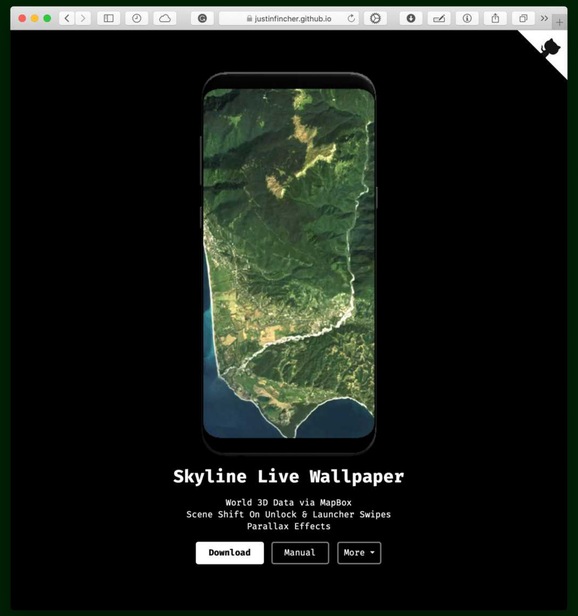
```json
{
  "Unity":
  {
    "Visual":
    {
      "TextureProviderMode": "ArcGIS",
      "ResolutionMode": "Detailed",
```

JSON

JSON BASED STATE SYNC

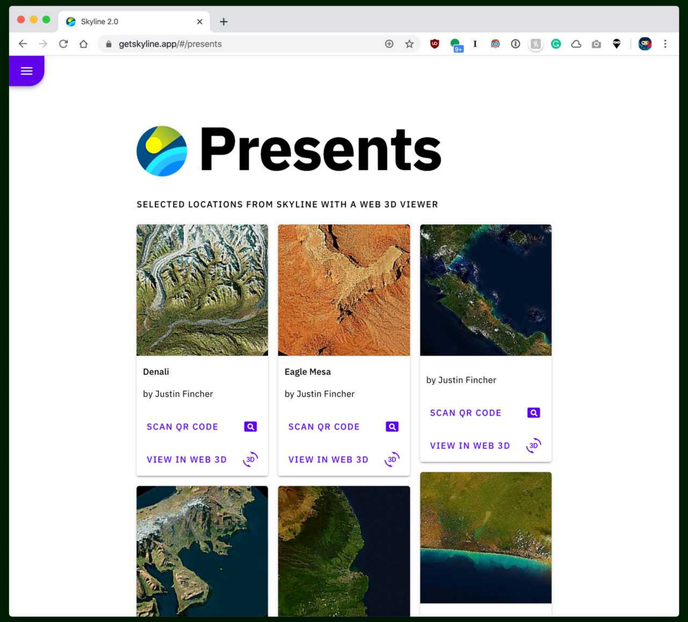
Skyline 2.x brings a whole new set of customizable settings, and it is not feasible to ***manually write getters and setters*** on both sides any more due to maintenance costs. Instead, I wrote a helper to ***parse dot-connected JSON path into nested structures*** and the tool to ***machine-generate corresponding methods automatically***. This solution works effectively as an observer pattern that only fires specific listeners instead of triggering all handlers and thus ***reduces the cost of native interop***.

ADDITIONAL WORKS



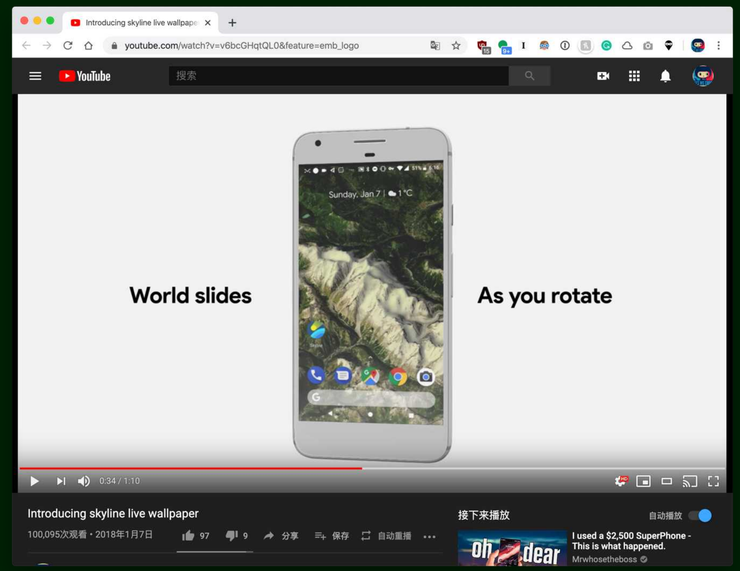
LANDING PAGE

The ***WebGL based page*** features a 3D mockup phone with a RenderTexture as screen and terrain meshes in three.js.



WEB VIEWER

Written in Vue.js, the ***viewer*** provides a way for users to share their favorite satellite images.



PROMO VIDEO

Rendered in Adobe After Effects and Cinema 4D. Passed 100k views on ***YouTube***.

*With Skyline, I nearly finished the live wallpaper framework for Unity, **UniLWP**. I said it is a wild card because it bridges the best of both worlds, that even a simple particle system in Unity would be amazing if equipped with sensor data from Android. Meet **Vortex**.*

VORTEX

"DATA-DRIVEN LIVE WALLPAPER"

The latest installment of the series in Dec 2018. What if we transform surrounding data into a ***flowing particle visualization***? Calculated from current ***weather, temperature, time, and human motion data*** of Google Awareness API, the force field in ***Vortex Live Wallpaper*** drives whirling trails in a spectacular fashion.

