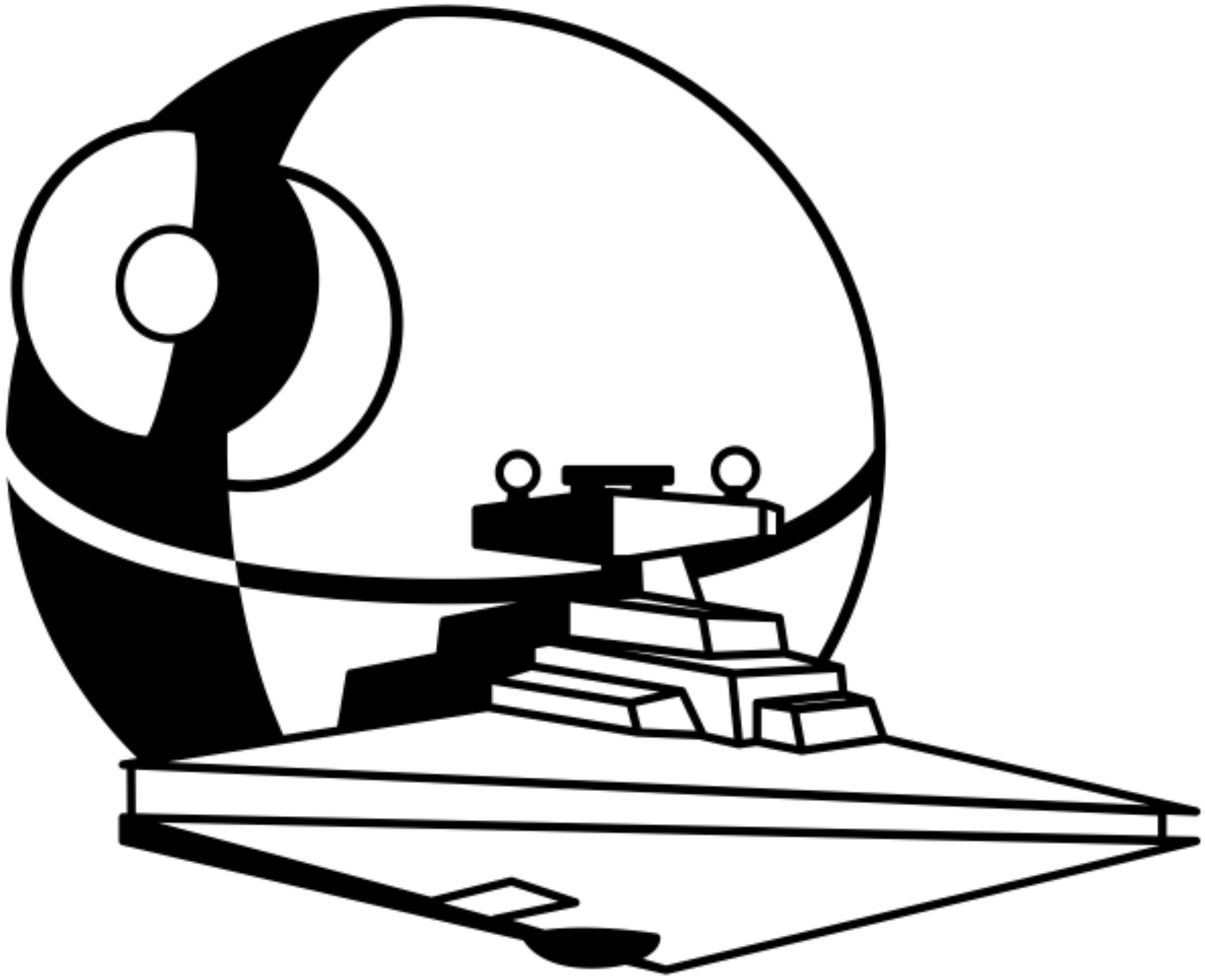
unity

VOID STAR



'Help us Steven Price you're our only hope'

Table of Contents

VoidStar Development Team Members

PRODUCERS
CLAYTON MATULICH
ELIE PUFFELIS
ERNESTO PARRA
YAO WANG

PRODJECT MANAGER
ERNESTO PARRA

PRODUCTION COORDINATOR
ERNSTO PARRA

GAME DESIGNERS
CLAYTON MATULICH
ELIE PUFFELIS
ERNESTO PARRA
YAO WANG

SYSTEMS COORDINATOR
YAO WANG

PROGRAMMERS
CLAYTON MATULICH
ELIE PUFFELIS
YAO WANG

TECHNICAL ARTISTS
ELIE PUFFELIS

AUDIO ENGINEERS
ELIE PUFFELIS

QUALITY ASSURANCE TESTERS
CLAYTON MATULICH
ELIE PUFFELIS
ERNESTO PARRA
YAO WANG

1 Executive Summary

Game Overview

Void Star is a first-person, timed, survival shooter game where the Player is a star pilot (boy or girl) who is on a mission to defend their base from being destroyed. Now the Player must run through their base and get to the loading dock where the star fighters are located, select one of them and take it up to space to defend from the oncoming attack from above. The Player may acquire points from destroying enemy ships, the more that are destroyed in the given time frame the better.

