X-Force 2021 Summer Fellows, Team Navy 5

Hiroshi Furuya, Victor Maciel, Austin Park, Jonathan Taylor, Kate Riordan

08/10/21

Welcome to

Visible Horizon

This document describes what is needed to get started working on *Visible Horizon*

Table of Contents

Who this is for

This document is targeted towards students who may be asked to continue development of our project. Such a student may be relatively new to Unity, and perhaps to coding in large projects altogether. This student is expected to know how to code, and will be expected to figure things out, as this document is not meant to be a guide to how to work in Unity or code in general.

Background

I-STALKER and the training simulator

Sailors operate the AN-SAY3 I-STALKER EO/IR sensor platform aboard aircraft carriers around the world, providing critical visual range sensor information--but they only use the most basic feature and receive no formal training. We utilized the same type of cutting edge human-centered design processes as innovative private companies to identify where we could help. We conducted interviews with I-STALKER engineers and reviewed interviews by a prior H4D team with civilian and military training experts to understand the situation. Then, we held iterative design sprints and stakeholder feedback sessions to identify pain points and value creators. This led to our formulation and implementation of the I-STALKER virtual simulator training game, designed to provide sailors flexible, engaging, and scalable training in varied target-rich environments.

For more information, see the quad chart, slide decks, or promo video included with this document.

Key terms and concepts

I-STALKER refers to the sensor platform itself; the color camera, IR camera, and laser rangefinder. I-STALKER is not the name of a user interface (UI). Technically, the UI that operators see is a version of the EBRISS system. EBRISS is a software platform that supports many different sensor platforms, of which the I-STALKER is one. The EBRISS UI contains controls for the I-STALKER, displays radar contacts, shows the local map, and more-- but it does not contain the video feed from the I-STALKER itself! For I-STALKER systems, the video feed is not sent to the computer running EBRISS/the operator station at all. Instead, it is usually sent to a separate TV or computer monitor usually mounted alongside the operator station. In other implementations of EBRISS for other sensor systems, the video feed may be sent to the same computer running EBRISS, in which case the video feed can be seen in a different application window. This configuration is what our game simulates.

What *Visible Horizon* does

Visible Horizon is a limited scope virtual simulator that is designed to give operators a chance to familiarize themselves with the I-STALKER system in target-rich environments. Currently, *Visible Horizon* and its source Unity project features:

- Simulated IR video feed on a detached window
- EBRISS UI visuals with the following features:
  - Moving radar contacts
  - Azimuth, elevation, and zoom indicators
  - Laser rangefinder simulation
  - Map zoom and rotate
- Virtual ocean environment populated with ships and boats of various sizes
- Unity scenes using new 3D assets demonstrating varied mission environments
- Game menus and user interfaces
- Scoring system based on a game timer and points for finding the range to objects in the scene

The Navy 5 team

Visible Horizon was started in the Summer of 2021, while COVID-19 restrictions were in effect. All work was conducted remotely with no in-person interaction with teammates or with NSWC Crane.

Team lead:
Hiroshi Furuya, Computer Science PhD student, UCF [hfuruya@knights.ucf.edu]

Team members:
Victor Maciel, Informatics B.S. Student, UW [victor88@uw.edu]

Austin Park, Tech Squad Volunteer / Security Analyst, [awppark1@protonmail.com]

Kate Riordan, Cybersecurity and Global Policy B.S. Student, IU

Jonathan Taylor, Informatics B.S. Student, IUPUI [jonmtayl@iu.edu]

> If you have questions, we may be available for contact to discuss further.

Project sponsor:

Siddharth Maini [siddharth.maini.civ@us.navy.mil]

This document was largely written by Hiroshi Furuya.

Project process

Project Process

We did not assume from the beginning that our project would be a game. As described in the background, we followed a human-centered approach to arrive at the conclusion that our limited scope game would be an appropriate product for the I-STALKER program. Most of the exercises we followed can be found at the Google Designs Sprints Kit website.

Meetings schedule

We had daily meetings with team members to go over what we had done the previous day and what we want to get done the rest of the week. We had weekly meetings with our project sponsor to update him and get his thoughts on our work.

