



SPACE CATS!

CISE4914 – Final Report

Spring 2020

Michael O'Connell

Wilson Gansler

15 April 2020

Abstract

This project emphasized digital gaming, in particular the implementation of a single player 3D action game in which the player controls a futuristic spaceship. This game simulates dogfighting among different factions in a fictional world with a cartoon inspired narrative.

We developed our game with the Unity 3D game engine, which allowed for a 3D aesthetic and allowed for basic collision detection along with the rapid development of custom physics, controls, and other systems such as AI enemies. We were able to develop a complete project with our limited development time and scope and create builds for Windows and MacOS.

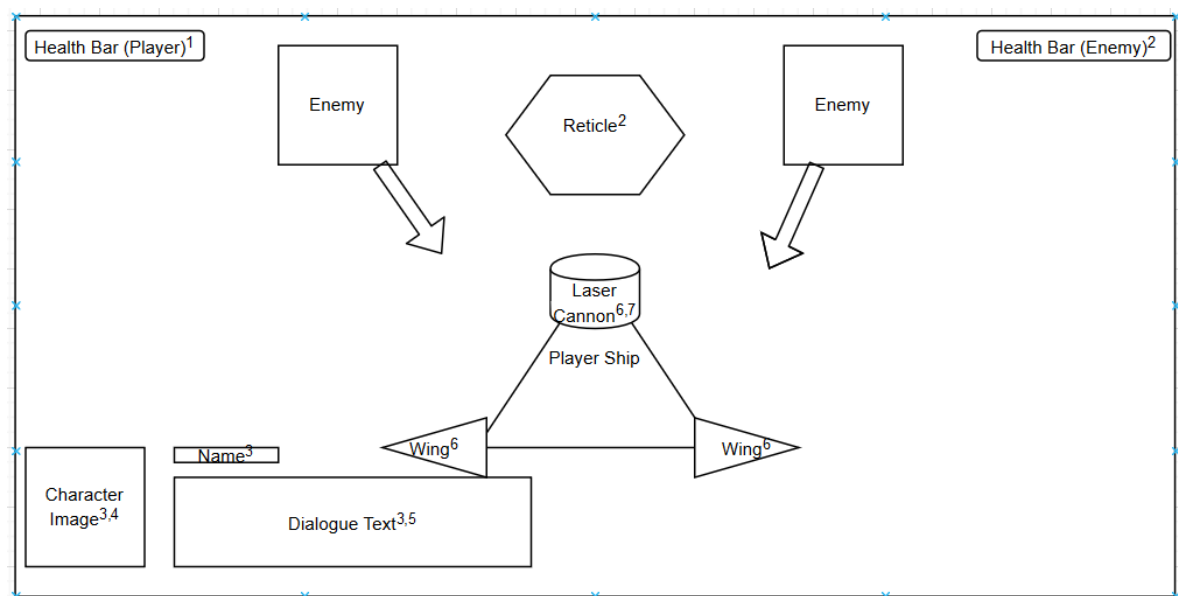
In the future, we hope to iteration upon our final build to create viable commercial product by creating custom artwork, fine-tuning our level design, and expanding our level count.

Introduction

This project emphasized digital gaming, in particular the implementation of a single player 3D action game in which the player controls a futuristic spaceship. The game simulates dogfighting among different factions in a fictional world with a light, cartoon inspired narrative. Our group chose to create a 3D game because we had little experience with 3D and we wanted to grow as developers. We were inspired by the game Star Fox 64 and wanted to create a game that conveyed a similar experience yet also create something new and entertaining.

We researched techniques used in Star Fox 64 such as an on-rails camera system to improve the feel of our game (Cardoso), and created a pseudo markdown language to facilitate our dialogue system for the narrative. While developing these new systems and algorithms, We carefully adhered to Microsoft's C# Documentation (Wagner), and the Unity Engine's documentation (Unity Technologies).

Solution – Technical Approach



1: Player Health only shows if (Health <= 50% || Player Heals || Player Takes Damage || etc.)

2: Enemy Health only shows if ((Reticle's Last Highlight is Alive & Visible) && (Enemy is a boss))

Reticle also changes color when over an enemy

3: These UI Elements only show while interacting/talking with an NPC (Reacts to player actions)

4: Character Image will most likely be stock images of cats or 2D Profile Portraits

5: Dialogue Text automatically advances and changes based on player behavior

6: These objects are destructible, they have hidden health bars but will start to emit smoke when their health is <=50 and increase in particle count as health decreases. When they are destroyed, functionality will change (harder to turn || lose power ups).

