CSCI 370 Final Report

The Plain Team

Ben Burghardt
Clark T. Howard
Mohak Dahal
Roberto Garcia

Revised June 14, 2024

CSCI 370 Summer 2024

Dr. Zibo Wang

Table 1: Revision history

Revision	Date	Comments
New	5/16/2024	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done XIII. Team Profile
Rev – 2	5/23/2024	VI. Design
Rev – 3	5/20/204	VII. Software Tests and Quality VIII. Ethics Principles
Rev-4	6/7/2024	IX. Project Completion Status X. Future Work XI. Lessons Learned XII. Acknowledgements
Rev-5	6/12/2024	References Appendix A – Key Terms
Rev-6	6/	VI. Ethics Principles VII. Software Tests and Quality

Table of Contents

I. Introduction

A. High level scope/description of the project

The project requires the team to develop a Unity program that randomly generates a mass amount of realistic images of aircraft on airfields and runways. The program must allow for additional 3D models of aircraft to be imported so that images are generated with them. The generated images will then be used for training a machine learning (ML) model that is capable of correctly identifying and classifying various types of aircraft in the generated images. The ML model will produce annotated images overlaying the original image showing where the identified aircraft is within the image. The algorithm will be able to identify many various classes of aircraft from a variety of angles, light levels, obstructed views, and groupings of aircraft. The project is exploratory and meant to determine if the client will go on to invest time in expanding upon it themselves or with a future field session.

B. Client

Stratom was founded in 2001 as a one-man consulting and engineering services shop with a laser focus on cutting-edge engineering solutions, Stratom today leverages decades of expertise to excel at practical problem-solving aimed at reducing the risk associated with autonomous systems development.

Inspired by EOD robots, IED defeat tools and other advances in the defense sector, Stratom has evolved its areas of expertise, expanding its IP portfolio to bring unparalleled core technical expertise to the defense industry and beyond to truly make a difference in the world. Through a deep focus on the advanced technology horizons becoming the cutting-edge products of tomorrow and the development of an intimate understanding of client pain points, needs and niche fit, Stratom has earned a reputation as a trusted partner in delivering holistic automated logistics solutions for both defense and commercial clients.

Specializing in robotics, unmanned vehicles, automation, system engineering/integration and payload development, Stratom provides innovative, creative and mission-focused engineering, R&D and logistics/supply chain solutions that are not only tailored to meet our customers' unique needs but also address critical real-world problems.

C. Source of Data

Models were obtained from various free online resources including GrabCad, Unity Assets Store, and CGTrader. Information on Unity programming was largely derived from the Unity manual and open source forums for computer programmers to congregate and share knowledge.

A majority of data used in the training of a machine learning algorithm is artificially generated by the Unity program.

D. Stakeholder

Stratom is the sole direct stakeholder, in that they will be the only ones utilizing the developed software. The project is not meant for commercial nor distributive use. However, there is a high probability that this project will affect US military airfields and some commercial entities like public airports.

II. Functional Requirements

Functional Requirements

1. Working Unity program that performs the following:
    a. Turn models into prefabs
        i. Prefabs are, effectively, perfect copies of models that should be used instead of the original model when planning to use the model multiple times or modify the model in any way
    b. Instantiate prefabs with correct scale, positioning, and rotation
    c. Use collisions to randomly place objects on an airfield while maintaining a realistic scene
    d. Randomly move camera to capture the scene at various angles
    e. Correct placement of background image
    f. Create and store screenshots of our scene in a uniform location
    g. Creates annotations from screenshots to be used in the training of the ML model
2. A machine learning algorithm that
    a. Is trained on 1,000 Unity screenshots and annotations
    b. Can analyze 1,000 images in under an hour and accurately determine if there is a plane present
    c. Is capable of using instance segmentation to create a mask for each plane present in the image

III. Non-Functional Requirements

Non-Functional Requirements

1. Unity
    a. Capable of generating at least 1,000 scenes under one hour
    b. Captured images have planes that take up a non-negligible amount of space in the image
    c. Background image appears semi-realistic, meaning image quality is not extremely poor or stretched out
2. Machine learning algorithm
    a. Has a non-zero confidence rating

