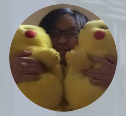
Jeff Salvage, CCI Advisor



Laura Mo *Team Lead*

Colan Biemer *Programming Lead*

Gabby Getz *Programmer*

Shreya Patel *Programmer*

Jasmine Marcial *Programmer*

Alliy Frauenpreis *Programmer*

Dr. Frank Lee, Stakeholder & DIGM Advisor

Matt Bodner *DIGM lead*

Dave Petersen *Artist*

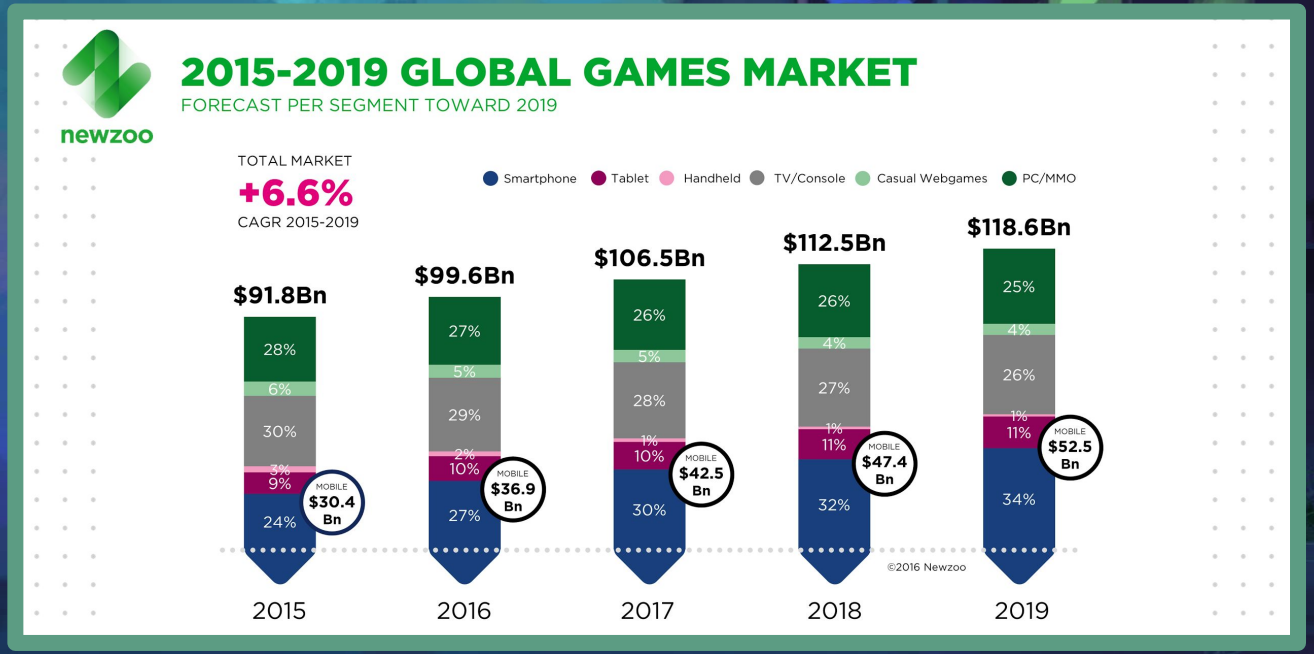
Michael Rodriguez *Artist*

Keano Jan Osmillo *Artist*

Andrew Mylet *Artist*

Joseph Santos *Artist*

Project Motivation



HIGHWATER IS

A roguelike survival game

Survival

Manage health, warmth, and hunger