DATA

MAPPING

COORDINATE

SUNRISE/SUNSET

DAY PROGRESS

HUMIDITY

TEMPERATURE

MOTION/ACTIVITY

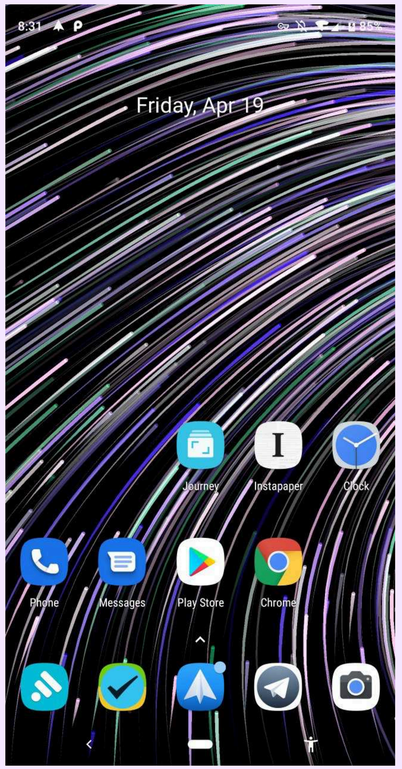
WEATHER

COLOR

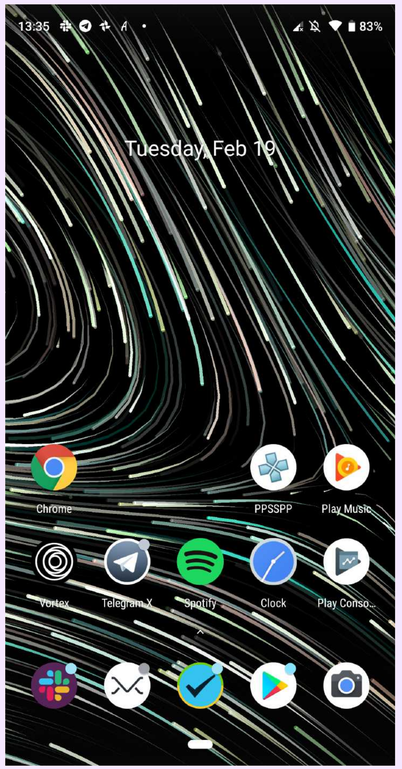
RADIUS

SPEED

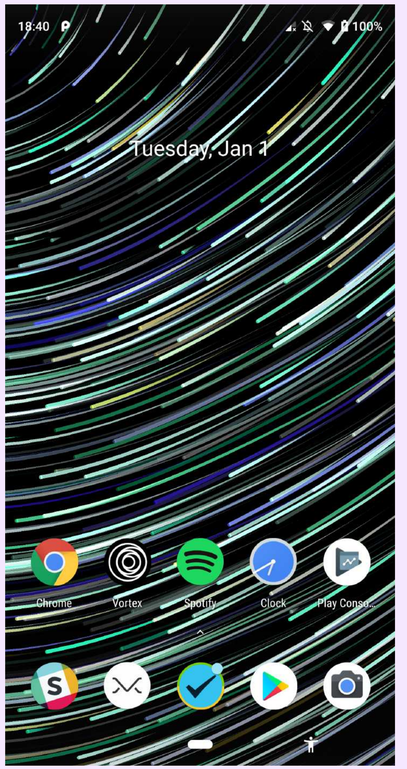
TURBULENCE



8:31 WALKING
CLEAR



13:35 RUNNING
RAINING



18:40 STILL
FOGGY

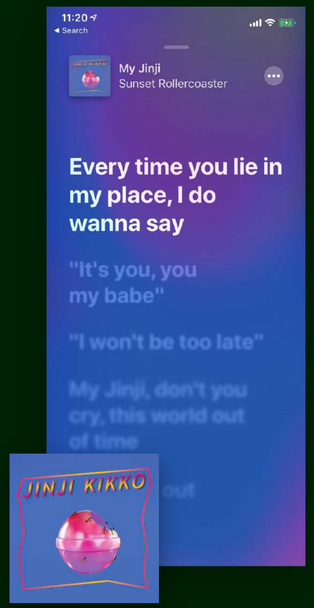
At this stage, I started to explore more on the data-driven concept and lightweight rendering solutions, the time of which I saw the gorgeous album visualization in the live lyrics view of Apple Music. And you guessed it. Meet ***Diffuse****.*

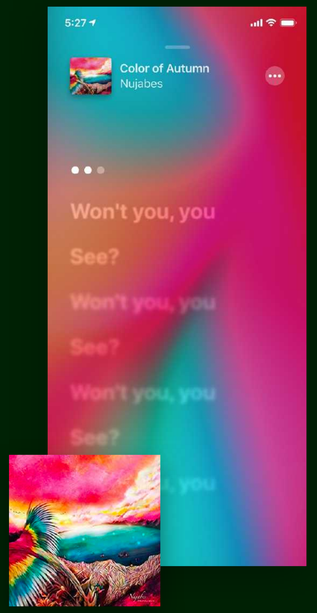
DIFFUSE

“AUDIO REACTIVE FLUID“

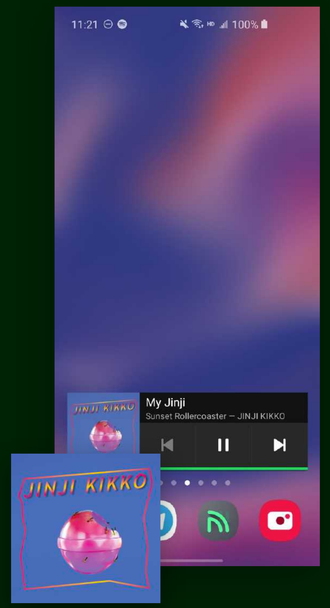
Diffuse is a live wallpaper that displays the current playing album in a ***blurred, twisted fluid-motion*** manner. Unlike Apple Music’s visualization, Diffuse works with pretty much any music players, Spotify, SoundCloud, Tidal, you name it. With additional audio permission, Diffuse can analyze ***audio spectrum*** and squeeze the fluid accordingly, giving the visualization a sense of ***pulse***.