IV. Risks

1. No person on the project has worked with Unity or has experience with working with machine learning algorithms and having to find our own model which cause the following risks:
    a. Delays in development
        i. Mitigated by having constant progress reports with client
    b. Incoherent/unfunctional code
        i. Mitigated by doing constant code reviews and analysis of our code to ensure the code integration is seamless
    c. Being unable to run code on certain models and have it behave as expected
        i. Mitigated by implementing additional code that verifies model is usable or tries to make the model conform to our requirements for it

V. Definition of Done

Both the Unity software and ML model must satisfy all the aforementioned functional and non-functional requirements. The developed software must be tested with private 3D models owned by the client and used to classify and analyze private images of the aircraft. The product has been delivered as executables with the raw code attached so that the client may modify it for their purposes.

For our verification of meeting this definition:

1. Create simple tests that ensure compilation and software quality
2. Manual verification of a sample of screenshots and scenes to ensure Unity work meets defined requirements and to ensure there is some degree of realism in the screenshots
3. Verifying confidence rating of ML model is non-zero
    a. If ML model is not finished Roboflow will be used with manual annotations to verify the scene realism

VI. System Architecture

List of Technical Design Issues:

1. Collision detection between aircrafts
2. Smart random plane placement
3. Rendering 3D models as prefabricated objects
4. Importing environment for ML
5. Applying lighting and textures to specific airports