Getting started

This describes what we used and generally how things were set up, so that you may do the same if you wish. If you struggle, usually you can Google/Youtube things because the tech stack of Unity + GitHub + VSCode/Rider is extremely common.

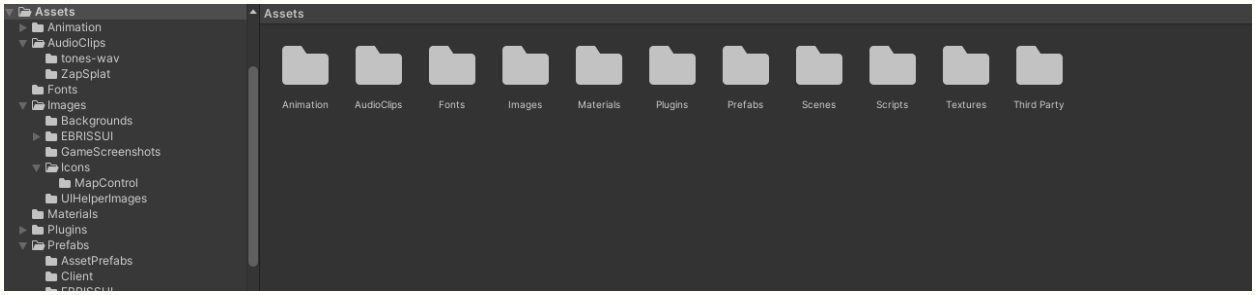
Software

- Unity Hub
- Unity 2019.4 LTS (through Unity Hub)
- GitHub Desktop
- IDE for writing Unity C# scripts (Recommendations: VS Code, Jetbrains Rider -free for students-)
- See the Student Software Install and License Instructions document for detailed instructions on how to get Unity and GitHub Desktop setup.

Working with the Unity project

Project structure

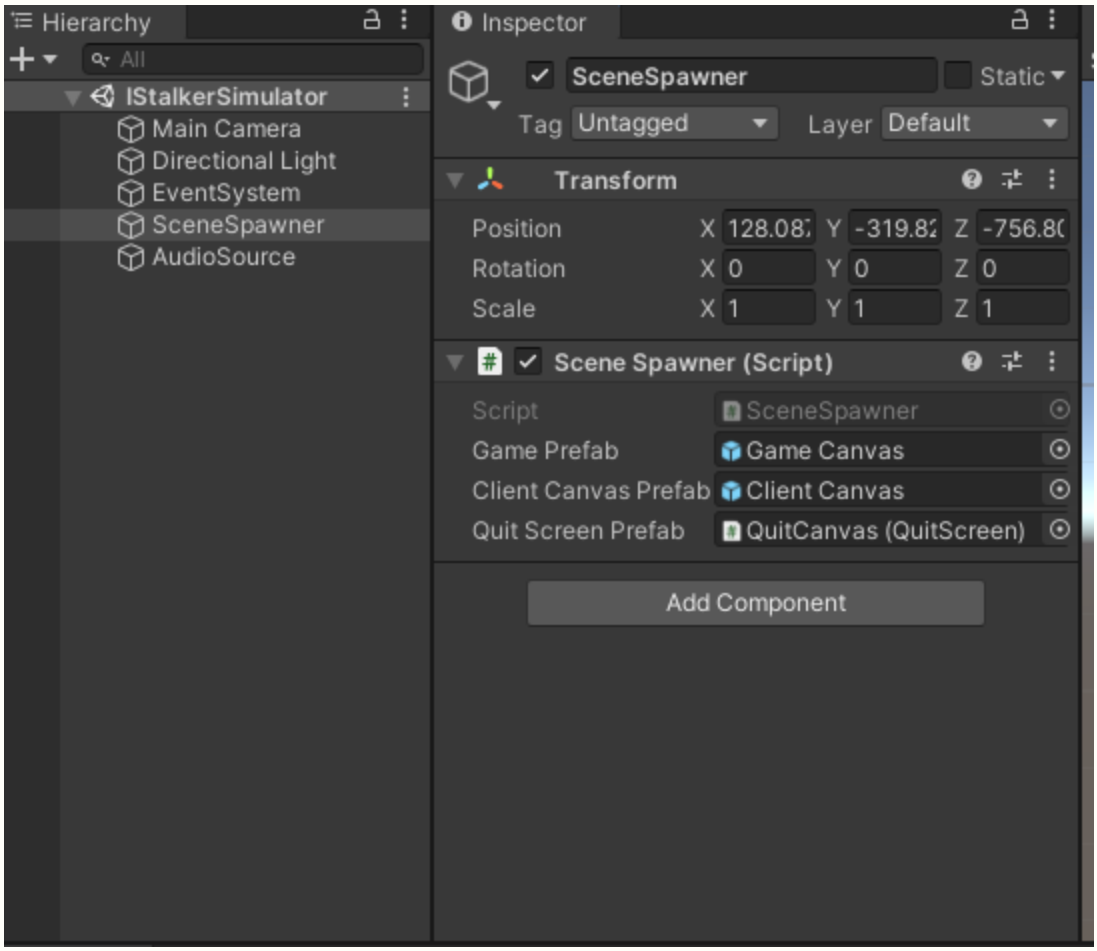
Unity projects are managed from within Unity itself. Pretty much all content, files, etc. that are used to develop the product are contained within the Assets folder. Scripts we write *all* go into the `Assets/Scripts` folder. I would recommend you follow this approach. Likewise, prefabs, images, audio clips, etc. *all* go into their respective folder.