Permanent death

Procedural Cityscape

Different environment every playthrough

Intuitive Crafting System
Multipurpose and
dynamic items

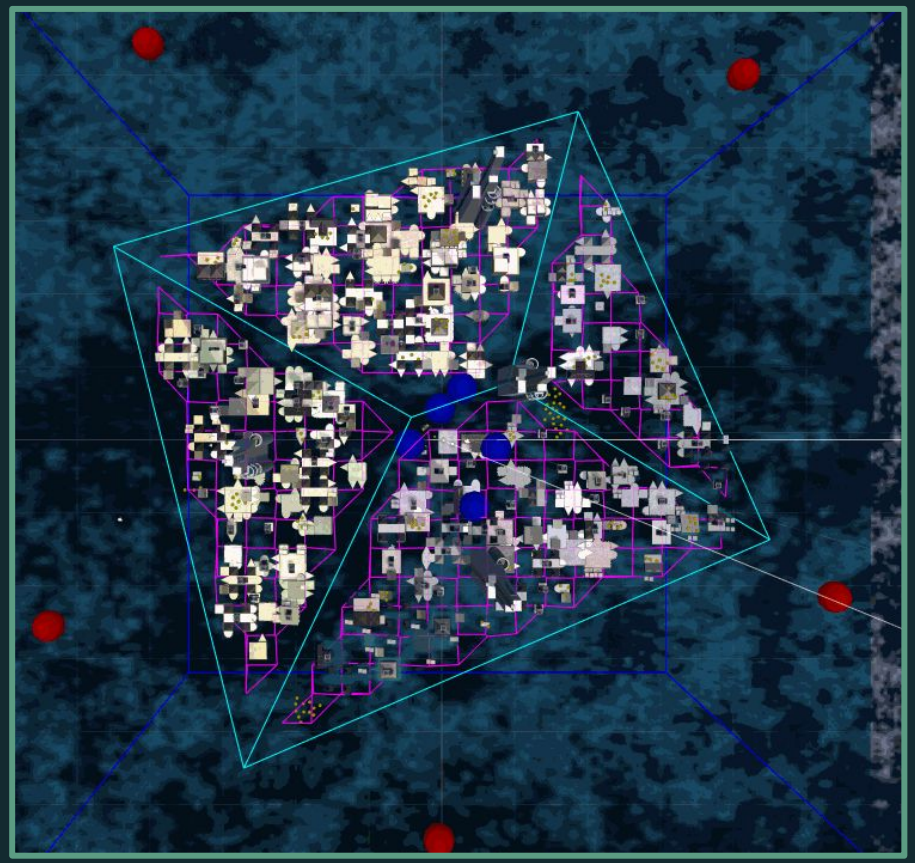
Discoverable Story
Integrated into the environment
and radio system

Simulated Weather
Realistic weather system the player can predict
City slowly filling with water

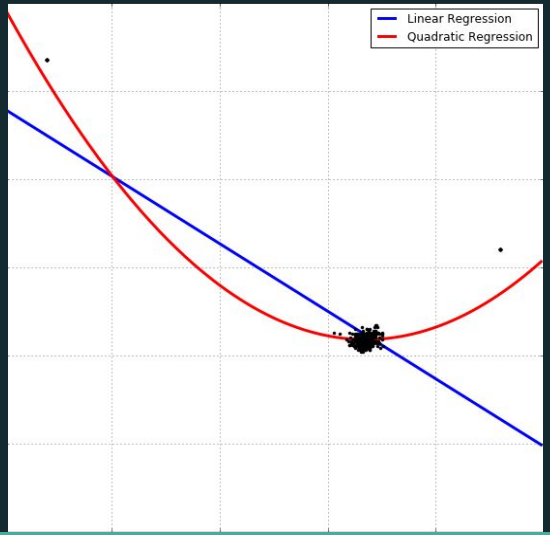
Weather System Comparisons

	Call of Duty	Banished	Highwater
Precipitation	✓	✓	✓
Water	✓	✓	✓
Temperature		✓	✓
Weather Based Events		✓	✓
Wind Speed			✓
Based on Real Data			✓
Dynamic Pressure System			✓