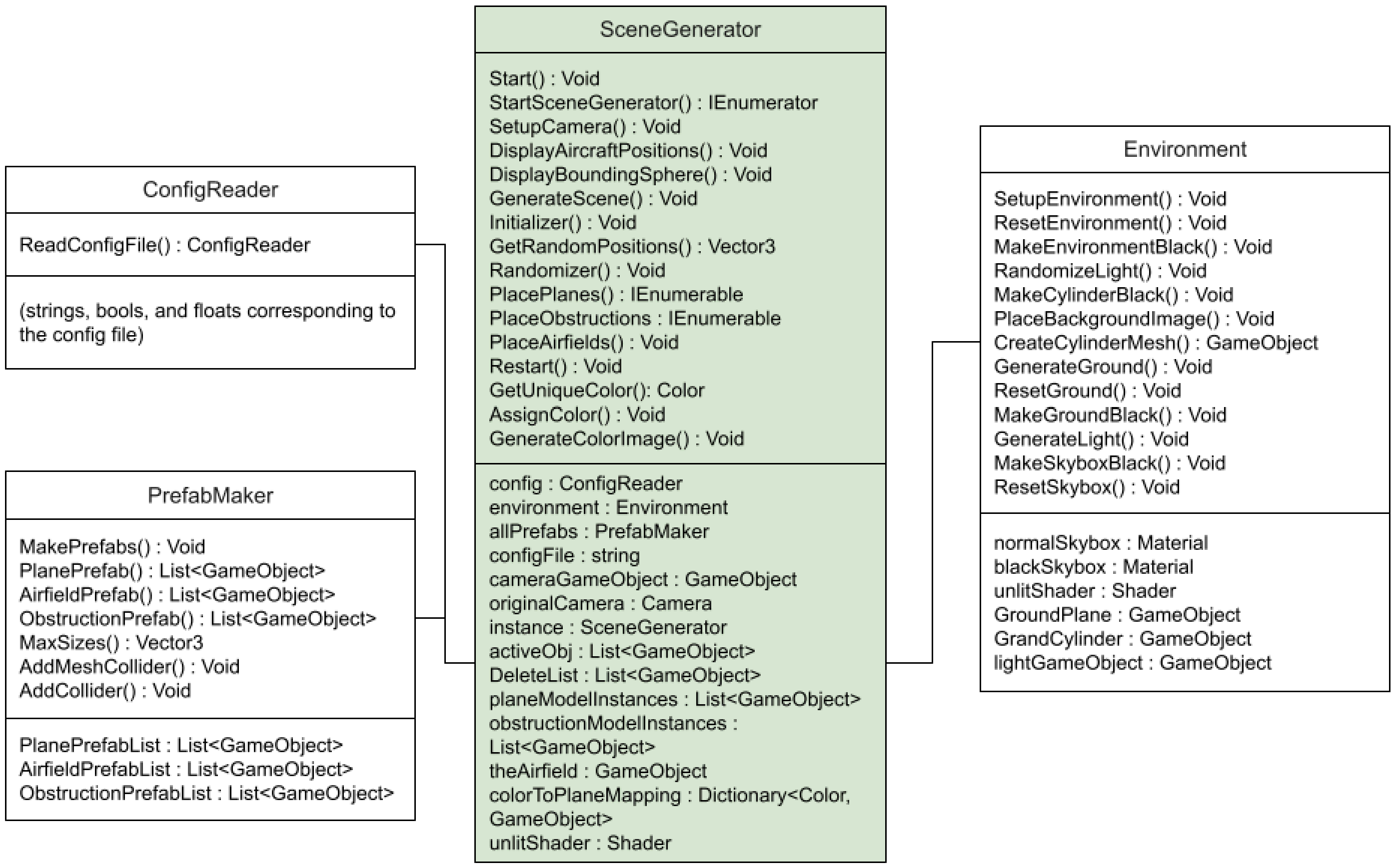

SceneGenerator
Start() : Void
StartSceneGenerator() : IEnumerator
SetupCamera() : Void
DisplayAircraftPositions() : Void
DisplayBoundingSphere() : Void
GenerateScene() : Void
Initializer() : Void
GetRandomPositions() : Vector3
Randomizer() : Void
PlacePlanes() : IEnumerable
PlaceObstructions : IEnumerable
PlaceAirfields() : Void
Restart() : Void
GetUniqueColor(): Color
AssignColor() : Void
GenerateColorImage() : Void
config : ConfigReader
environment : Environment
allPrefabs : PrefabMaker
configFile : string
cameraGameObject : GameObject
originalCamera : Camera
instance : SceneGenerator
activeObj : List<GameObject>
DeleteList : List<GameObject>
planeModelInstances : List<GameObject>
obstructionModelInstances : List<GameObject>
theAirfield : GameObject
colorToPlaneMapping : Dictionary<Color, GameObject>
unlitShader : Shader
ConfigReader
ReadConfigFile() : ConfigReader
(strings, bools, and floats corresponding to the config file)
PrefabMaker
MakePrefabs() : Void
PlanePrefab() : List<GameObject>
AirfieldPrefab() : List<GameObject>
ObstructionPrefab() : List<GameObject>
MaxSizes() : Vector3
AddMeshCollider() : Void
AddCollider() : Void
PlanePrefabList : List<GameObject>
AirfieldPrefabList : List<GameObject>
ObstructionPrefabList : List<GameObject>
Environment
SetupEnvironment() : Void
ResetEnvironment() : Void
MakeEnvironmentBlack() : Void
RandomizeLight() : Void
MakeCylinderBlack() : Void
PlaceBackgroundImage() : Void
CreateCylinderMesh() : GameObject
GenerateGround() : Void
ResetGround() : Void
MakeGroundBlack() : Void
GenerateLight() : Void
MakeSkyboxBlack() : Void
ResetSkybox() : Void
normalSkybox : Material
blackSkybox : Material
unlitShader : Shader
GroundPlane : GameObject
GrandCylinder : GameObject
lightGameObject : GameObject

Figure 1- Unity Function UML Describing Internal Interactions

The Unity program has all functionality starting at the center of the UML tree with the start function in SceneGenerator, shown in figure 1. The start function is called on execution of the program and is responsible for initialization of environment information. The start function begins by reading in the config file, with the ConfigReader class, which stores a variety of runtime information like number of scenes to generate, actual size of each 3D model, etc. The start function will then call upon PrefabMaker to make the airfield prefabs, plane prefabs, and obstruction prefabs to read in all supplied 3D models, normalizing them based on their max sizes so that they can be reused. Finally the start function will call the Environment class to generate static environment assets, like the ground with GenerateGround, that will not be reinstated throughout the program. After initialization is finished, StartSceneGenerator is called and will run for the life of the program.

StartSceneGenerator is responsible for generating every scene, capturing every screenshot, and parsing out all of the information in each screenshot. StartSceneGenerator starts by calling GenerateScene which will initialize the environment, place the background image on a cylinder enclosing the environment, and randomize the placement of the planes, airfields, and obstructions. StartSceneGenerator will then take a screenshot of the scene and generate a colored image of the scene that will be used for data generation. PixelMagic will be called to analyze the colored image to find the polygons and bounding boxes of each aircraft in the image and call GenerateAnnotations. GenerateAnnotations will make a JSON file in COCO dataset format that contains all the necessary information in the picture for the machine learning algorithm to learn. Once all the information is made, the scene will be reset with the Restart function and StartSceneGenerator will loop until all scenes and screenshots are made.

