Software Requirements Specification

for

Flight Simulator

Version 3.0a

Prepared by

Team 7

Hari Ganesan - PES1UG20CS158

Harsh Jolad - PES1UG20CS160

Harshit T - PES1UG20CS161

Aryan Sharma - PES1UG20CS168

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Team 7	The first version of the SRS document has been drafted with all the requirements being incorporated into the document.	11/09/22
2.0	Team 7	The first version of the SRS document has been drafted with all the requirements being incorporated into the document.	24/10/22
3.0	Team 7	The first version of the SRS document has been drafted with all the requirements being incorporated into the document.	24/11/22

Introduction

1.1 Document Purpose

The product whose software requirements are specified in this document is Flight Simulator. The purpose of this document is to present a detailed description of the product, Flight Simulator. This document is intended to

- Explain the purpose and features of the product, Flight Simulator
- The constraints under which the product must operate
- How the product would respond to different users' requests.

The document's primary goal is to help the reader get a better understanding of the project. The document is intended for the developers of the product, the end users of the product who have been identified in the later sections, and to the professors who would review the project.

1.2 Product Scope

The software being developed is a surveillance Flight simulator. The product would simplify the whole process of creating virtual models and tours by:

- Simulate Flight patterns in different environmental conditions.
- Training and educative simulations

The points mentioned above would greatly simplify the creation of development of virtual tours of places with complex architecture more easily.
It also makes it easier to simulate and train Flight pilots.

1.3 Intended Audience and Document Overview

1.3.1 Intended Audience:

This document is primarily intended for the:

- Developers using this software.
- Software engineers who would work on further development of the project
- The professors who would review the document and finally,
- Flight enthusiasts and pilots.

1.3.2 Document Overview:

The first chapter, that is the Introduction section of the document is intended to introduce the reader to the product, Flight Simulator.
The second chapter, Overall Description section of SRS document provides an overview of the overall functionality of the product. It describes the informal requirements.

The third chapter, Specific Requirements section, of SRS document is written primarily for the developers and describes in technical terms the details of the functionality of the product.
The second and the third chapter of the document describe the same software product but are intended for different audiences and thus use different language.

1.4 Definitions, Acronyms and Abbreviations

1	Industrialists	The main users of this applications to whom the suppliers are needed to get in various necessary components needed for their production and also to sell their finished product.
2	Suppliers	The small-scale business people who act as middlemen to supply the components as required by the industrialists and also to sell the product.
3	SRS	SRS stands for Software Requirement Specification. It is a document that completely describes all of the functions of a proposed system and the constraints under which it must operate.
4	Team Head	Team head is an individual who is responsible for all the actions undergoing under his/her team.

1.5 Document Conventions

Formatting Conventions:

- The font style for the headings of each section is Arial Bold and the font size is 14.
- The font style for the headings under each section is Arial Bold and the font size used is 11.
- For the remainder of the document, the font style is Arial and the font size is maintained at 11.
- Italics has been used to indicate comments.
- The text is single spaced and margins are maintained at 1'' separation.

1.6 References and Acknowledgments

1.6.1 References:

https://unity.com/learn

1.6.2 Acknowledgments:

We would like to express our thanks of gratitude to our Software Engineering teacher, "Dr.Sapna V M" for her guidance and support in completing this project. We are thankful to our teacher for her ongoing support during the project, from initial advice, and encouragement, which led to the final report of this project.

2. Overall Description

2.1 Product Perspective

The proposed product, Flight Simulator is a new self-contained product. The product, Flight Simulator will gather information about various environments that are needed to simulate real life physics to test a virtual Flight of certain specifications. The product will empower the user, be it a professional or a nonprofessional person to efficiently simulate different type of planes. The product, Flight Simulator is intended to define a development methodology for the user, beginning with the requirements phase and continuing through to the execution phase.
The product, Flight Simulator will implement the following functionalities:

- Creating 3D models
- Creating virtual tours
- Flight Simulations and training
- Wind and weather conditions

An important attribute of Flight Simulator is its independency from the kind of Operating System that the computer on which it runs, as it executes on a system independent browser.