Technical Summary

Void Star will be developed in about 3 weeks by 4 people using the Unity Gaming Engine. For the asset creation Google Poly was used in order to find the desired assets as well as other websites such as the Unity Asset Store. The game was developed as a student project for an Advanced 3D Game Design class and will not be used to accumulate any form of revenue and will not be published to any store for distribution.

The game will initially be deployed on Mac OS

Minimum Mac OS X requirements:
PC – Mac
OS – Mac OS X 10.14.4
Graphics Card – Intel Iris Plus Graphics 640 1536 MB

2 Equipment

Hardware

The four members of this development team used an all Mac setup which including of two 2018 13-inch MacBook Pro's, one 2018 15-inch MacBook Pro, and one 2015 15-inch MacBook Pro. All MacBook's are running the latest version of Mac OS X Mojave 10.14.4. Additional hardware choices included an iPhone XS Max running iOS 12.2 to begin testing deployment for the game in iOS.

Product	**Task**	**Quantity**	**Cost**
MacBook Pro 13'	Documentation Programming Game Creation Game Testing	2	~$3,600
MacBook Pro 15'	Programming Game Creation Game Testing Photoshop	2	~$6,000
			Total: $9,600

Software

The software used in the creation of this game was mainly the Unity Engine implementing the C# language for coding, Microsoft Word to write the documentation, Photoshop for the design of the in-game HUD's, GitKraken and GitHub as version control interfaces, and Google Poly for finding the desired assets for the game. The use of software from each team member depended on their role and responsibility they played in the development of this game. Programmers being the ones who used Unity the most to actually create the game.

Product	Task	Cost	Quantity
Unity 2019	Game Creation and Development	Free	4
Microsoft Word Student Edition	Writing Documentation	Free	1
Photoshop	Designs	$25/year	1
Google Poly	Asset Finding	Free	2
GitKraken	Version Control	Free	3
GitHub	Version Control	Free	1

Total: $25

3 Evaluation

Game Engine

The gaming engine utilized to created Void Star is the Unity Gaming Engine (Personal Edition). With Unity it is very easy to visualize and create a 3D game in this engine and the process of putting the game together and to build the game on several different platforms made the game creation process easier.

Target Platform

Void Star was initially created with the idea to make the game for the Mac OS platform on computer but there is work currently being done to bring this game to the mobile platform, specifically iOS. The Mac platform was easy for us to deploy on since this game was created and tested on said platform, but we understand that this is not the most popular platform. Being able to deploy the game on a mobile platform could bring the game to more people if it were ever released to an audience. Bringing the game to two different markets.

4 Scheduling

Development Plan

Product	**Week 1**	**Week 2**	**Week 3**	**Week 4**
Brainstorm Meetings	Game Concept, Role Assignment, Platform Deployment	Idea for implementing AI and PCG in game	Idea for iOS deployment	
Assets, Art & Design	Style for game selected	Use of old assets	Search for new assets	
Game Layout	Physical drawn out game design	Began the design of levels	Further design of levels	Ideas for layout of iOS layout
Scripting		Character select, character movement, menu scene, first person shooter and views	UI Functionality, game timer, score keeper, improvement of character movement, terrain adjustments	Discussion of Unity Cloud Build for extra credit
Audio & Images	Folders created for audio and image housing	Skyboxes created and audio added		
Documentation		Began rough draft GDD & TDD, readme update	Rough Draft of TDD & GDD finished, Issue 1 listed, wiki page for works cited created, readme pushed	

Deadlines

April 30th – MVP Due with GDD and TDD rough drafts

May 7th or 9th – Final game build and presentation due with final drafts of game design and technical design document