Our project follows a prefab-based approach. Essentially, there is a single scene with a single main script that loads prefabs, which themselves have scripts that load other behavior. Most changes you will make will be to these prefabs.

Code execution

Specifically, `SceneSpawner` exists in the scene already. On Start, it instantiates the Client and Game canvases.



"Client" refers to the menus and screens other than the simulation itself. "Game" refers to the simulation. If you open `SceneSpawner.cs`, you can see this behavior in detail. `SceneSpawner.cs` initializes instances of `GameSwitcher.cs` and `ClientStateSwitcher.cs`. Each script itself similarly initializes many other scripts.

What new features look like

Typically, when you write a new feature for the game, it will require information from other objects. Take `MapRadarContactTracker.cs`, a Game function, for example. It requires information about the UI where the map is shown as well as information about the radar contacts it shows. These dependencies are usually provided through `GameSwitcher.cs`. In general, `ClientStateSwitcher.cs` and `GameSwitcher.cs` serve as 'hubs' where other features can be wired together. This results in `ClientStateSwitcher.cs` and `GameSwitcher.cs` getting longer and longer while other scripts generally stay within 50 lines or so. In fact, almost every script in our project has references that eventually trace back to either one of these two scripts.

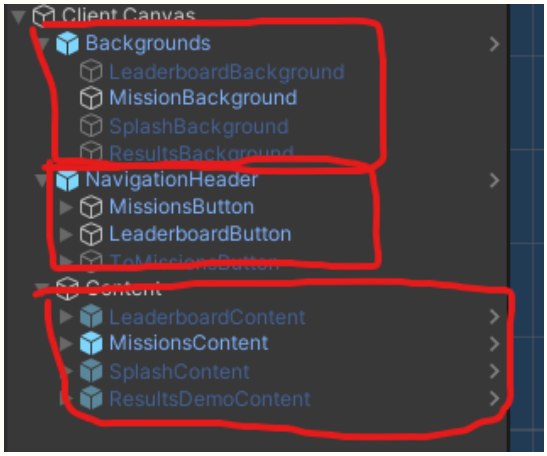
> Every modern IDE has a 'follow reference' feature where you can see where instances of a class or its members (properties, methods, etc.) get used elsewhere in the project.