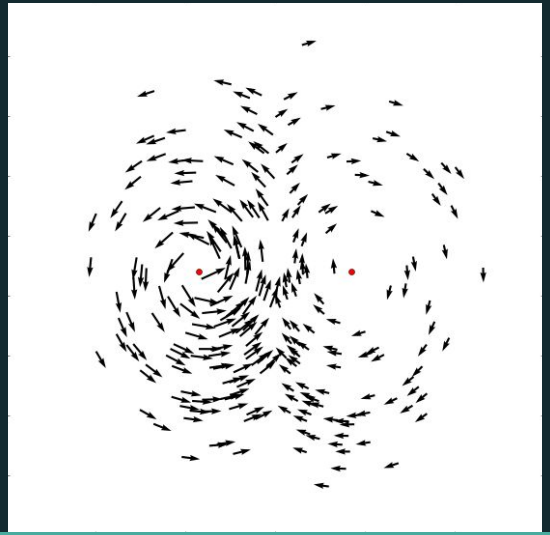
Pressure Systems AND weather



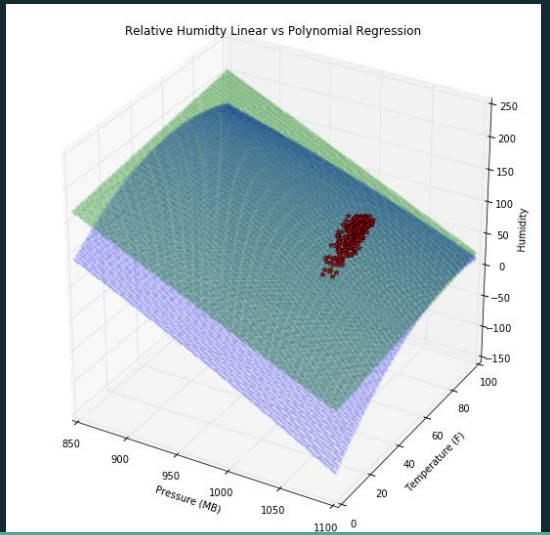
Weather System



Sea Level Pressure to Wind Speed



Wind Speed Direction Compared to Pressure Centers



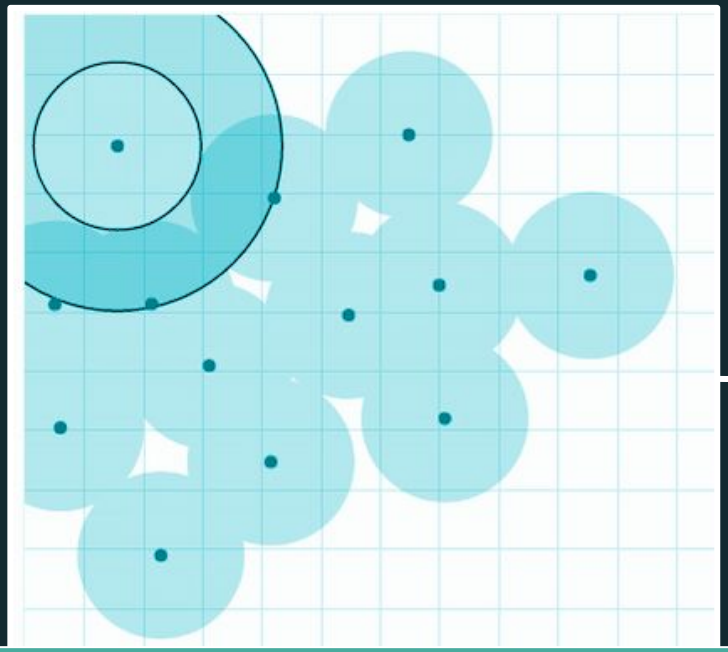
Relative Humidity from Pressure and Temperature

WEATHER UNDERGROUND

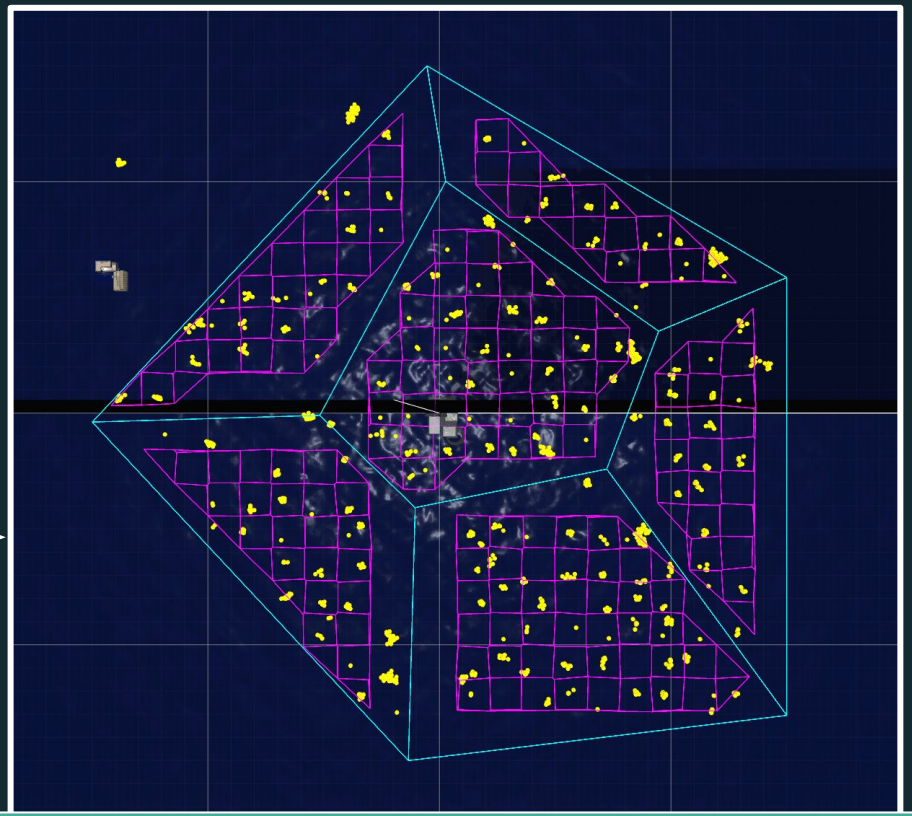
scikits learn machine learning in Python