SHOWCASE

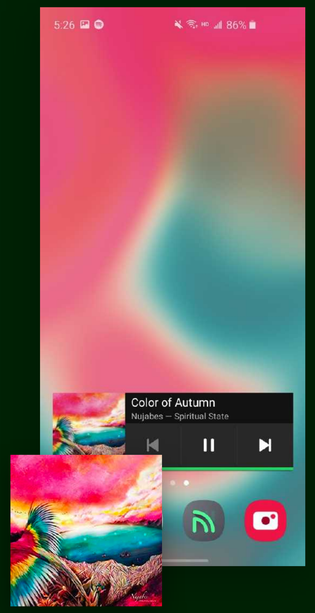




APPLE MUSIC



DIFFUSE WITH



SPOTIFY

RENDER PIPELINE

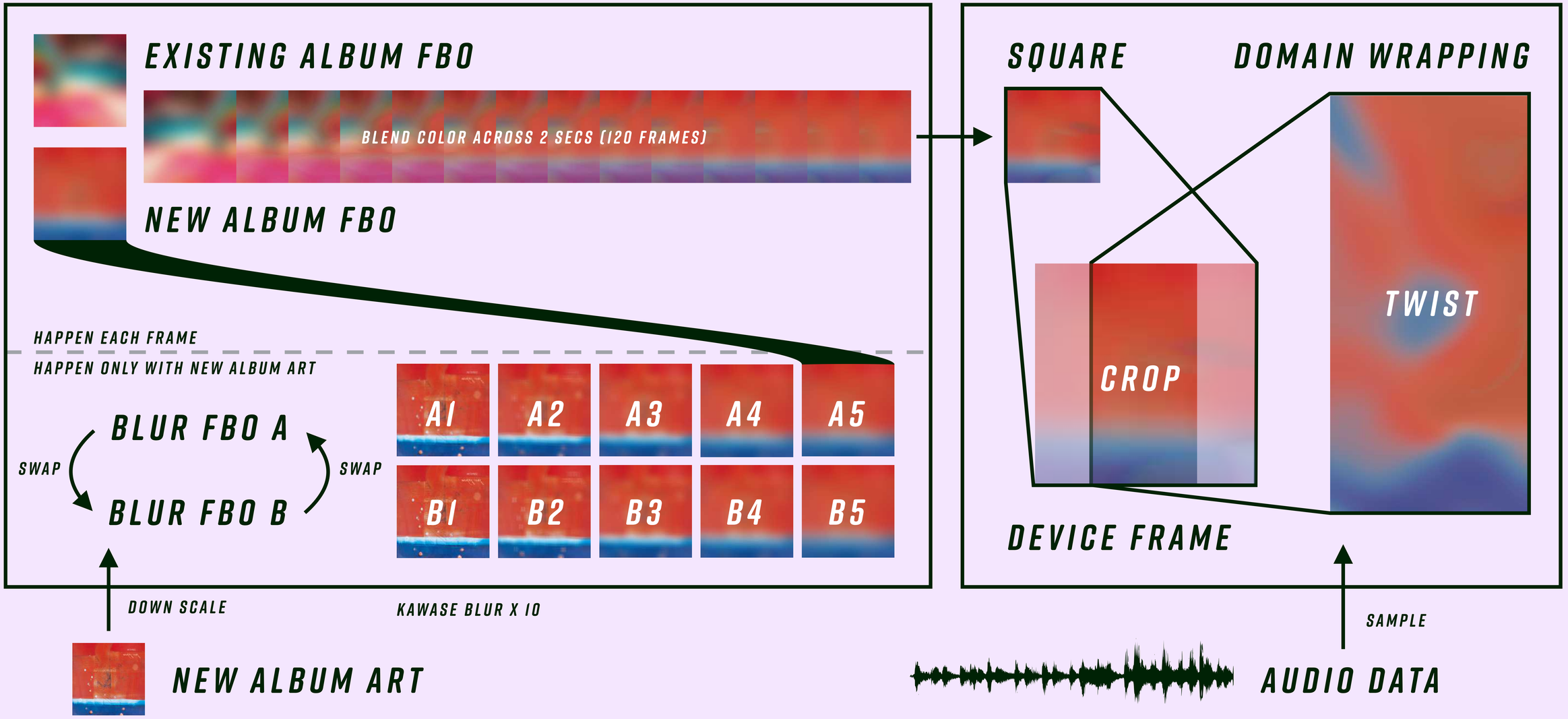
I picked ***LibGDX*** as the rendering framework for its lightweight design and simple APIs to manipulate frame buffers. To achieve a similar effect to Apple Music, an album art is first blurred with ***Kawase Blur*** kernels into gradients, then further twisted by ***Domain Wrapping*** noise functions into fluid-like blobs. From there another pass of blur is applied but with less radius to filter out any sharp edges, generating the final image.

BLUR & FADE

(1~11 PASSES, 1~2 MS)

NOISE

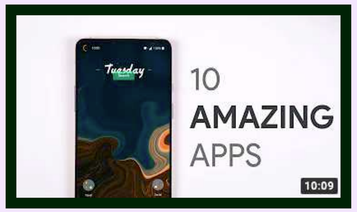
(3 PASSES, 2 MS)



Have I ever mentioned that Diffuse is popular among Android customization communities on YouTube? Because it is.