5 Work Environment

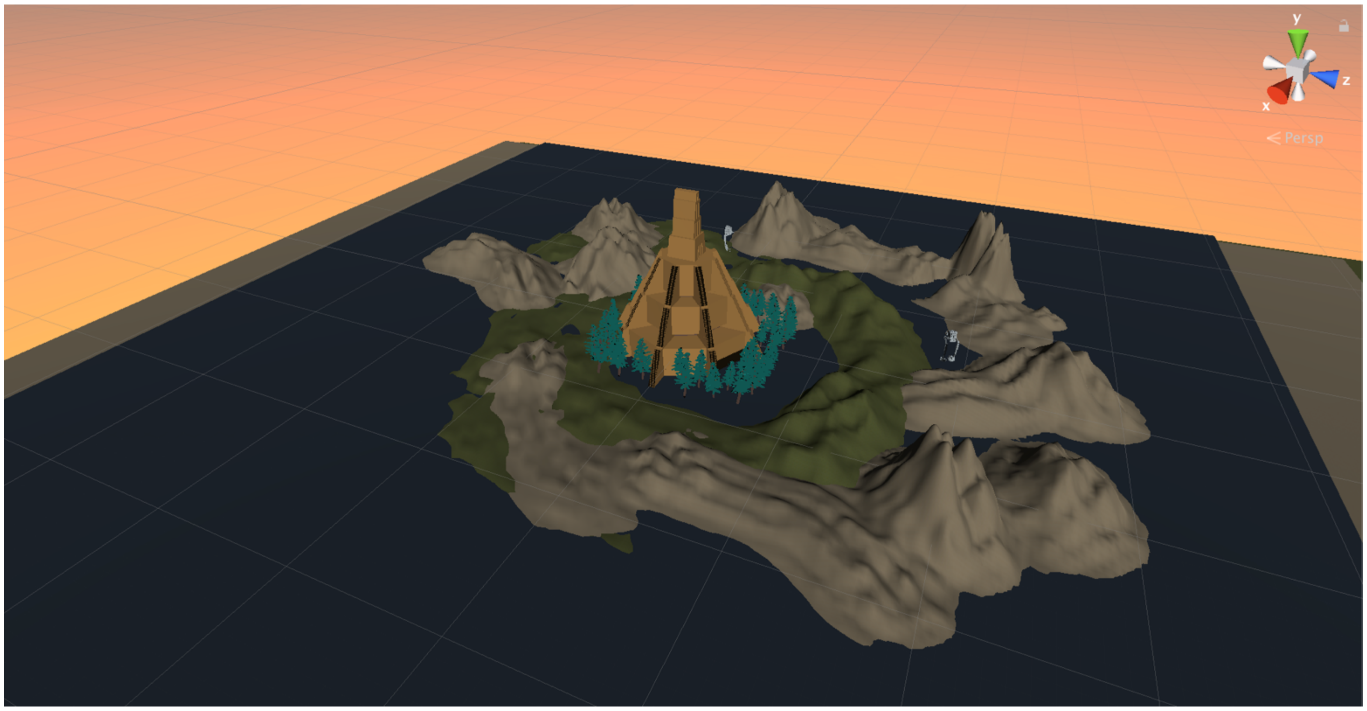
Remote Collaboration

The team is comprised of four members, three undergraduate and one graduate computer science students from San Diego State University. Four students with busy induvial schedules had us consider the best possible option to be able to work together being physically apart so we used version control software such as GitKraken and GitHub in order to push and pull changes to the project and creating branches for members to work on their own work.