matplotlib

Sampling point generation

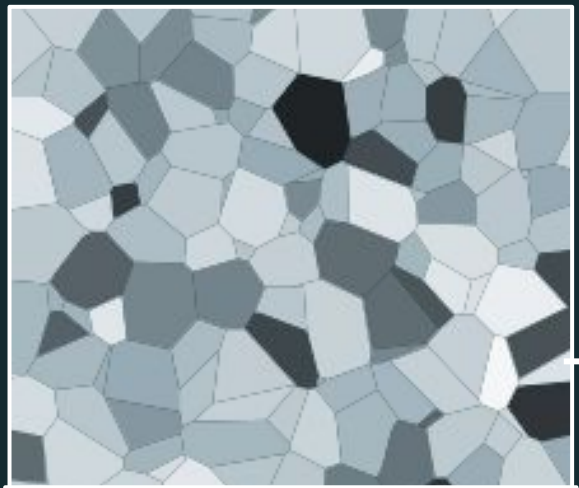


Poisson Sample Point Generation

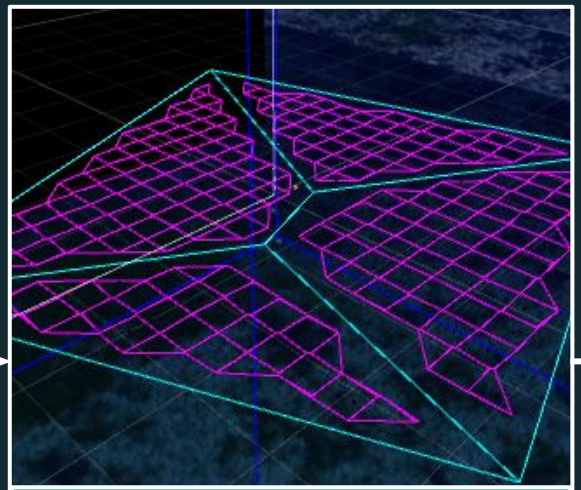


Item Placement

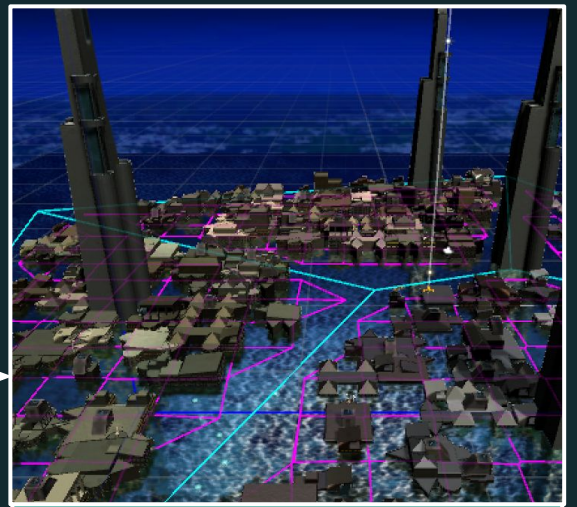
Procedural City Generation



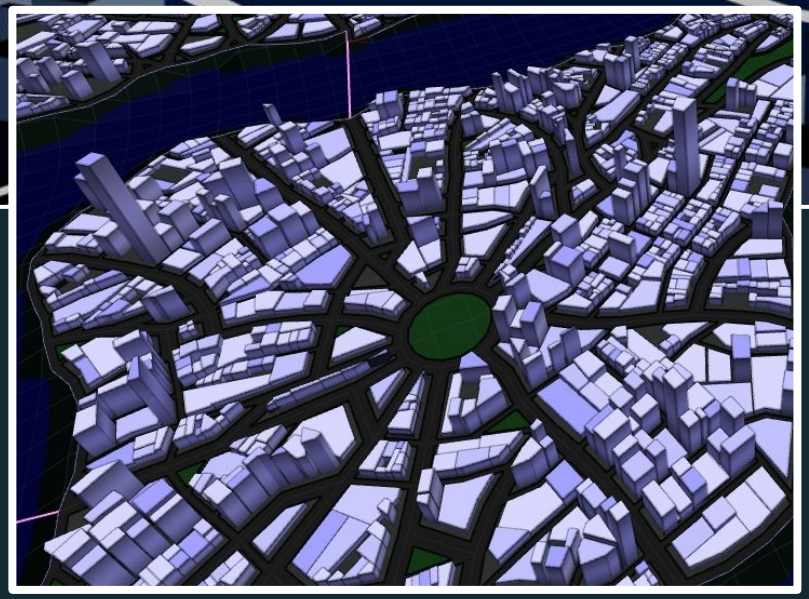
Voronoi Diagram with seed points created with **Poisson Distribution** to generate districts and city edges.