2.2 Product Functionality

These are the major functionalities of the product; Flight Simulator will achieve:

- Creating 3D models
- Creating virtual tours
- Flight Simulations and training
- Flying pattern of Flight specified

2.3 Users and Characteristics

The system will support four types of user privileges:

- Plane Designers
- Pilots
- Plane Enthusiasts

The various users that we expect the software to be used by are:

1.	Plane Designers	Plane designers who can simulate how a plane will act in certain weather conditions
2.	Pilots	To simulate life like situations for training.
3	Plane Enthusiasts	User who enjoys flying.

Table no:3

All the above-mentioned users are assumed to have a minimal knowledge of the technical

aspects of a product.

2.4 Operating Environment

The software will be designed to work on any version of Android, IOS. The software is completely web based and runs on popular web browsers namely Firefox, chrome, internet explorer (IE8 and above).

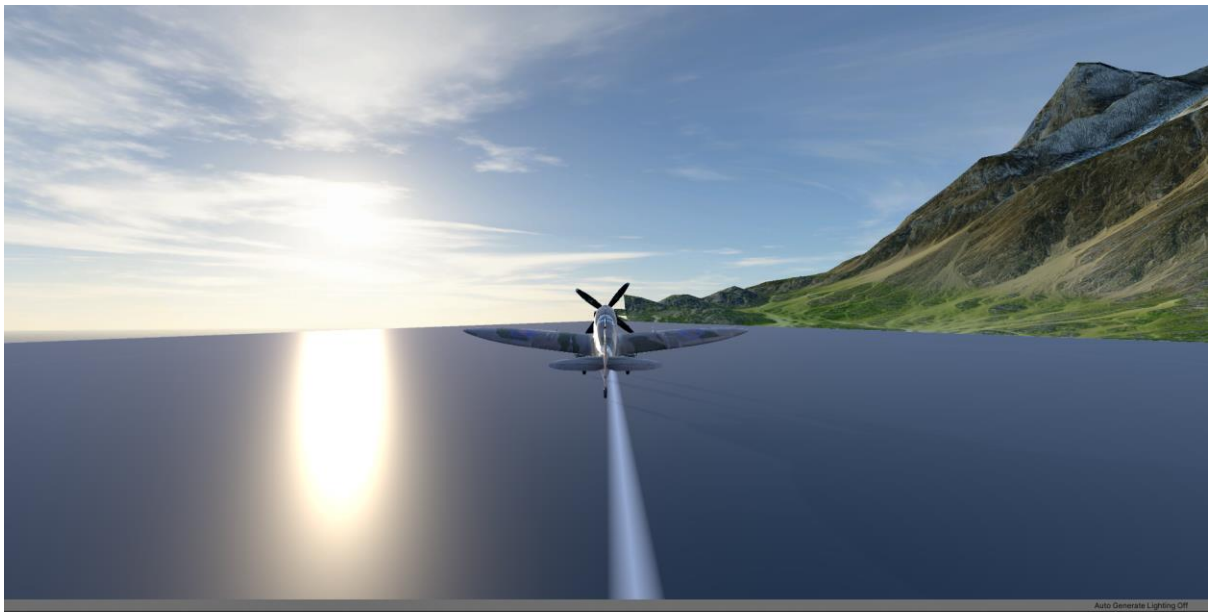
2.5 Design and Implementation Constraints

The application/ software has a few limitations. They are:

- Changing weather conditions when flying.
- Adjusting the speed linearly.
- Custom plane designs cannot be imported directly. Need FBX files.

2.6 User Documentation (Demo)

Basic Simulations



Keyboard Input

- **Space –** Acceleration
- **Shift + Space –** Deceleration
- **W, S –** Pitch Control
- **A, D –** Roll Control
- **Q,E –** Yaw Control
- **LCtrl –** Flaps Up
- **RCtrl –** Flaps Down

2.7 Assumptions and Dependencies

Assumptions

- The user is familiar with web-based software and how to fly a plane.
- The user's Computer can handle Nvidia PhysX system and has a graphics card with 2GB or higher VRAM.
- The user used keyboard as an input device.