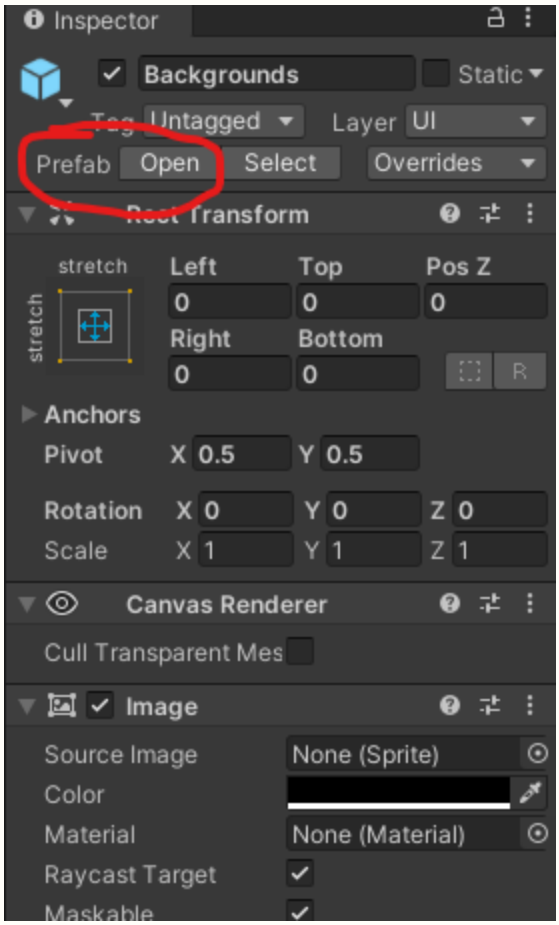
Usually, when you create a new feature, there is a visual component and a behavior component. In our project, we typically have scripts that are nothing but reference containers to Unity visuals, e.g. `ContactTableItem.cs`. These visuals are referenced by scripts that define user interface behavior. For `ContactTableItem.cs`, the corresponding 'behavior' script would be `ContactsTable.cs`.

The Client prefab

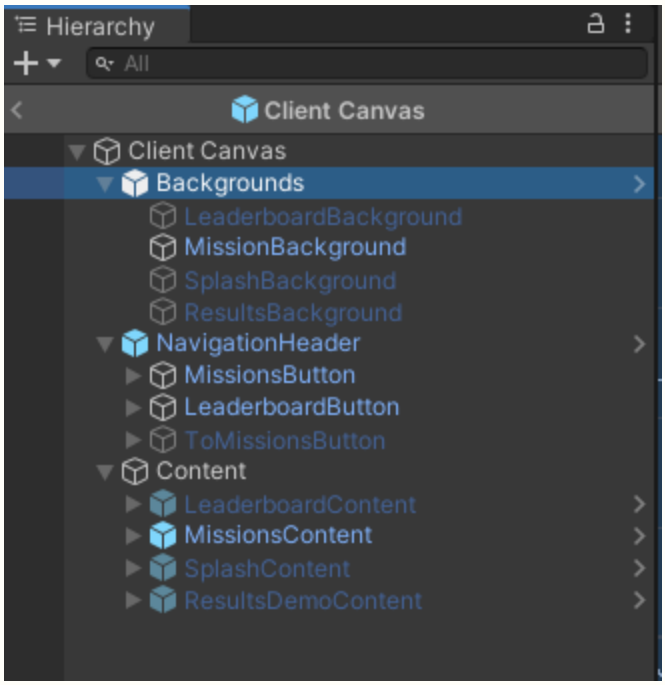
`Assets -> Prefabs -> Client -> Client Canvas`

The Client prefab is further split into three groups of prefabs: The Backgrounds prefab, the Navigation Header prefab, and the Content prefabs.





Make your changes by clicking on whatever prefab you want to change in the Hierarchy and click Prefab: Open