6 Levels

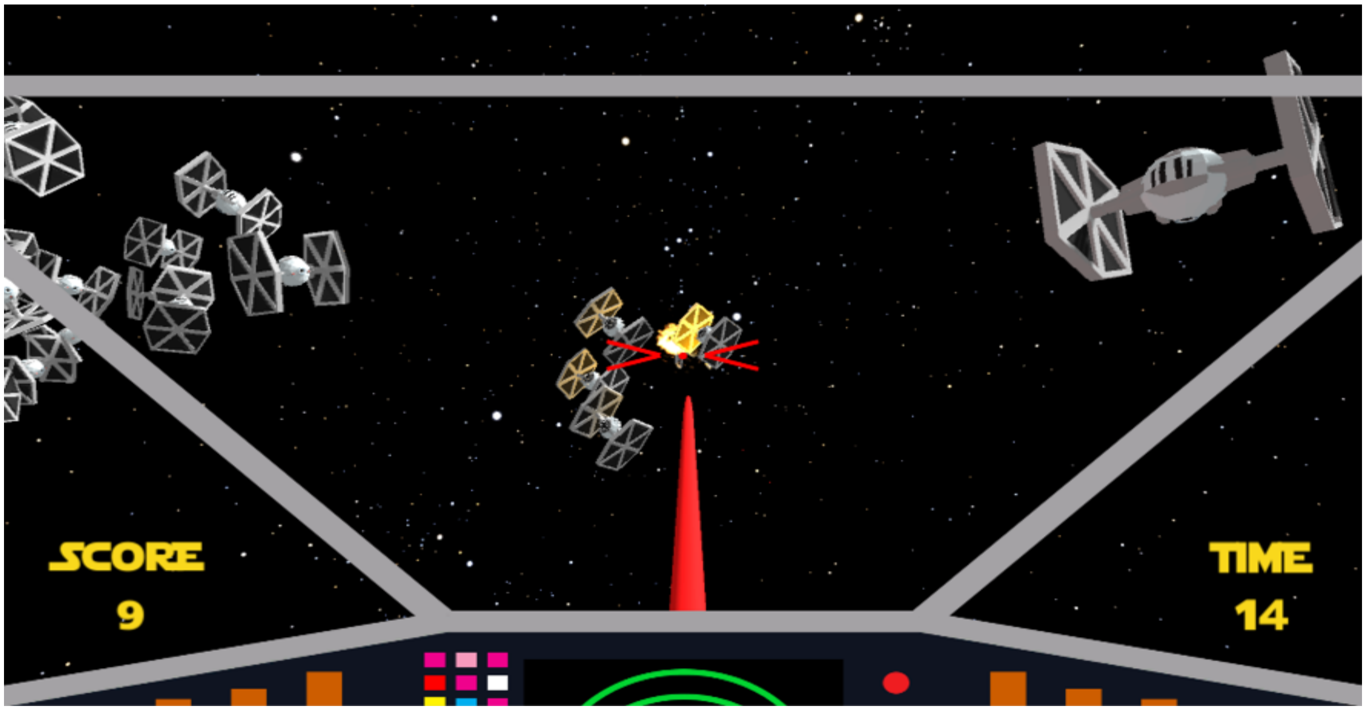
Level 1

Void Star begins with the player waking up after an assault has been launched on their planet of Yavin. The player will spawn at the same point for every iteration. The initial level is a large terrain with a temple in the middle where the players first objective lies. Laser blast rain down from the sky and walkers stand in the players way of making it to their objective.



Level 2

Once the player makes it to their objective they are transported to space where the main purpose of the game is played out. The first-person shooter view will put the player in a turret like position where the player can look around shoot and destroy the enemy boids.



Asset List

Player	Game Camera
Enemies	Tie Fighters
Player Ships	X-Wing (Closed & Opened Wing)
	Tie Fighters
	Tie Silencers
Terrain Walkers	ATST’s
Environment	Yavin Temple
	Terrain created in Unity

7 Complexity Analysis

Procedurally Generated Terrain

Building the Mesh - O(n^2)

To build vertices and set the location of the triangles within the mesh plane, we had to set up a loop to go the distance of the width of the map, and within that loop, create another loop to reach the distance of the height of the map. With these nested for loops we have a complexity of O(n^2)

Building the Color Overlay - O(n^2)

Although we have 3 nested for loops here, the thirst for loop is not dependent on the size of the map like the other 2 nested loops. For the two first loops we have the same as before, iterating through all vertices on the map. The last loop here iterates through all the different types of regions we have in order to add color, but the region amount will be a constant, and not dependent on the size of the map.

Generating Noise - O(n^2)

Again, here we have 3 nested loops and the innermost being a constant based loop. The two outer loops again iterate through the all the vertices in the whole map (width and then height), and for each one, we loop again through the specified constant Octaves value, generate a pseudo random value using the built it Unity Perlin noise function, then assign that value to the vertices' heights. We then normalize the height values using the Inverse Lerp function so that we can have values between 0 and 1.

Artificial Intelligence

Spawning and Array of Boids - O(1) or O(n)

Runtime can determine on specific implementation. In my implementation, in the editor or in the code you can set how many boids to spawn. By default, I spawn 20, so when I loop through every boid in the array to spawn them, it will be a constant time. But in a possible future implementation, I would like the user to be able to select in the UI the number of boids to spawn, therefore making n a non-constant and making the runtime O(n). In this method we also assign every boid a reference to the flock manager/controller so that it will always have point of reference when moving.

Apply Rules - O(1)

When we apply rules to our boids, what this means is that we update the positions, headings, and speeds of our boids in reference to other boids. This information is gathered in calculated in the update method, but we only run it 10 to 20 percent of the time in order to reduce calculation overhead (since we don't need to make THAT many calculations a second).

This is all done in constant time because we know the size of the array of boids and their respective neighborhood stats. Again, although this is done in constant time, calculating new headings and spacing is a lot of overhead, and will increase greatly with the number of boids given.