10 INCREDIBLE ANDROID APPS YOU NEED TO TRY! BY SAM BECKMAN

329K VIEWS

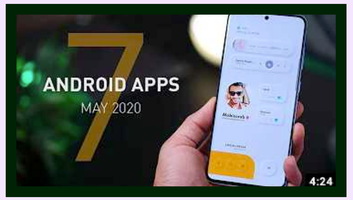


BEST ANDROID APPS - MAY 2020! BT HOWTOMEN

180K VIEWS

BEST ANDROID APPS - JULY 2020! BY ZACHARY ANDERSON

69K VIEWS



TOP 7 MUST HAVE ANDROID APPS - MAY 2020! BY MOBISCRUB

106K VIEWS

TOP 10 BEST APPS FOR ANDROID - FREE APPS 2020 (MAY) BY GADGET GIG

142K VIEWS

END

Live Wallpaper Series By Haotian Zheng

Additional Info: https://portfolio.justzht.com

NODE EDITOR DEVELOPMENT

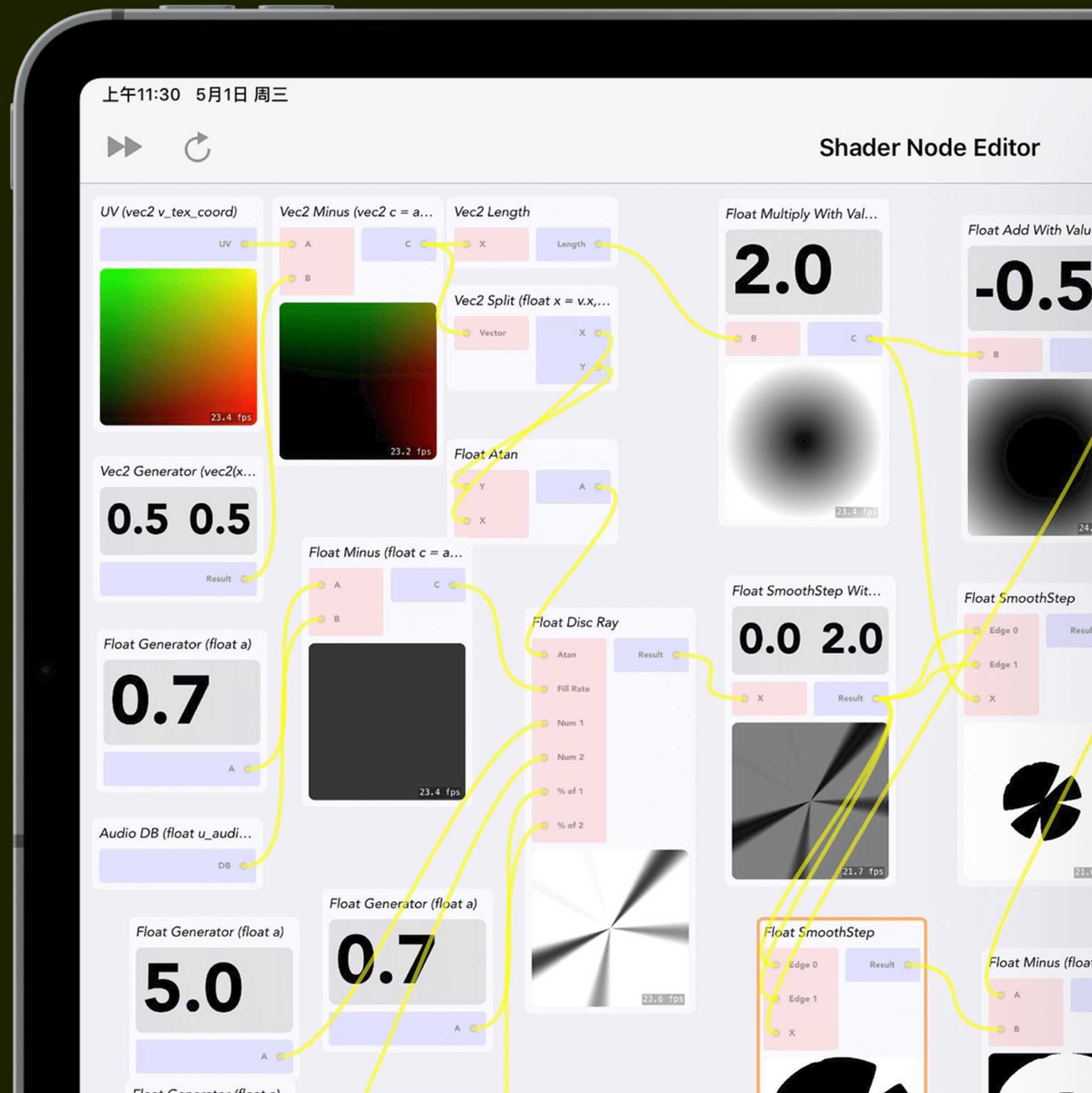
By Haotian Zheng

An interesting observation on mobile GUI patterns is that most of the time, we are dealing with lists or tables, and there isn't much to say about graph editing. Well search no more, meet ***Shader Node****.*

SHADER NODE

"APPLE WWDC 2020 STUDENT CHALLENGE WINNER"

Shader Node is a Swift Playground written in Mar 2019 and later improved in Mar 2020 that features my own implementation of the node editor framework using pure Apple frameworks for ***shaders creation***. With ***audio input nodes***, Shader Node turns your iPad into a portable creative VJ machine.



ORIGIN

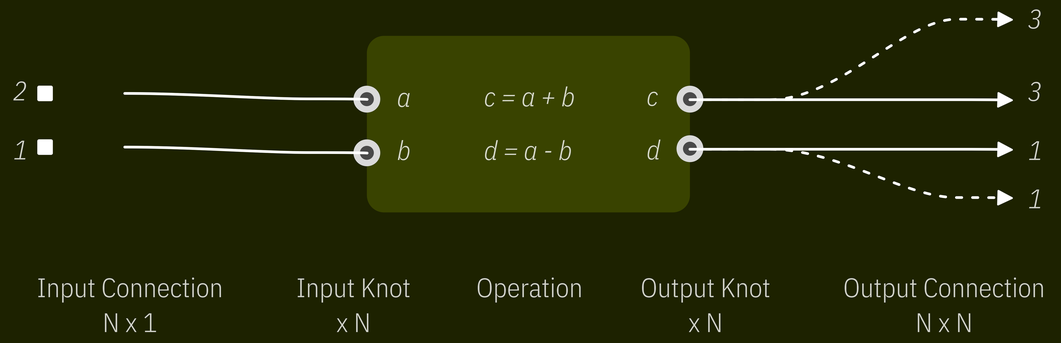
Node-based editor is a pretty common non-linear editing GUI pattern used in multi-media software, including Max7, Houdini, and Unreal Blueprint. It is worth noting that node editor is a broader term with several variations depending on specific purposes, and ***Shader Node is a value-based one***.

VARIATIONS

Trigger-based nodes are like synapses, they do not store temporary values but just pass the signal to any node connects to the output knot.

These kinds of nodes are usually used in dialog designers because although the current line is triggered by the previous spoken line, the content of the current is not affected by any of the previous ones.

Consequently, trigger-based nodes can have multiple connections to other nodes on both input and output knots, and it is more of an asynchronous system without the limitation in step or time.



Value-based nodes work differently. The output has specific value for the next node to consume.

This feature makes it suitable for creative graphical coding, as program executions not only follow control flow but also require parameters.

Because the value on the input knot needs to be certain at any time, the input knots can only be connected to one output knot a time. Typically value-based nodes are synchronized, meaning the different timings on input does not affect the output value.