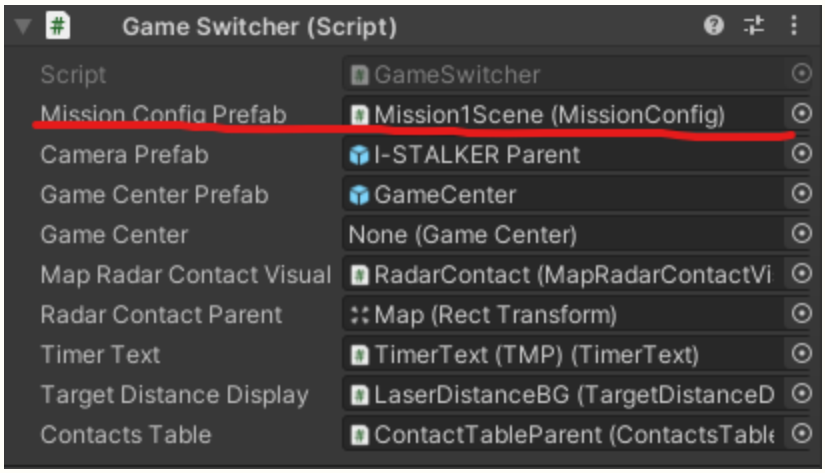


The Game prefab

`Assets -> Prefabs -> Game -> Game Canvas`

The Game Canvas prefab is split into two prefabs: the EBRISS UI Frame and the Video Parent. The former contains the elements used to simulate the EBRISS UI, and the latter is used to display the video feed.

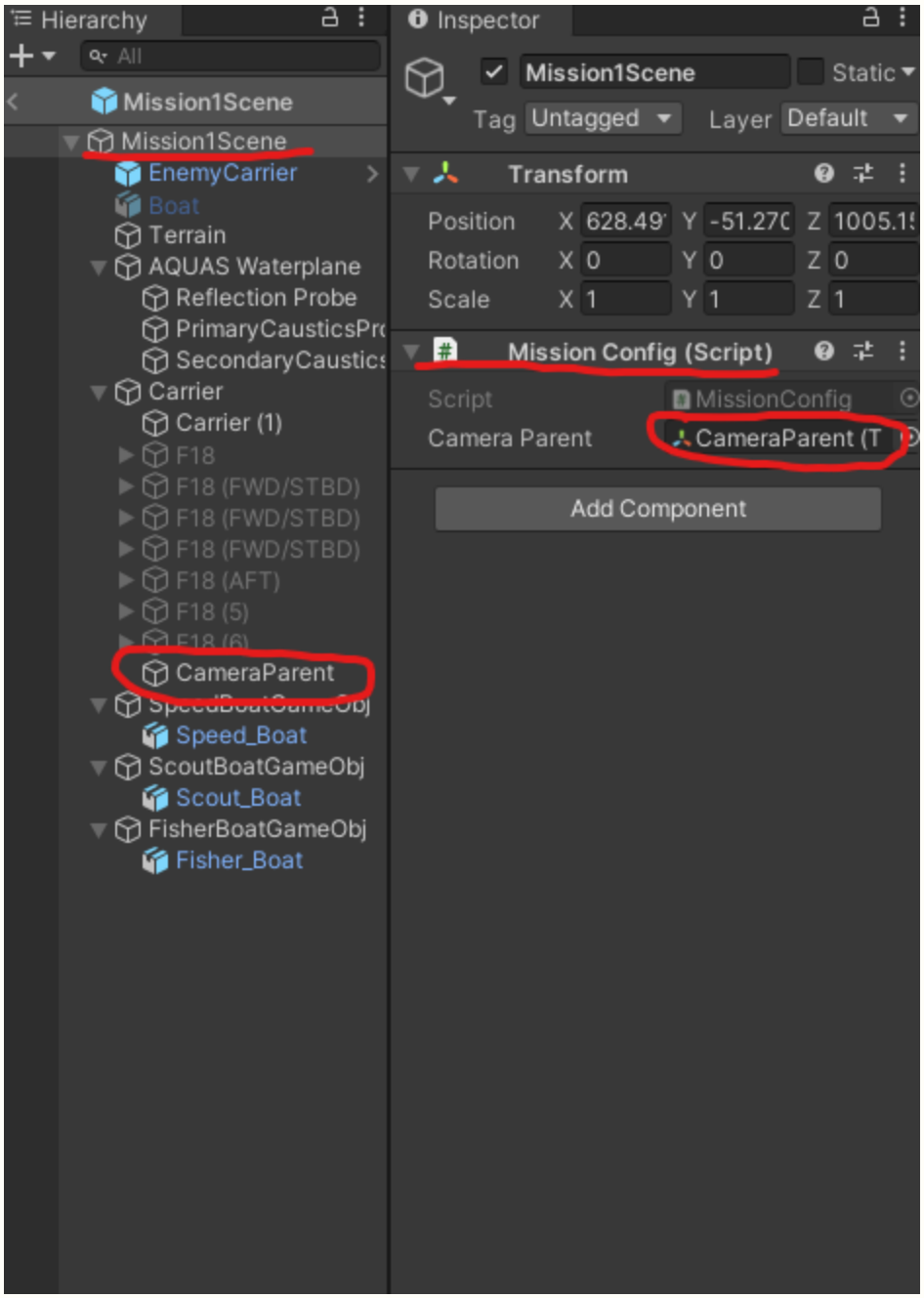
A third prefab is also used here: the Mission Config prefab. On the Game Switcher component attached to the root of the Game Canvas prefab, we can see there is a reference to “Mission Config Prefab.”