2.8 Code Base

1. Airplane Controller

```
using System.Collections;
using System.Collections.Generic;
using UnityEditorInternal;
using UnityEngine;
using UnityEngine.SceneManagement;

public class AirplaneController : MonoBehaviour
{
    [SerializeField]
    float rollControlSensitivity = 0.2f;
    [SerializeField]
    float pitchControlSensitivity = 0.2f;
    [SerializeField]
    float yawControlSensitivity = 0.2f;
    [SerializeField]
    float thrustControlSensitivity = 0.01f;
    [SerializeField]
    float flapControlSensitivity = 0.15f;


    float pitch;
    float yaw;
    float roll;
    float flap;

    float thrustPercent;
    bool brake = false;
    bool permabrake = false;

    AircraftPhysics aircraftPhysics;
    Rotator propeller;

    private void Start()
```

```
{
    aircraftPhysics = GetComponent<AircraftPhysics>();
    propeller = FindObjectOfType<Rotator>();
    SetThrust(0);
}

private void Update()
{
    if (Input.GetKeyDown(KeyCode.R))
    {
        SceneManager.LoadScene(0);
    }

    if (Input.GetKey(KeyCode.Space))
    {
        SetThrust(thrustPercent + thrustControlSensitivity);
    }
    propeller.speed = thrustPercent * 1500f;

    if (Input.GetKeyDown(KeyCode.LeftShift))
    {
        thrustControlSensitivity *= -1;
    }

    if (Input.GetKeyDown(KeyCode.B))
    {
        permabrake=!permabrake;
    }

    if (Input.GetKeyDown(KeyCode.LeftControl))
    {
        if(flap<60f)
        {
            flap += flapControlSensitivity;
            //clamp
            flap = Mathf.Clamp(flap, 0f, Mathf.Deg2Rad * 40);
        }

    }

    if (Input.GetKeyDown(KeyCode.LeftAlt))
    {
        if(flap>0f)
        {
            flap -= flapControlSensitivity;
            //clamp
            flap = Mathf.Clamp(flap, Mathf.Deg2Rad * -40, 0f);
        }
```

```
        }

        pitch = pitchControlSensitivity * Input.GetAxis("Vertical");
        roll = rollControlSensitivity * Input.GetAxis("Horizontal");
        yaw = yawControlSensitivity * Input.GetAxis("Yaw");
    }

    private void SetThrust(float percent)
    {
        thrustPercent = Mathf.Clamp01(percent);
    }

    private void FixedUpdate()
    {
        aircraftPhysics.SetControlSurfecesAngles(pitch, roll, yaw, flap);
        aircraftPhysics.SetThrustPercent(thrustPercent);
        aircraftPhysics.permaBrake(permabrake);
    }
}
```