Voronoi Diagram with regularly spaced seed points to create city blocks.



Generate building meshes and pack buildings onto edges of each block



Comparisons

Also block-based city layouts

Buildings are **voxel** which are more easily managed in memory

Subversion

Introversion Software

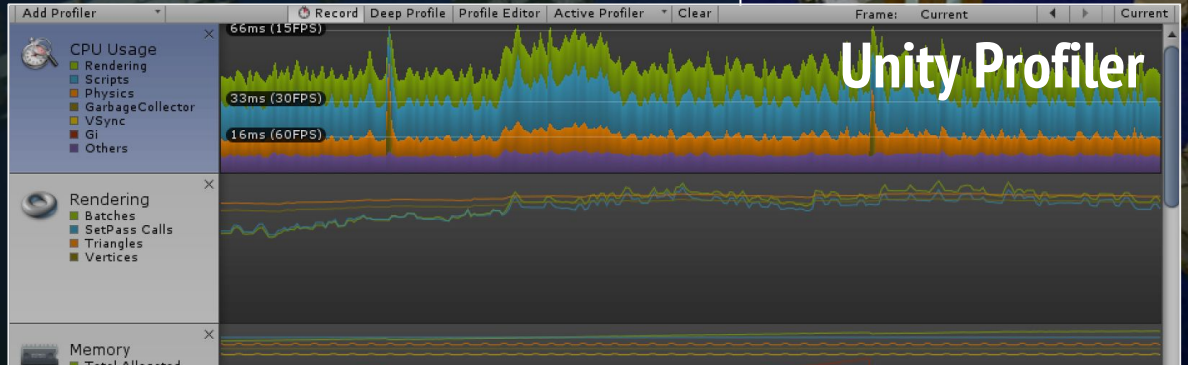
(Unreleased)

City Optimization

Load city buildings in **chunks**

Creature and item **pools**

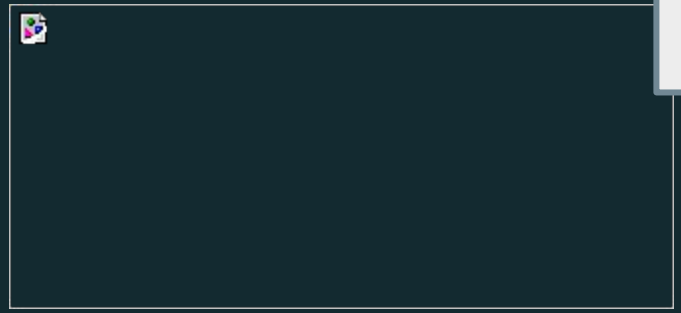
Hide buildings with fog to save on rendering complex meshes in distance