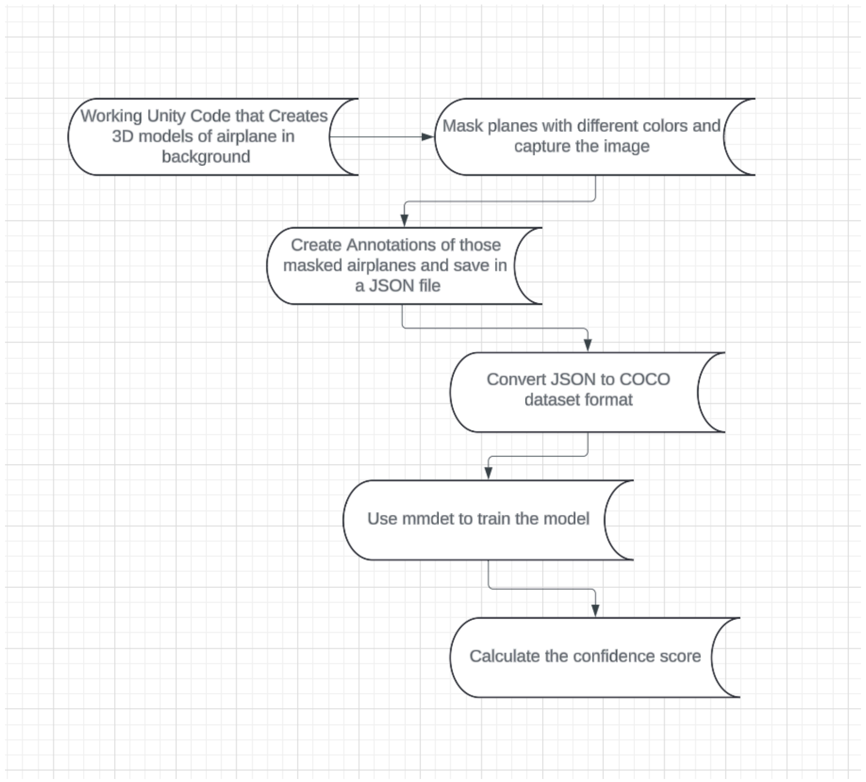

Working Unity Code that Creates 3D models of airplane in background
Mask planes with different colors and capture the image
Create Annotations of those masked airplanes and save in a JSON file
Convert JSON to COCO dataset format
Use mmdet to train the model
Calculate the confidence score

Figure 2- Machine Learning Model Internal Process

After the Unity program runs it will generate several folders containing all of the screenshots and all of the JSON annotations. These files will then be used to train the machine learning algorithm with the MMDetection model. After training the algorithm, it will evaluate its confidence rating and be tested against actual images of the aircraft to evaluate accuracy of the algorithm and the efficiency of the generated training materials.

VII. Software Test and Quality

Software testing and quality assurance was ensured via the usage of editor tests and play state test which will test code functionality while Unity is in the “Editor” state and the “Play” state respectively. Additionally, we verified we were generating quality images to be used for training of the ML model by ensuring that planes took up a non-insignificant percent of the screenshot taken.

This was done as follows:

1. Iterate through the pixels of the screenshot
2. Whenever we encounter an RGB value we have assigned to a plane, we increment a variable that tells us how many pixels that color takes up

3. Once we've finished analyzing the entire screenshot we will calculate the percentage of pixels that each color takes up based on the total amount of pixels, if it's below .1% we will disable the plane that was assigned that specific color and proceed to annotating the screenshot

List of N-Unit Tests that help maintain "Editor" & "Play" mode Quality Assurance:

1. Asset Loading
    a. Folder and 3D files exist
    b. Read in config file
    c. Models are normalized and contain appropriate components
2. Asset Placement
    a. Prefabs exist and accessible
    b. Planes, airfields, and obstructions given a location and rotation
3. Scene Randomization
    a. Selected prefabs exist in scene and non selected prefabs do not exist in scene
    b. Prefab placement collisions
    c. Generates specified number of scenes
4. Screenshot and Annotation Generation
    a. Screenshot taken for each scene
    b. Generate multiple annotation for same screenshot
    c. Empty scene annotation
5. Image Reading
    a. Images exist
    b. Config file exists and specifies valid information