That prefab is what determines what the virtual environment looks like. In order for a prefab to be eligible to be assigned here, it must have a MissionConfig component attached to the root with a Camera Parent Transform assigned.

Creating a mission scene

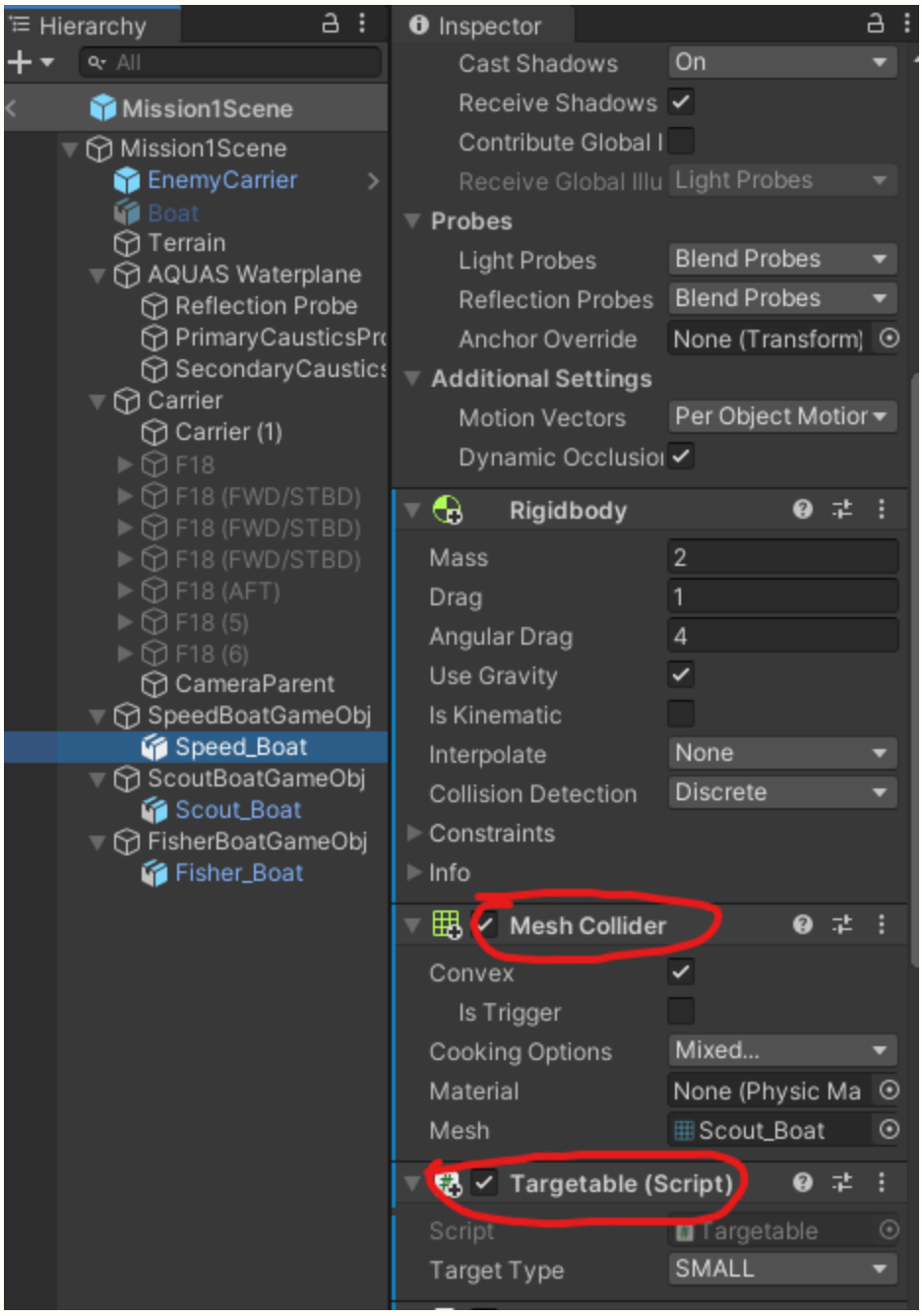
A mission scene is any prefab that has a Mission Config component attached to the root GameObject of the prefab with a Camera Parent identified.