INTERFACE

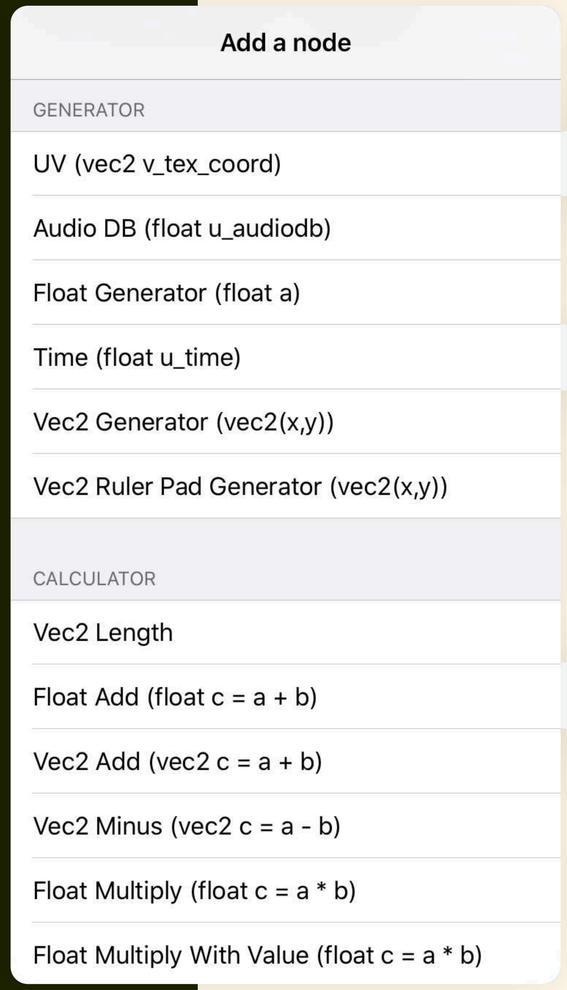
Shader Node, along with the internal framework, was written from the ground up with only UIKit Dynamics and SpriteKit to achieve a full-fledged node editor experience.

Every node in this list is a subclass from a base node model that contains enough information to construct the node view.

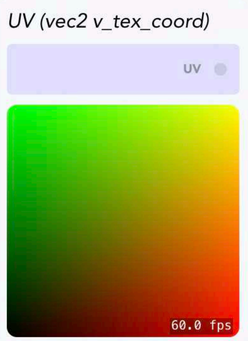
```
@objc public class NodeData: NSObject {
    var inPorts : Array<NodePortData>
    var outPorts : Array<NodePortData>
    var previewOutportIndex : Int
}

public class NodePortData: NSObject {
    var title : String
    var connections : Set<NodeConnectionData>
    var requiredType : ShaderDataType.Type
}

public class NodeConnectionData: NSObject {
    var inPort : NodePortData!
    weak var outPort : NodePortData!
}
```

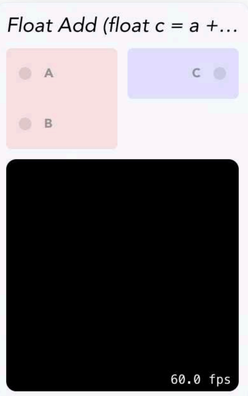
Then Shader Node collects all types of nodes in runtime by invoking objc_getClassList.

NODE LIST



NODE UI





MACHINE GENERATED CODE

```
uniform vec2 u_resolution;
vec2 uv = gl_FragCoord.xy/
u_resolution.xy;
vec2 node_x_out_0 = uv;
```

```
uniform float u_time;
float node_x_out_0 = u_time;
```

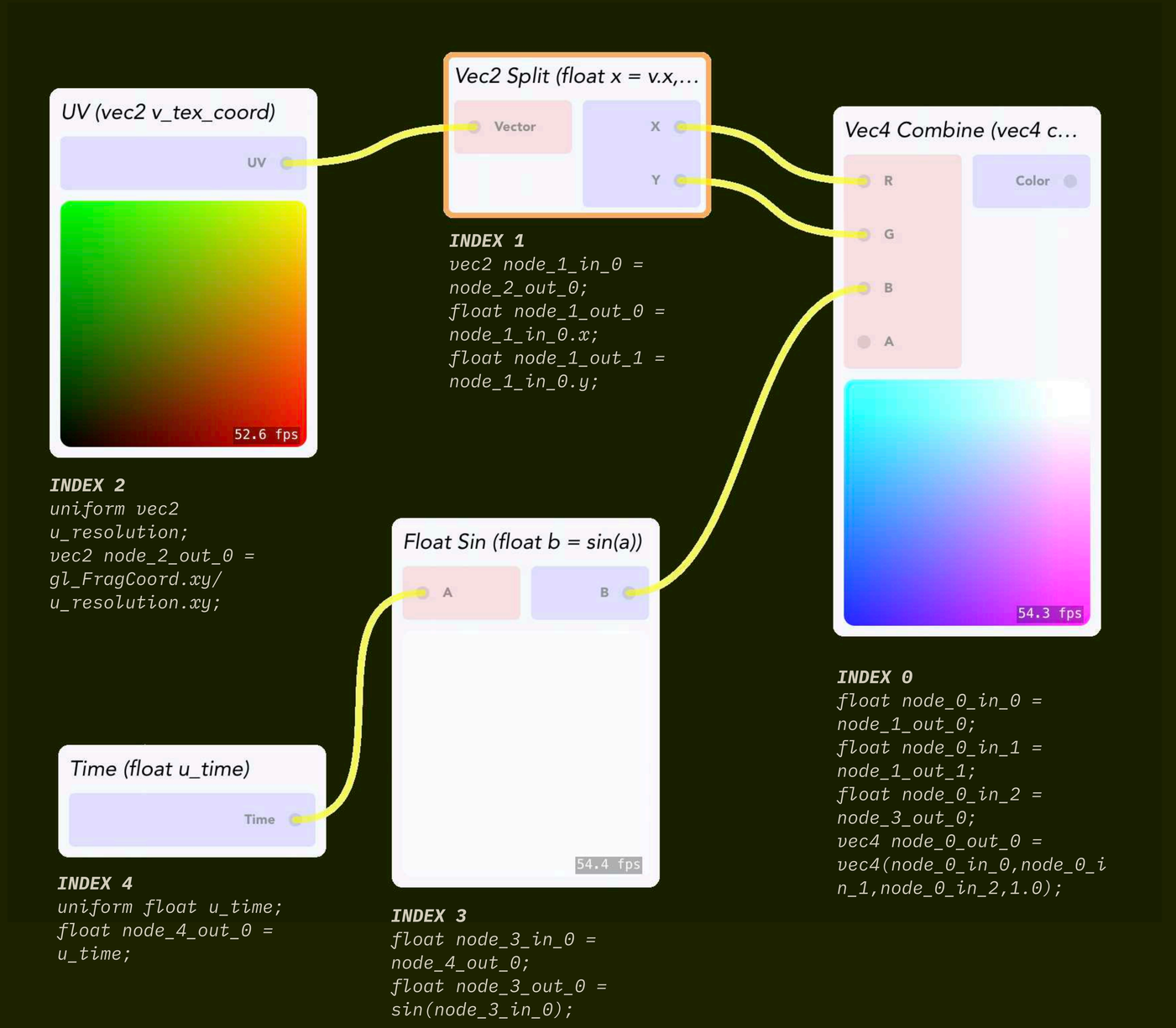
```
float node_x_in_0 = 0.0;
float node_x_in_1 = 0.0;
float node_x_out_0 =
node_x_in_0 + node_x_in_1;
```

EXECUTION

Shader Node uses a 2-pass approach that isn't so efficient but at least is reliable. Given a node graph representing shader code as below:

```
uniform vec2 u_resolution;
uniform float u_time;

void main() {
    vec2 uv = gl_FragCoord.xy/u_resolution.xy;
    float sint = sin(u_time);
    gl_FragColor = vec4(uv, sint, 1.0);
}
```

In the 1st pass, Shader Node would search the graph, collect linkage information, and build up a trace path for each node. In the 2nd pass, Shader Node would declare variables for each knot on nodes and equal operations on linked knots, and finally, append the code in order of inversed trace path.