ML model testing and quality assurance was done by using the previously mentioned pixel verification and expanding upon it to create a unique polygon for each plane in the screenshot that was kept and creating an annotation for it that lists all coordinates of where that plane should be in the screenshot. This is done to ensure that the ML model is not trained on bad data such as having a screenshot with two planes clearly visible and a third only taking up a tiny portion of the screenshot which may cause the ML model to misidentify small objects.

Program Analysis

1. Static: testing of Unity functionality while it is in the "Editor" state
    a. N-Unit Testing
        i. Unity editor tests
        ii. Unity play tests
    b. Nullity verification
        i. Ensures correct start up and instantiation
    c. Code Review/Restructuring
        i. Increasing abstraction of the main class
2. Dynamic
    a. Manual verification of Unity scenes, screenshots, and annotations
        i. Annotations are verified through a function that looks at the pixels of an image to give precise data on where planes are for the ML model training

Test Results as of 6/12/2024

1. Unity Tests:
    a. Unity tests return expected results and have verified that the project code is correctly integrated.
    b. Images created with Unity are in line with what was expected to be generated for an exploratory project, however the pixel verification has created doubt within the team about its possible scalability.
2. Machine Learning Tests:
    a. Testing for the ML model has yet to be completed and plans have been put in place to expedite the Roboflow manual verification process