2. Aircraft Physics

```
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;

[RequireComponent(typeof(Rigidbody))]
public class AircraftPhysics : MonoBehaviour
{
    const float PREDICTION_TIMESTEP_FRACTION = 0.5f;

    [SerializeField]
    float thrust = 0;
    [SerializeField]
    List<AeroSurface> aerodynamicSurfaces = null;
    [SerializeField]
    List<ControlSurface> controlSurfaces = null;

    Rigidbody rb;
    float thrustPercent;
    BiVector3 currentForceAndTorque;

    public void SetThrustPercent(float percent)
    {
        thrustPercent = percent;
    }

    public void SetControlSurfecesAngles(float pitch, float roll, float yaw, float flap)
    {
```

```
        foreach (var controlSurface in controlSurfaces)
        {
            if (controlSurface.surface == null) return;
            switch (controlSurface.type)
            {
                case ControlSurfaceType.Pitch:
                    controlSurface.surface.SetFlapAngle(pitch * controlSurface.flapAngle);
                    break;
                case ControlSurfaceType.Roll:
                    controlSurface.surface.SetFlapAngle(roll * controlSurface.flapAngle);
                    break;
                case ControlSurfaceType.Yaw:
                    controlSurface.surface.SetFlapAngle(yaw * controlSurface.flapAngle);
                    break;
                case ControlSurfaceType.Flap:
                    controlSurface.surface.SetFlapAngle(flap * controlSurface.flapAngle);
                    break;
            }
        }
    }

    private void Awake()
    {
        rb = GetComponent<Rigidbody>();
    }

    private void FixedUpdate()
    {
        BiVector3 forceAndTorqueThisFrame =
            CalculateAerodynamicForces(rb.velocity, rb.angularVelocity, Vector3.zero, 1.2f, rb.worldCenterOfMass);

        Vector3 velocityPrediction = PredictVelocity(forceAndTorqueThisFrame.p);
        Vector3 angularVelocityPrediction = PredictAngularVelocity(forceAndTorqueThisFrame.q);

        BiVector3 forceAndTorquePrediction =
            CalculateAerodynamicForces(velocityPrediction, angularVelocityPrediction, Vector3.zero, 1.2f,
rb.worldCenterOfMass);

        currentForceAndTorque = (forceAndTorqueThisFrame + forceAndTorquePrediction) * 0.5f;
        rb.AddForce(currentForceAndTorque.p);
        rb.AddTorque(currentForceAndTorque.q);

        rb.AddForce(transform.forward * thrust * thrustPercent);
    }

    private BiVector3 CalculateAerodynamicForces(Vector3 velocity, Vector3 angularVelocity, Vector3 wind, float
airDensity, Vector3 centerOfMass)
    {
```

```
        BiVector3 forceAndTorque = new BiVector3();
        foreach (var surface in aerodynamicSurfaces)
        {
            Vector3 relativePosition = surface.transform.position - centerOfMass;
            forceAndTorque += surface.CalculateForces(-velocity + wind
                -Vector3.Cross(angularVelocity,
                relativePosition),
                airDensity, relativePosition);
        }
        return forceAndTorque;
    }

    private Vector3 PredictVelocity(Vector3 force)
    {
        return rb.velocity + Time.fixedDeltaTime * PREDICTION_TIMESTEP_FRACTION * (force / rb.mass +
Physics.gravity);
    }

    private Vector3 PredictAngularVelocity(Vector3 torque)
    {
        Quaternion inertiaTensorWorldRotation  = rb.rotation * rb.inertiaTensorRotation;
        Vector3 torqueInDiagonalSpace  = Quaternion.Inverse(inertiaTensorWorldRotation)  * torque;
        Vector3 angularVelocityChangeInDiagonalSpace;
        angularVelocityChangeInDiagonalSpace.x  = torqueInDiagonalSpace.x  / rb.inertiaTensor.x;
        angularVelocityChangeInDiagonalSpace.y  = torqueInDiagonalSpace.y  / rb.inertiaTensor.y;
        angularVelocityChangeInDiagonalSpace.z  = torqueInDiagonalSpace.z  / rb.inertiaTensor.z;

        return rb.angularVelocity  + Time.fixedDeltaTime * PREDICTION_TIMESTEP_FRACTION
            * (inertiaTensorWorldRotation  * angularVelocityChangeInDiagonalSpace);
    }


    public void permaBrake(bool isBraking) //increases drag on wheels
    {
        //add drag on wheels
        SphereCollider[] wheels = FindObjectsOfType<SphereCollider>();

        //change based on isBraking
        float friction;
        if (isBraking)
        {
            friction = 0.2f;
        }
        else
        {
            friction = 0f;
        }

        foreach (SphereCollider wheel in wheels)
```

```
        {
            wheel.material.dynamicFriction = friction;
        }
    }

#if UNITY_EDITOR
    public void CalculateCenterOfLift(out Vector3 center, out Vector3 force, Vector3 displayAirVelocity, float
displayAirDensity, float pitch, float yaw, float roll, float flap)
    {
        Vector3 com;
        BiVector3 forceAndTorque;
        if (aerodynamicSurfaces == null)
        {
            center = Vector3.zero;
            force = Vector3.zero;
            return;
        }

        if (rb == null)
        {
            com = GetComponent<Rigidbody>().worldCenterOfMass;
            foreach (var surface in aerodynamicSurfaces)
            {
                if (surface.Config != null)
                    surface.Initialize();
            }
            SetControlSurfecesAngles(pitch, roll, yaw, flap);
            forceAndTorque = CalculateAerodynamicForces(-displayAirVelocity, Vector3.zero, Vector3.zero,
displayAirDensity, com);
        }
        else
        {
            com = rb.worldCenterOfMass;
            forceAndTorque = currentForceAndTorque;
        }

        force = forceAndTorque.p;
        center = com + Vector3.Cross(forceAndTorque.p, forceAndTorque.q) / forceAndTorque.p.sqrMagnitude;
    }
#endif
}

[System.Serializable]
public class ControlSurface
{
    public AeroSurface surface;
    public float flapAngle;
    public ControlSurfaceType type;
```

```
}

public enum ControlSurfaceType { Pitch, Yaw, Roll, Flap }
```