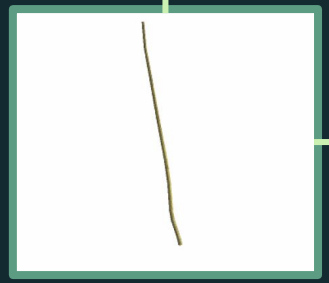
50,000 FT VIEW

Architectural Diagram

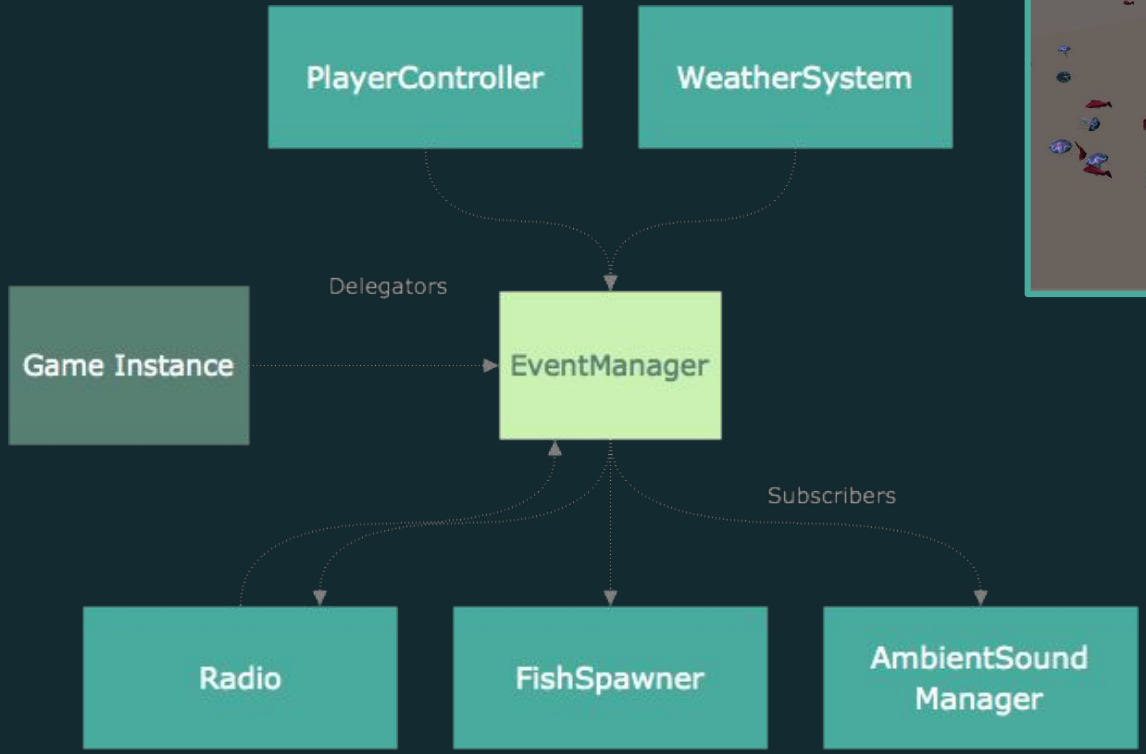
Item Crafting



Config Files

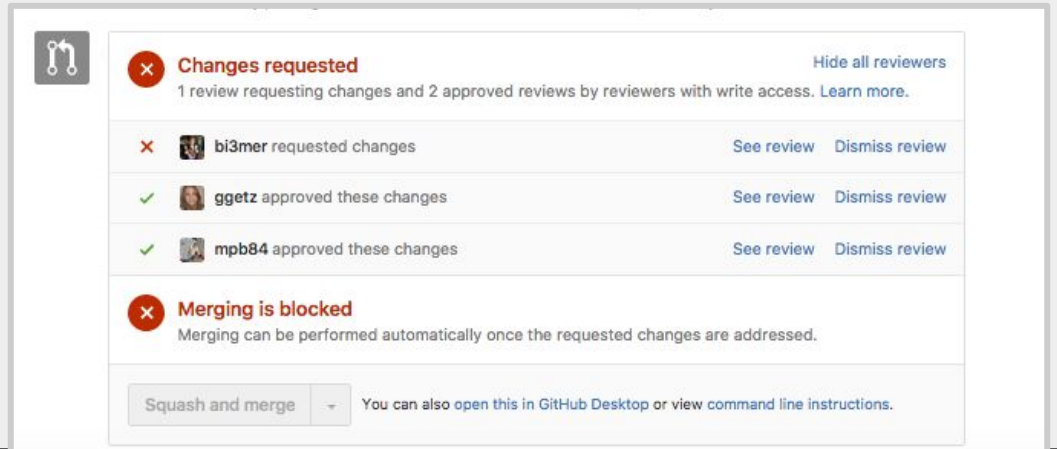


Event Manager



Development Pipeline

user creates pull request | reviewer approves or denies | user merges pull request



Software & Game Development Standards

Game Performance
Run at 60 FPS on machine with recommended requirements

XML Documented
All code required standard C# XML documentation for public methods and members

Coding Style
Followed Microsoft C# Coding Conventions as well as own Style Guide

Unity Best Practices
Proper asset naming, project settings, and directory structure, use of prefabs

Designer Friendly
All values were exposed to be changed by designers through Unity and configuration files

GOF Design Patterns
Including Bridge, Controller, Decorator, Factory, Observers, Singleton, etc.