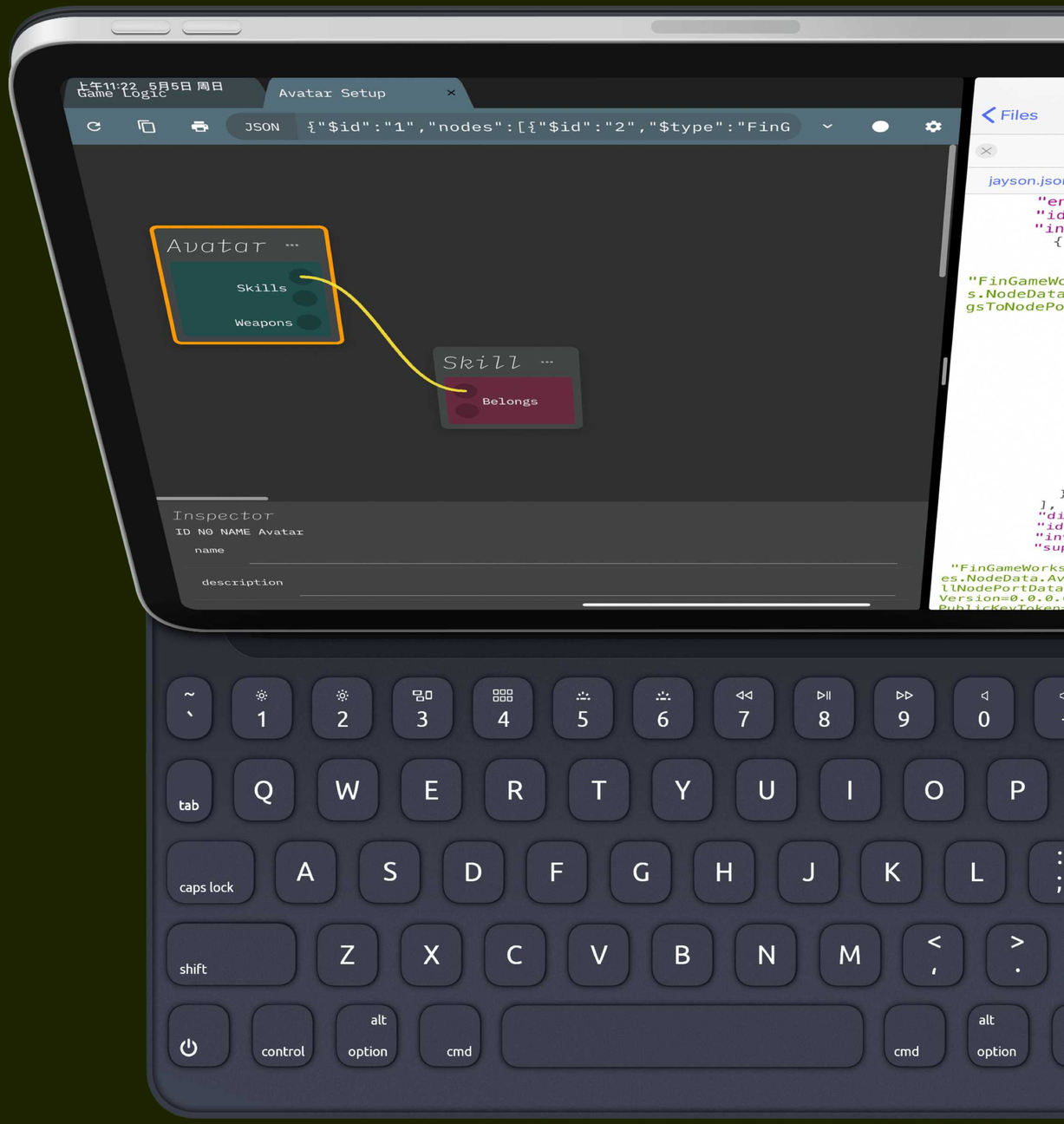


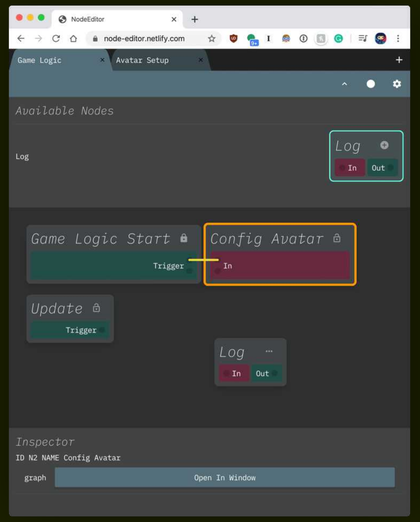
After I finished Shader Node, I went to a Unity event and talked to the Shanghai R&D team. And I thought, why not leverage the cross-platform game engine to make a no-code IDE from my node editor framework? Meet ***ScriptNode****.*

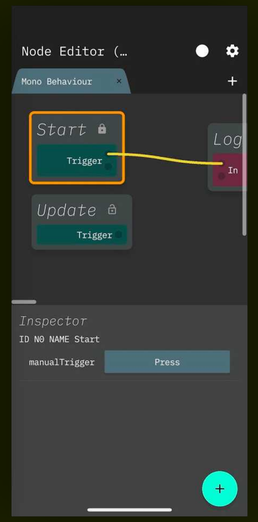
SCRIPTNODE (WIP)