3. Aero Surface

```
using System;
using UnityEngine;

public class AeroSurface : MonoBehaviour
{
    [SerializeField]  AeroSurfaceConfig config = null;

    public AeroSurfaceConfig Config => config;
    public float FlapAngle => flapAngle;
    public Vector3 CurrentLift { get; private set; }
    public Vector3 CurrentDrag { get; private set; }
    public Vector3 CurrentTorque { get; private set; }
    public bool IsAtStall { get; private set; }

    float flapAngle;

    float area;
    float correctedLiftSlope;
    float zeroLiftAoA;
    float stallAngleHigh;
    float stallAngleLow;
    float fullStallAngleHigh;
    float fullStallAngleLow;

    bool initialized;

    private void Awake()
    {
        Initialize();
    }

    public void Initialize()
    {
        area = config.chord * config.chord * config.aspectRatio;

        //correction for 2 parts of the wing instead of one
        if (gameObject.CompareTag("Wing"))
        {
            float aspectRatio = 2.1f;
            correctedLiftSlope = config.liftSlope * aspectRatio /
                (aspectRatio + 2 * (aspectRatio + 4) / (aspectRatio + 2));
        }
        else
        {
```

```
            correctedLiftSlope = config.liftSlope * config.aspectRatio /
                (config.aspectRatio + 2 * (config.aspectRatio + 4) / (config.aspectRatio + 2));
        }

        SetFlapAngle(0);
        initialized = true;
    }

    public void SetFlapAngle(float angle)
    {
        if (!gameObject.activeInHierarchy || config == null) return;

        flapAngle = Mathf.Clamp(angle, -Mathf.Deg2Rad * 50, Mathf.Deg2Rad * 50);

        float theta = Mathf.Acos(2 * config.flapFraction - 1);
        float flapEffectivness = 1 - (theta - Mathf.Sin(theta)) / Mathf.PI;
        float deltaLift = correctedLiftSlope * flapEffectivness * FlapEffectivnessCorrection(flapAngle) * flapAngle;

        float zeroLiftAoaBase = config.zeroLiftAoA * Mathf.Deg2Rad;
        zeroLiftAoA = zeroLiftAoaBase - deltaLift / correctedLiftSlope;

        float stallAngleHighBase = config.stallAngleHigh * Mathf.Deg2Rad;
        float stallAngleLowBase = config.stallAngleLow * Mathf.Deg2Rad;

        float clMaxHigh = correctedLiftSlope * (stallAngleHighBase - zeroLiftAoaBase) + deltaLift *
LiftCoefficientMaxFraction(config.flapFraction);
        float clMaxLow = correctedLiftSlope * (stallAngleLowBase - zeroLiftAoaBase) + deltaLift *
LiftCoefficientMaxFraction(config.flapFraction);

        stallAngleHigh = zeroLiftAoA + clMaxHigh / correctedLiftSlope;
        stallAngleLow = zeroLiftAoA + clMaxLow / correctedLiftSlope;

        float blendAngle = Mathf.Deg2Rad * Mathf.Lerp(8, 14, Mathf.Abs(Mathf.Abs(Mathf.Rad2Deg * flapAngle) -
50) / 50);
        fullStallAngleHigh = stallAngleHigh + blendAngle;
        fullStallAngleLow = stallAngleLow - blendAngle;
    }

    public BiVector3 CalculateForces(Vector3 worldAirVelocity, float airDensity, Vector3 relativePosition)
    {