See the `Assets -> Prefabs -> Game -> MissionPrefabs -> Mission1Scene` prefab for an example.

CameraParent determines where the simulated I-STALKER camera will be placed.

In the game, players can point at objects in the scene and get the range to that object. In order for Unity to know that an object is targetable, it must have a Collider and a Targetable attached to it.



The RigidBody and Buoyancy components attached to the boats in the scene are there to provide the floating behavior, but technically don't have to do with the ability to be targeted by the I-STALKER. Whenever you add a new 3D object to a mission, making it float is a very touchy process involving experimenting with the various parameters on the Buoyancy component attached to that particular object.

Sounds

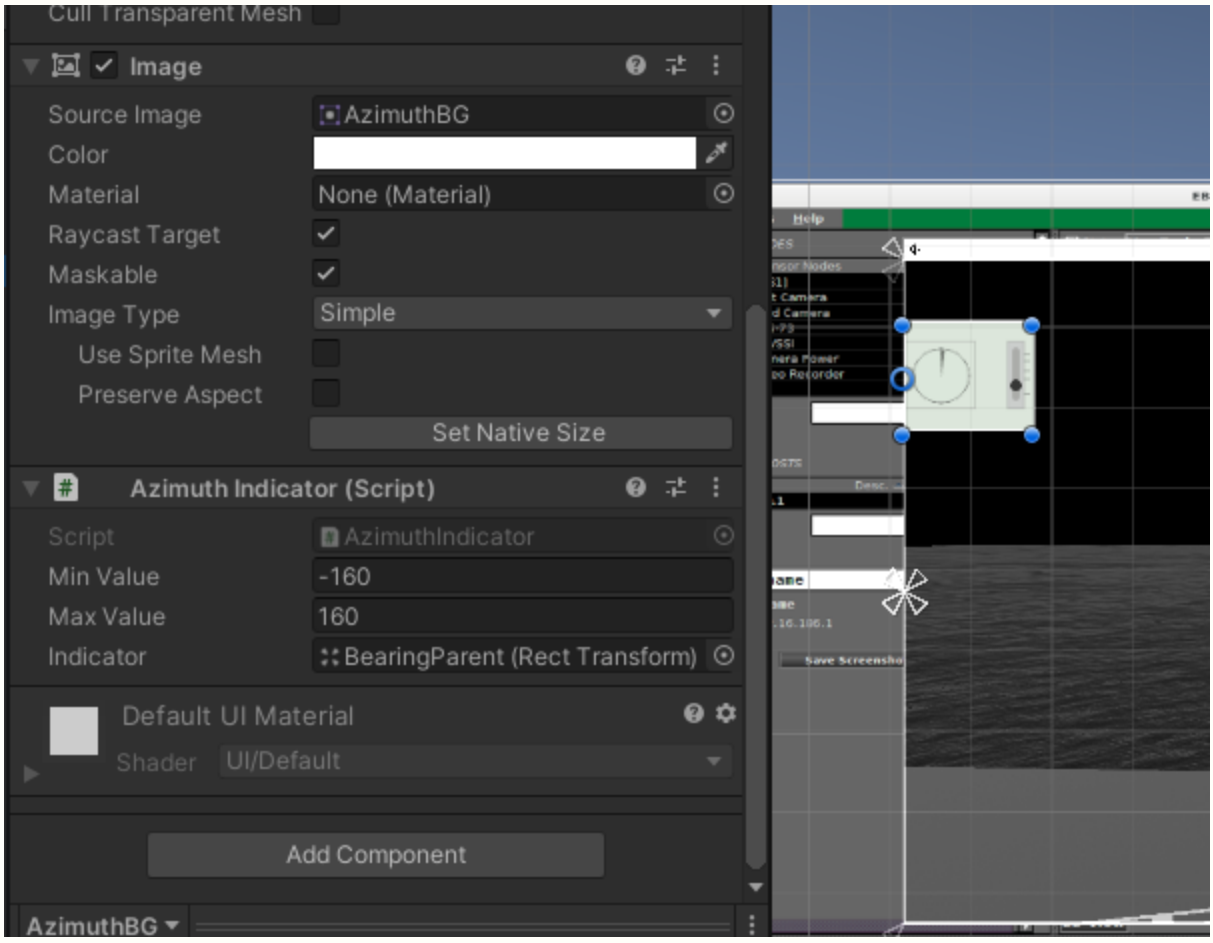
UI sounds, of which usually only one will play at a time, is handled through `AudioPlayer.cs`. Other sounds, such as the music played in the menu screens, is generated through other scripts and AudioSources, as demonstrated by the use of `AutomaticBackgroundMusic.cs` (seen in the Background prefab, `Assets -> Prefabs -> Client -> Backgrounds`).