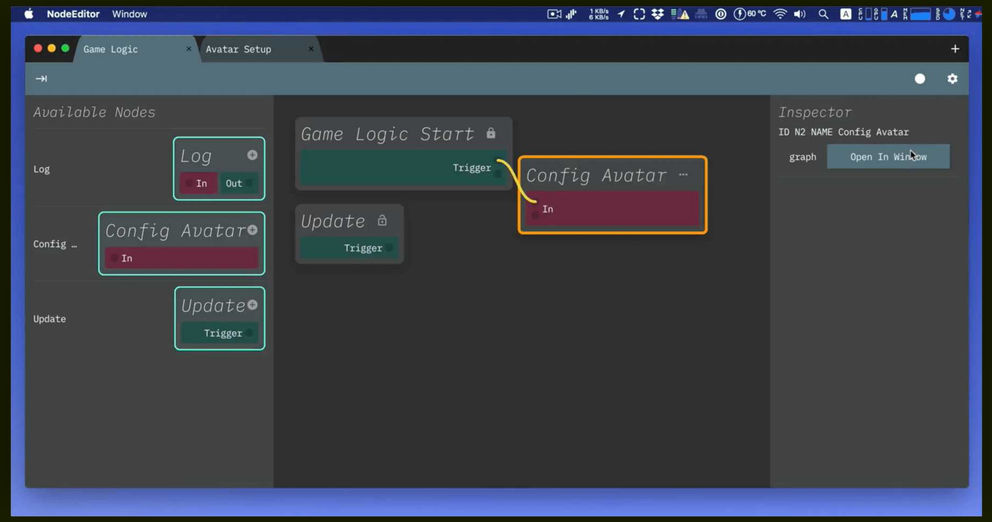
"DEVELOP GAMES WITHOUT CODING"

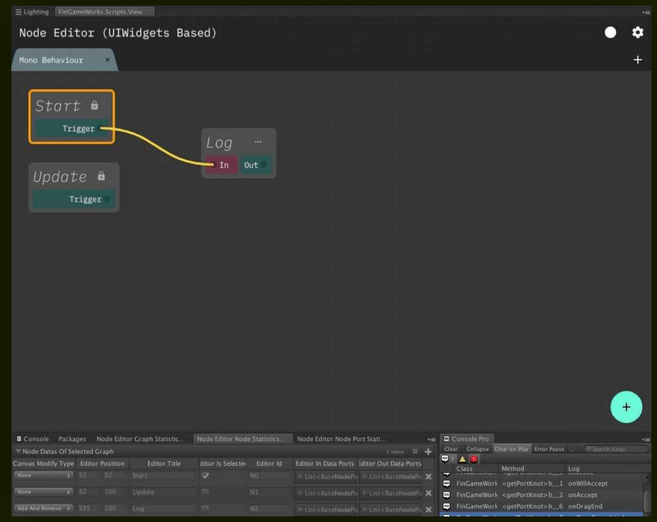
ScriptNode is a cross-platform visual programming IDE initialized from Apr 2019. With a node-based GUI written in UIWidgets, a C# port of Flutter from Unity Shanghai, ScriptNode is capable of creating custom monobehaviors from node canvases and mounting them to GameObjects, essentially ***replacing*** the coding work previously needed in Unity game development.











WEB MOBILE DESKTOP UNITY PLUGIN

The development of ScriptNode is now paused, but it will resume later. As of Mar 2019, ScriptNode (***Web Demo***) was able to run on those platforms as well as handle basic Unity operations, including logging, transform editing, and monobehavior callbacks. I also included some wrappers for typical game elements like a data model for avatars, which would ***help beginners make games*** in the ***no-code*** setting.

END

Node Editor Development By Haotian Zheng

Additional Info: https://portfolio.justzht.com

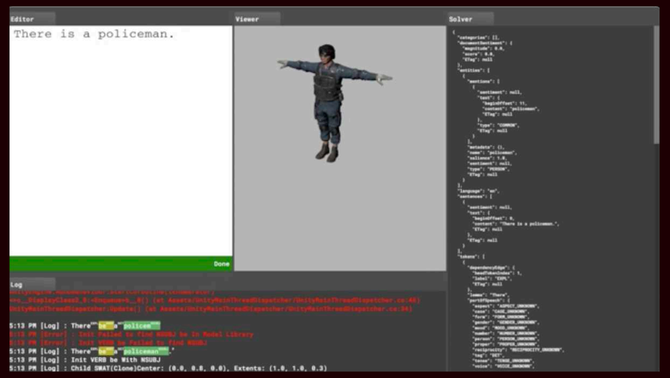
MISCELLANEOUS COLLECTION

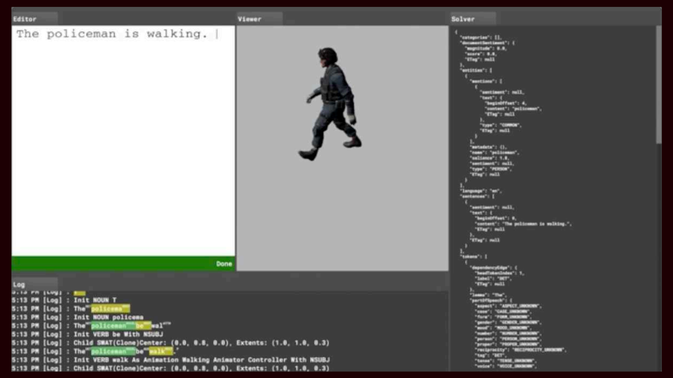
By Haotian Zheng

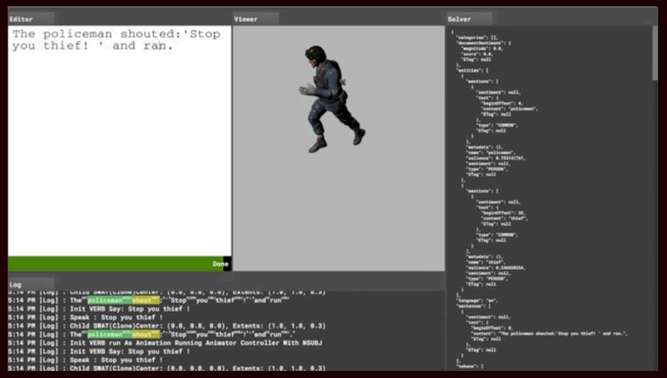
This section contains some small personal projects and company projects that are game-related as supplementary materials. For projects other than games, please visit my ***portfolio***.

SCREENPLAY REPL

As chief engineer and co-founder of ***rct studio***, I developed the prototype of the Screenplay REPL in Dec 2019 as a module of our in-house Morpheus Engine. The prototype analyzes input texts and generates corresponding 3D animations using Unity and Google Natural Language API.