        BiVector3 forceAndTorque = new BiVector3();
        if (!gameObject.activeInHierarchy || config == null) return forceAndTorque;

        if (!initialized) Initialize();

        Vector3 airVelocity = transform.InverseTransformDirection(worldAirVelocity);
        airVelocity = new Vector3(airVelocity.x, airVelocity.y);
```

```
    Vector3 dragDirection = transform.TransformDirection(airVelocity.normalized);
    Vector3 liftDirection = Vector3.Cross(dragDirection, transform.forward);

    float dynamicPressure = 0.5f * airDensity * airVelocity.sqrMagnitude;
    float angleOfAttack = Mathf.Atan2(airVelocity.y, -airVelocity.x);


    IsAtStall = !(angleOfAttack < stallAngleHigh && angleOfAttack > stallAngleLow);
    Vector3 aerodynamicCoefficients = CalculateCoefficients(angleOfAttack);
    CurrentLift = liftDirection * aerodynamicCoefficients.x * dynamicPressure * area;
    CurrentDrag = dragDirection * aerodynamicCoefficients.y * dynamicPressure * area;
    CurrentTorque = -transform.forward * aerodynamicCoefficients.z * dynamicPressure * area * config.chord;

    forceAndTorque.p += CurrentDrag + CurrentLift;
    forceAndTorque.q += Vector3.Cross(relativePosition, forceAndTorque.p);
    forceAndTorque.q += CurrentTorque;

    return forceAndTorque;
}

private Vector3 CalculateCoefficients(float angleOfAttack)
{
    Vector3 aerodynamicCoefficients;
    if (angleOfAttack < stallAngleHigh && angleOfAttack > stallAngleLow)
    {
        aerodynamicCoefficients = CalculateCoefficientsAtLowAoA(angleOfAttack);
    }
    else
    {
        if (angleOfAttack > fullStallAngleHigh || angleOfAttack < fullStallAngleLow)
        {
            aerodynamicCoefficients = CalculateCoefficientsAtStall(angleOfAttack);
        }
        else
        {
            Vector3 aerodynamicCoefficientsLow;
            Vector3 aerodynamicCoefficientsStall;
            float lerpParam;

            if (angleOfAttack > stallAngleHigh)
            {
                aerodynamicCoefficientsLow = CalculateCoefficientsAtLowAoA(stallAngleHigh);
                aerodynamicCoefficientsStall = CalculateCoefficientsAtStall(fullStallAngleHigh);
                lerpParam = (angleOfAttack - stallAngleHigh) / (fullStallAngleHigh - stallAngleHigh);
            }
            else
            {
                aerodynamicCoefficientsLow = CalculateCoefficientsAtLowAoA(stallAngleLow);
                aerodynamicCoefficientsStall = CalculateCoefficientsAtStall(fullStallAngleLow);
```

```
                lerpParam = (angleOfAttack - stallAngleLow)  / (fullStallAngleLow - stallAngleLow);
            }
            aerodynamicCoefficients = Vector3.Lerp(aerodynamicCoefficientsLow, aerodynamicCoefficientsStall,
lerpParam);
        }
    }
    return aerodynamicCoefficients;
}

private Vector3 CalculateCoefficientsAtLowAoA(float angleOfAttack)
{
    float liftCoefficient = correctedLiftSlope * (angleOfAttack - zeroLiftAoA);