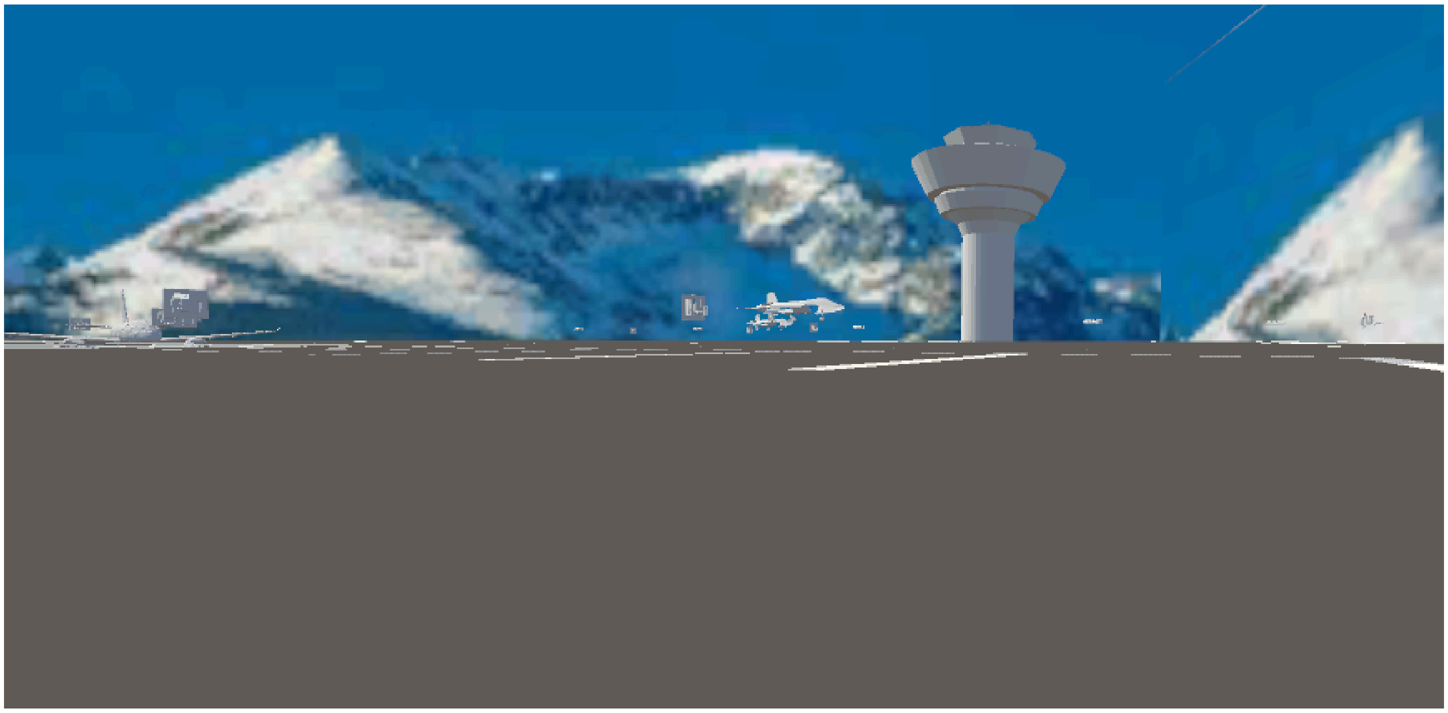
VIII. Project Ethical Considerations

1. Licensing concerns
    a. ACM/IEEE Principle 1.5 – Respect foreign works
        i. It must be verified that all models used in Unity fall under the public domain or are allowed to be used for non ML educational purposes.
2. Ethically ambiguous
    a. ACM/IEEE Principles 1.2 & 3.1 – Avoid harm & ensure the public good
        i. We cannot discount that the ability for an algorithm to recognize planes may be used for purposes other than its planned purpose of being used for automatic refueling of planes and instead be used as a means of identifying opposing military vehicles.

IX. Project Completion Status

As of 6/7/2024:

The Unity aspects of the project are rapidly approaching completion, with testing being the last major component that requires work. As of now we can produce images of random instances, however the problem arises when looking at how realistic our images are. The goal was to create an image of pseudo realistic unique images, the part that the Unity project is struggling with is achieving realisticness. The background is not realistic by any measurement and due to the poor quality of models and lack of built in textures in Unity we are unable to realistically instantiate our models using logic based on naive placement or reliant on poor meshes.



The above image shows the latest screenshot of our scenes, here we can see that there is an airfield with multiple planes in the scene, with obstructions also present and the background not being stretched to an unrecognizable image or too pixelated.

The machine learning aspect of the project is not completed. While the configuration file is working there have been constant setbacks after setbacks that have made it impossible to get a confidence rating on the quality of the Unity images. Roboflow work has begun and will be used as a part of a larger patchwork solution/workaround.

X. Future Work

Future work should be aimed at making the Unity code make more realistic images, this can be done in the following ways:

1. Change the quality of the background images, this can be done in the following ways:
    a. Use images that are only present in the top half of a 1920x1080 image so that the bottom half is not cut off when being seeing by the Unity camera
    b. Change background images from being a Quad GameObject in Unity to a Canvas attached to the camera, this would greatly increase the background image quality but due to time constraints and the difficulty of trying to project a Canvas away from the camera it was scrapped
2. Improving the quality of models used in creation of images, the current working models required large amounts of manual changes to make sure they represented how planes would appear in a realistic environment
3. Camera placement: as it stands the camera aims to have all planes in the image, this may lead to unrealistic images and may cause the algorithm to be trained with faulty data
4. Annotation creation: currently annotations for images are done by analyzing an annotated image checking for pixels and determining polygons, this process is time consuming but will give us extremely accurate coordinates for where planes are made. Based on the client meetings it may be better to instead

use raycasting instead, which will result in worse coordinates for where planes are but lower overhead time and allow for scalability

XI. Lessons Learned

1. Working as a group allowed us to understand each others code worked and made code integration easier
2. Playtesting in Unity comes with a large amount of caveats and should be done incrementally so as not to have the play tests eventually take large amounts of time to run
3. Unity editor mode vs. play mode require different approaches and trying to refactor code from running on editor mode to running on play mode will take large amounts of time. Therefore it is best practice to code in Unity as though you will always be in play mode
4. The configuration file should be prioritized early on for training a ML algorithm
5. Allocating enough time for learning for research, learning, and experimenting with new tools and frameworks is critical, especially when no one on the team has prior experience with them. Learning takes time
6. Testing should be done as the project progresses and not leave them at the last minute since there could be more complications happening towards the end
7. After each major restructuring there should be a dedicated set of time to ensure all branches correctly working with the new structure

XII. Acknowledgments

Special thanks to Annie Didier and Elizabeth Gilmour from Stratom for their help with trying to get the ML model working. We also appreciate the rest of the kind people at Stratom for agreeing to meet with us twice a week and providing us with constructive feedback and suggestions.