How does xyz work?!?

When in doubt, go into Play Mode, click into scene mode, and click on whatever is doing the thing that you're wondering about. Usually, the script responsible for the behavior is attached to the GameObject exhibiting the behavior.

Example: How do the azimuth and elevation indicators work!?

Answer: Click on it in the scene mode and you'll see that there's an instance of AzimuthIndicator.cs attached to it.



Future targets for development

	Function	Feasibility/Status
1	More EBRISS UI functions	In-progress
2	Additional missions, environments, realistic 3D models	In-progress
3	Headless mode for use as environment simulator with real EBRISS system	Line-of-sight (weeks)
4	Offline, non-networked multi-computer multiplayer mode	Line-of-sight (weeks)
5	Support for wired controllers	Line-of-sight (days)
6	Mission authoring by end users	Feasible (months)

For 1, see the document titled "EBRISS UI functions," which enumerates the many features of the EBRISS UI that we want to implement.

For 2, see `Assets/Scenes/NewEnvironmentScenes` for .unity scenes that demonstrate ideas for new mission prefabs.

Resources

Documents

- This document, and others included with it
- Slide decks, posters, and videos from Navy 5's presentations to X-Force and stakeholders
- Hacking4Defense final slide deck + interview notes
- Navy 5 design artifacts
- Email records from team lead Hiroshi Furuya
- US Navy Brand Guidelines

People

- Navy 5 team members and project sponsor (see Team)
- Ben Peterson, NSWC Crane engineer who has served as I-STALKER installer and trainer ben.peterson@navy.mil

- Keith Milhouse, NSWC software lead who knows about the EBRISS software platform that is used with I-STALKER
  keith.milhouse@navy.mil

Tech support

- Google- one advantage to Unity and C# is the huge amount of content on the internet surrounding questions and answers to using Unity and coding in C#. If you have questions on how to do something using Unity, I usually Google what my problem is plus the word “unity.” For coding questions, usually the problem is actually independent of Unity, so I would Google what I’m trying to do plus the word “C#.”
- Unity tutorials are very mature and well-supported. They are an excellent way to start. For reference, it took the team lead, a previous professional Unity developer, several months as a student to learn Unity. Following the purpose made Unity tutorials may take days or even weeks, but will likely be more effective and efficient than figuring it out yourself