7: The Laser Cannon will never be completely destroyed, instead the player will just lose the double laser power up.

Before we started prototyping, we created a mock up diagram for what we wanted our game to look like, from there we started importing test assets and creating our mechanics. To best match the specifications of this design.

Creating Satisfying Controls

To move our spaceship, I needed to get player input as a normalized 2D vector (so that moving diagonally would not be faster than moving horizontally or vertically). I then adjusted for the current orientation of the camera by using vector modification and multiplication (Unity Technologies).

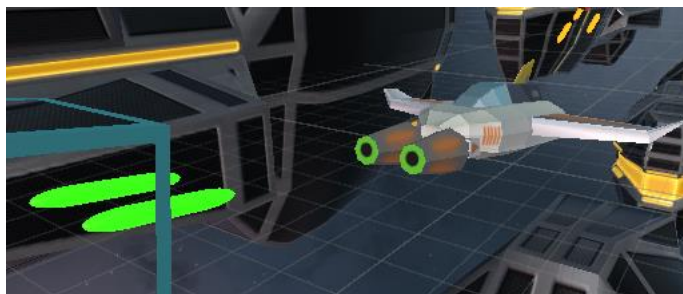
Afterwards I calculated velocity and smoothed it towards the new velocity using linear interpolation with respect to acceleration along with calculating rotations to enhance the feeling of control the player had. Then I could account for our games framerate on multiple systems and perform movement.

I also wanted to reward player's for avoiding obstacles and learning each level so I made the player's wings destructible and reduced the turning angle and speed when they were destroyed. I also added a laser powerup which would double the amount of lasers a player could shoot until the additional laser component was destroyed.

Facilitating Combat

To create a dynamic combat system, I first needed to create a component for health. I made health polymorphic so that it could be used for multiple game objects i.e. the player, enemies, destructible ship parts, destructible set pieces. I also added modifier and response functions to handle taking damage, acquiring health pickups, and death depending on the object.

I started testing my health component by creating a contact damager that would check for collisions and decrement health. Upon completing this I had realized that I could reuse this implementation in my projectile system.



1. Now we have projectiles (Make sure their not instantiated as children)

Projectiles would be game objects with contact damagers that would have a specified lifetime and velocity which would come from projectile emitters and emit a sound effect. I created a generalizable emitter for the player's laser canons and realized that our base combat system was complete.

Navigating a 3D Space

Upon testing my mechanics I had realized that moving in 3D without limitations was confusing for players so I developed my on rails system. At this point I wanted to be able to dynamically create these bounds to simplify level development.



1. On Rails System (Bounded)

I developed a prototype in which the developer could duplicate 3D points and drag them through a 3D space. Each 3D point contained the coordinates to create a rectangle which a developer could specify and each rectangle would be connected to the next in the sequence to create a frustum when read by my on rails boundary system.

This prototype had limitations in creating cinematic events, but was perfect for our tutorial level which introduced our player to the controls. Upon further iteration, Wilson was able to create an on rails system which mimicked that of a 3D roller coaster using Unity's Cinemachine API (Cardoso).



On Rails System (Cinematic) – Testing Level

Managing Dialogues

Dialogues were a challenge to implement. I learned how to implement basic dialogues in Unity from Brackeys and decided to expand upon it to take input from “.txt” files and create my own markdown language (Thirslund). Dialogue strings would be separated by new lines with special markups to specify what to update, e.g. [Name=Sam] would change the displayed name to Sam and [Image=Sam] would change the image displayed to Sam.png.

I also created Dialogue Triggers, which were responsible for triggering the dialogue manager (Thirslund). They were open to any implementation such as when a player takes a certain amount of damage, or when a player clicks on an object, but we ended up using collision based checkpoints for the most part as we wanted to tell a linear story as our player progressed through our level.

The Dialogue Triggers would be received by the Dialogue Manager which would, parse the file’s contents, toggle the dialogue UI, and cycle through text with a machine independent timer.

The Development of Enemies



1. Combining simple AI Components to create emergent behavior.

Enemies started out as simple turrets that had a sight range, cooldown and a firing speed. I added Wilson’s turret prefab to a ship asset and started developing movement AI for the parent

object. At first we tried creating state-driven AI that would chase and corner a careless player from all directions, but those enemies would just frustrate players. We then decided to create reactionary AI instead.