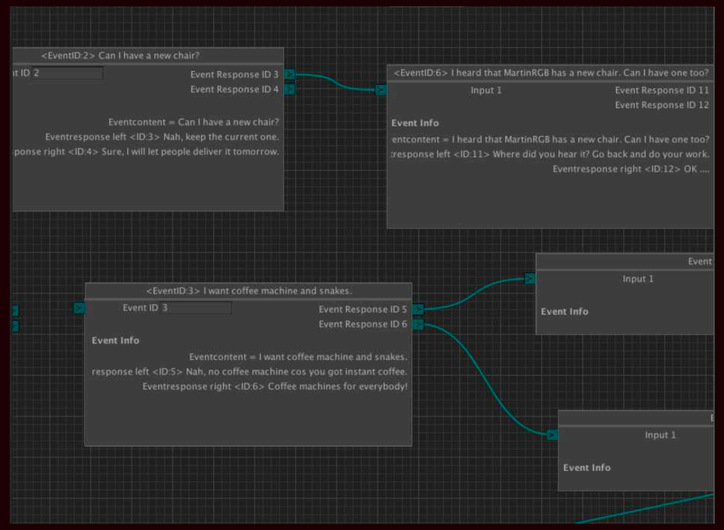


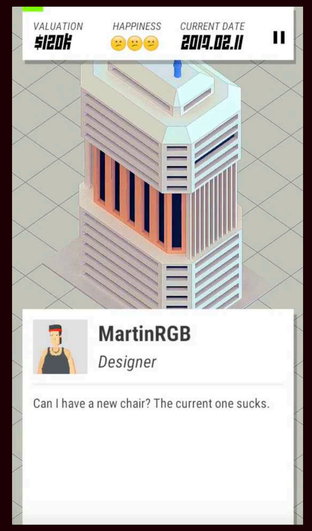


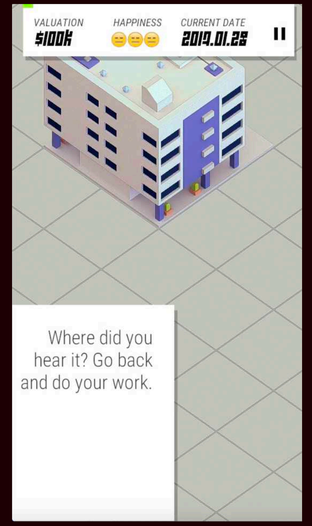


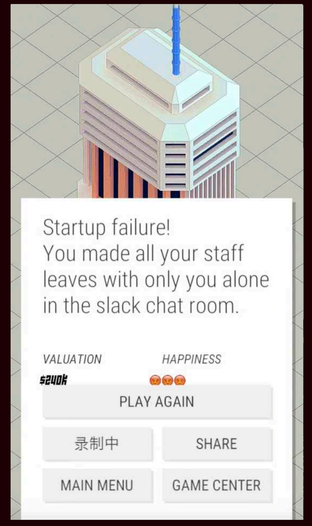
CHINA STARTUP SIMULATOR

China Startup Simulator (made in Jan 2017) is a casual game where you play as a startup entrepreneur, replying to questions that would have unexpected effects on the valuation and happiness of staff. This game was heavily inspired by card-swiping games like Reigns.



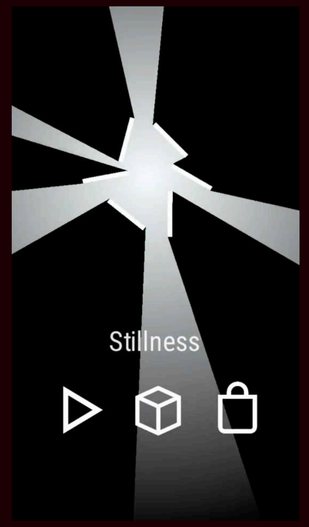


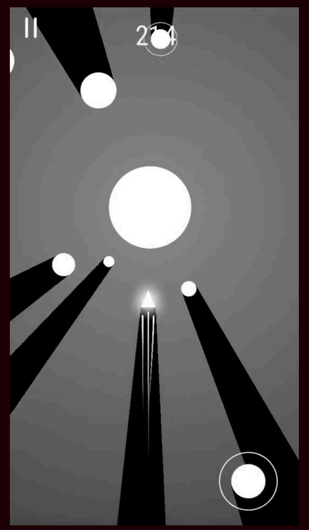


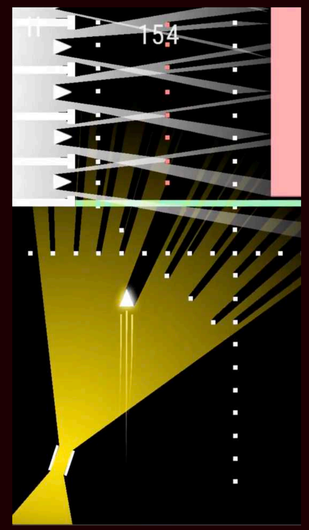


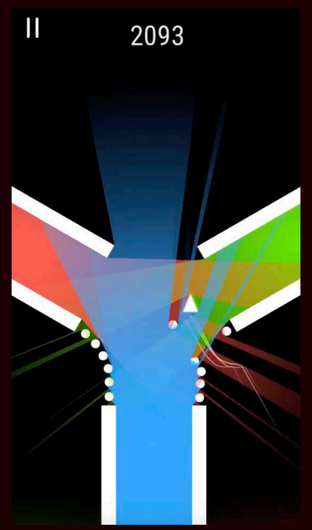
STILLNESS

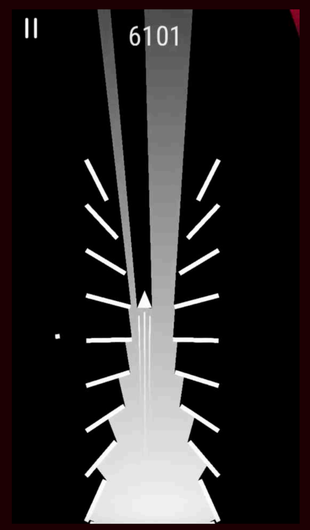
Stillness (made in June 2015) is a game for me to experiment with the long-shadow art style.











END

Miscellaneous collection By Haotian Zheng

Additional Info: https://portfolio.justzht.com