Also, thanks to Dr. Zibo Wang for giving us great feedback on our report and presentations.

XIII. Team Profile

Clark T Howard

Senior at Colorado School of Mines

Computer Science Major – Computer Engineering Pathway

Hometown: Edmond, Oklahoma

Mohak Dahal

Senior at Colorado School of Mines

Computer Science Major

Proud US Soldier

Born and raised in Nepal, living currently in Dener, CO.

Experience: Java, Python, SQL, C++, C#

Interests: Video Games, Sports

Ben Burghardt

Senior at Colorado School of Mines

Computer Science

Hometown: Denver, CO

Experience: Computer Science Teaching Assistant, Amazon, Personal Java and C++ Projects

Interests: Movies, Games, Baking, Foreign Language

I am very excited to learn these new tools and new applications of existing technologies to advance growing technologies.

Roberto Garcia

Senior at Colorado School of Mines

Computer Science – Intelligent System Pathway

Hometown: Ft Morgan, CO

Experience: Java, C++, Ruby, Python, C, C#

Interest: League of Legends, Cooking, Food trucks

References

Below are a list of websites that we used to begin our research and frequently referenced while working on the project

Learning how to work with ML models

"Transfer Learning for Computer Vision Tutorial — PyTorch Tutorials 1.7.0 documentation," *pytorch.org*. https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
"Train with customized models and standard datasets — MMDetection 3.3.0 documentation," *mmdetection.readthedocs.io*. https://mmdetection.readthedocs.io/en/latest/user_guides/new_model.html
GeeksforGeeks, "Introduction to Convolution Neural Network," *GeeksforGeeks*, Aug. 21, 2017. https://www.geeksforgeeks.org/introduction-convolution-neural-network/
"COCO - Common Objects in Context," *cocodataset.org*. https://cocodataset.org/#home

Collisions

U. Technologies, "Unity - Scripting API: Collision," *docs.unity3d.com*. https://docs.unity3d.com/ScriptReference/Collision.html

Meshes and Meshcolliders

U. Technologies, “Unity - Scripting API: Mesh,” *docs.unity3d.com*.
https://docs.unity3d.com/ScriptReference/Mesh.html

Quads

U. Technologies, “Unity - Manual: Example: creating a quad,” *docs.unity3d.com*.
https://docs.unity3d.com/Manual/Example-CreatingaBillboardPlane.html

Canvas

U. Technologies, “Unity - Scripting API: Canvas,” *docs.unity3d.com*.
https://docs.unity3d.com/ScriptReference/Canvas.html
Testing

“About Unity Test Framework | Test Framework | 1.4.2,” *docs.unity3d.com*.
https://docs.unity3d.com/Packages/com.unity.test-framework@1.4/manual/index.html
“NUnit.org,” *nunit.org*. https://nunit.org/

ML model implementation debugging

“MaskRCNN not in Model Registry · Issue #10052 · open-mmlab/mmdetection,” *GitHub*.
https://github.com/open-mmlab/mmdetection/issues/10052

Async & Coroutines

U. Technologies, “Unity - Scripting API: AsyncOperation,” *docs.unity3d.com*.
https://docs.unity3d.com/ScriptReference/AsyncOperation.html

JSON Serialization

U. Technologies, “Unity - Manual: JSON Serialization,” *docs.unity3d.com*.
https://docs.unity3d.com/2020.1/Documentation/Manual/JSONSerialization.html

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	**Definition**
ML model	A model that, based on our screenshots of Unity scenes will be able to use MMDET to find planes in real images
MMDET/MMDetection	A library used by Pytorch that allows a model to observe an image and based on its training find unique instances of objects in the image
COCO	Common Objects in Context —-- Specific formatting of data that describes objects within an image, used in the training of the model
GameObject	Type of object in Unity, used for scene manipulation
Canvas	Gameobject in unity that uses a Canvas component and is used to always create a background for a camera
Quad	GameObject in Unity that is used to create as a way to place an image into a scene in Unity
Prefab	Prefabricated object, unity GameObject type that is made from models. Used widely to manipulate an object and instantiate an object without risking losing original data of the model it was made from.