We then used a simple movement script which would respond to when a player had reached a certain point and would travel across a preset path. If you took too long to defeat an enemy, they would fly away and you would miss out on points. I then combined this behavior with Wilson's flocking and swarming behavior for more difficult special groups of enemies to add more variety and life-like behavior to our game.

From the success of our enemy AI, we created a reactionary boss enemy which combined our other mechanics such as destructible parts. As more parts were destroyed, the boss would become more aggressive and when low on health would launch a powerful desperation attack.

Solution – Technical Results

We were able to connect all of our systems across our development milestones into game builds and have carried that momentum into our final build which runs with no errors.



1. UI Enabled, navigating a combat sequence while receiving a narrative briefing. (Player has lost their left wing)

A player will see buttons on the menu to learn about and start the game and then enter the game world and see ships and lasers flying along with the backdrop of a pink planet surrounded by warships and shrimp.



2. UI Disabled - Dodging enemy fire

The player will then learn how to move the character around, how to avoid taking damage, and how to identify and shoot enemies. They'll learn how to angle their ship to avoid attacks and shoot simultaneously and receive narrative briefings.



3. UI Disabled - A skilled player aiming for a risky powerup

The player will have then figured out the rules of the game e.g. how much the ship moves when a button is pressed, how to score points, and how to progress.

Conclusions

We created a 3D Space Shooter Action Game in Unity which simulates dogfighting and contains a lighthearted narrative about cat pilots. I developed our games core controls and

physics, a dynamic combat system, our initial on rails system with editor tools, along with creating systems for UI and dialogues.

I learned how to create responsive controls and satisfying physics, efficient polymorphic game components, develop level design tools, develop my own dialogue system with a custom markdowns, and create responsive AI from simple behaviors.

Since this project focused on mechanics, we did not have enough time to develop the artistic side of our game. We also had to make the decision to allow our player to phase through objects when taking damage in our cinematic level, this was because colliding with static objects would break our camera system and disorient the player which we didn't notice until late in development.

We believe that upon further iteration, we could keep improving our game's mechanics and aesthetics and eventually create a commercially viable product. We would also have to develop more levels, expand gameplay possibilities, and create custom art assets.

Standards

Space Cats! Was developed using Unity 2019.2.9.f1 which uses .NET 4.0.30319.42000 and defaults to ASCII encoding.

Constraints

All software was developed to run in real time in a 16:9 Aspect Ratio for 64-Bit Windows 10 Machines, our MacOS build is a secondary build that may not reflect the intended experience.

Acknowledgements

The author thanks Nick Heitzman for advising the project and for assistance with testing development builds, sending detailed feedback, reviewing our design ideas, and helping us set up our milestone schedule.

Works Cited

Cardoso, André, director. *Star Fox's Rail Movement / Mix and Jam*. *Youtube*, 7 July 2019, www.youtube.com/watch?v=JVbr7osMYTo.

Thirslund, Asbjørn. "Brackeys." *YouTUBE*, YouTube, 2015, www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA.

Unity Technologies. "Unity User Manual (2019.3)." Unity, Mar. 2019, docs.unity3d.com/Manual/

Wagner, Bill. "C# Docs." C# Docs | Microsoft Docs, 2019, docs.microsoft.com/en-us/dotnet/csharp/.

Zucconi, Alan. "Tutorials." Alan Zucconi, www.alanzucconi.com/tutorials/.

Biography

Michael J. O'Connell was born in Jacksonville, Florida on May 16, 1998. He completed his secondary education at Bishop Kenny High School, and is completing his baccalaureate degree in Computer and Information Science at University of Florida (Gainesville, FL), where he expects to graduate on May 3, 2020. Michael is an avid computer programmer, with professional experience in Game Development, is proficient in C++, C#, and Python, and always ready to help. He recently completed an internship with LUMA at Digital Worlds Institute, a student developed, professionally produced multimedia studio where he worked on City Builder Interactive for the City of Gainesville. He's currently learning the Unreal Engine and is finalizing his professional portfolio before applying to various game companies. Michael enjoys biking, hiking, spending time with friends, and (of course) video games. He also hopes to advance his skills in hopes of publishing his own game while meeting new friends along the way.