Playtesting
Qualitative game testing in addition to software tests

Dynamic UI
Scalable, responsive, and user tested

Testing

NUnit & Unity Test Tools

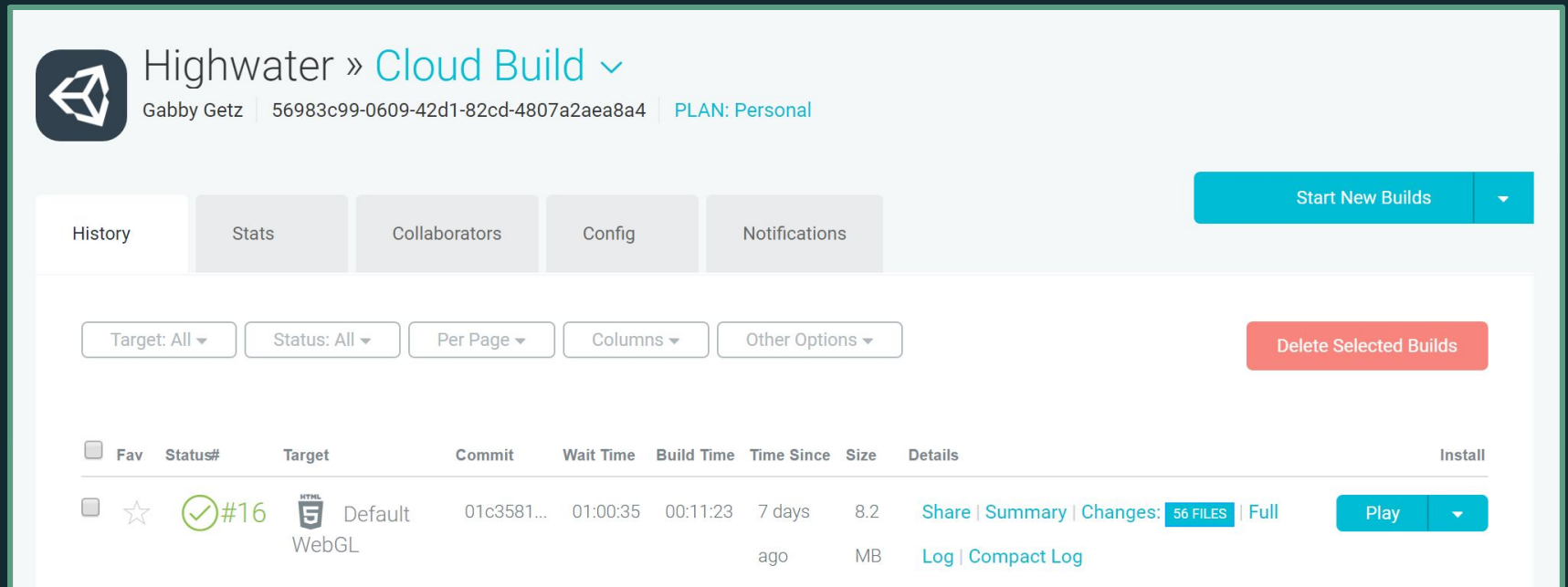
NUnit Framework 2.6 for backend code tests

Qualitative testing done through **playtesting** in tandem with the DIGM Team

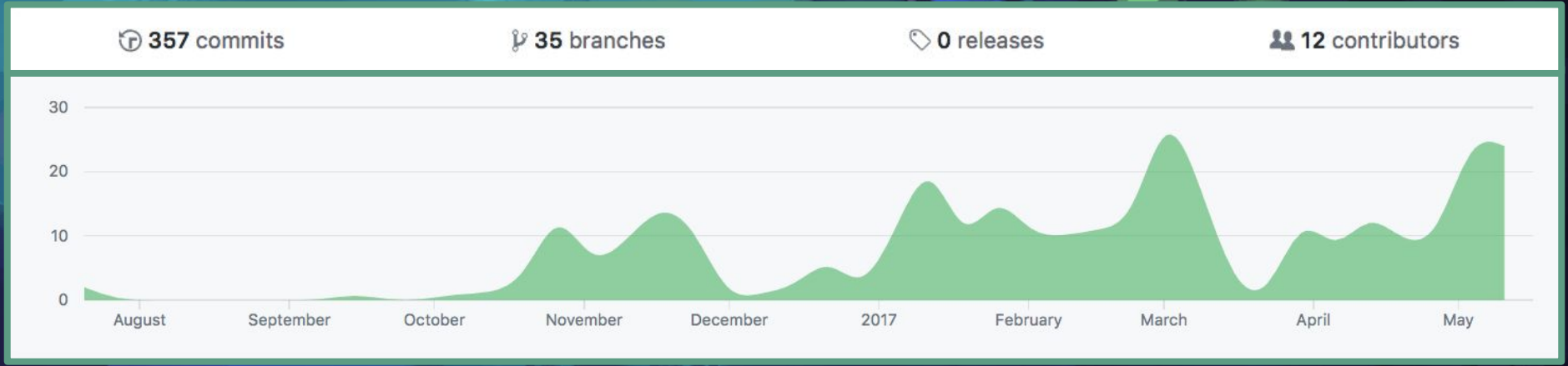
```
1  using UnityEngine;
2  using UnityEditor;
3  using NUnit.Framework;
4
5  [TestFixture]
6  public class BaseItemTests
7  {
8      [Test]
9      public void AddItemCategoryMethodShouldUpdateGetItemCategoryList()
10     {
11         //Arrange
12         BaseItem stick = new BaseItem("Sample Stick");
13         SolidCategory solid = new SolidCategory ();
14         stick.AddItemCategory (solid);
15         Stack stickStack = new Stack (stick, 4, "");
16
17         // Act
18         stickStack.Item.GetItemCategories();
19
20         // Assert
21         Assert.AreEqual (1, stickStack.Item.GetItemCategories().Count);
22     }
23 }
24
```

Continuous Integration

Unit Testing & Unity Cloud Build



Statistics



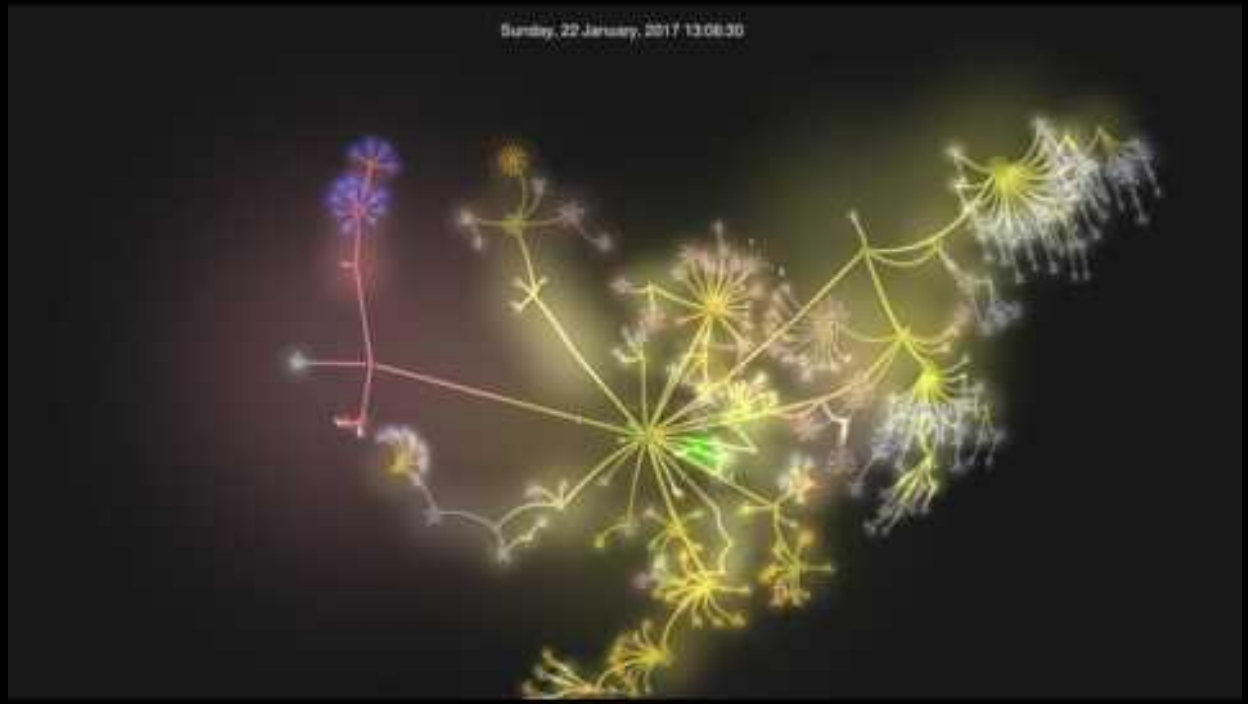
23,219 lines of code

1041 lines of unit tests

368 closed pull requests

422 closed issues

46 play-tests


Sunday, 22 January, 2017 13:08:30


THANK YOU!