    float inducedAngle = liftCoefficient / (Mathf.PI * config.aspectRatio);
    float effectiveAngle = angleOfAttack - zeroLiftAoA - inducedAngle;

    float tangentialCoefficient = config.skinFriction * Mathf.Cos(effectiveAngle);

    float normalCoefficient = (liftCoefficient +
        Mathf.Sin(effectiveAngle) * tangentialCoefficient) / Mathf.Cos(effectiveAngle);
    float dragCoefficient = normalCoefficient * Mathf.Sin(effectiveAngle) + tangentialCoefficient *
Mathf.Cos(effectiveAngle);
    float torqueCoefficient = -normalCoefficient * TorqCoefficientProportion(effectiveAngle);

    return new Vector3(liftCoefficient, dragCoefficient, torqueCoefficient);
}

private Vector3 CalculateCoefficientsAtStall(float angleOfAttack)
{
    float liftCoefficientLowAoA;
    if (angleOfAttack > stallAngleHigh)
    {
        liftCoefficientLowAoA = correctedLiftSlope * (stallAngleHigh  - zeroLiftAoA);
    }
    else
    {
        liftCoefficientLowAoA = correctedLiftSlope * (stallAngleLow  - zeroLiftAoA);
    }
    float inducedAngle = liftCoefficientLowAoA / (Mathf.PI * config.aspectRatio);

    float lerpParam;
    if (angleOfAttack > stallAngleHigh)
    {
        lerpParam = (Mathf.PI / 2 - Mathf.Clamp(angleOfAttack, -Mathf.PI / 2, Mathf.PI / 2))
            / (Mathf.PI / 2 - stallAngleHigh);
    }
    else
    {
        lerpParam = (-Mathf.PI / 2 - Mathf.Clamp(angleOfAttack, -Mathf.PI / 2, Mathf.PI / 2))
```

```
            / (-Mathf.PI / 2 - stallAngleLow);
        }
        inducedAngle = Mathf.Lerp(0, inducedAngle, lerpParam);
        float effectiveAngle = angleOfAttack - zeroLiftAoA - inducedAngle;


        float normalCoefficient = FrictionAt90Degrees() * Mathf.Sin(effectiveAngle) *
            (1 / (0.56f + 0.44f * Mathf.Abs(Mathf.Sin(effectiveAngle))) -
            0.41f * (1 - Mathf.Exp(-17 / config.aspectRatio)));
        float tangentialCoefficient = 0.5f * config.skinFriction * Mathf.Cos(effectiveAngle);

        float liftCoefficient = normalCoefficient * Mathf.Cos(effectiveAngle) - tangentialCoefficient *
Mathf.Sin(effectiveAngle);
        float dragCoefficient = normalCoefficient * Mathf.Sin(effectiveAngle) + tangentialCoefficient *
Mathf.Cos(effectiveAngle);
        float torqueCoefficient = -normalCoefficient * TorqCoefficientProportion(effectiveAngle);

        return new Vector3(liftCoefficient, dragCoefficient, torqueCoefficient);
    }

    private float TorqCoefficientProportion(float effectiveAngle)
    {
        return 0.25f - 0.175f * (1 - 2 * Mathf.Abs(effectiveAngle) / Mathf.PI);
    }

    private float FrictionAt90Degrees()
    {
        return 1.98f - 4.26e-2f * flapAngle * flapAngle + 2.1e-1f * flapAngle;
    }

    private float FlapEffectivnessCorrection(float flapAngle)
    {
        return Mathf.Lerp(0.8f, 0.4f, (flapAngle * Mathf.Rad2Deg - 10) / 50);
    }

    private float LiftCoefficientMaxFraction(float flapFraction)
    {
        return Mathf.Clamp01(1 - 0.5f * (flapFraction - 0.1f) / 0.3f);
    }
}
```

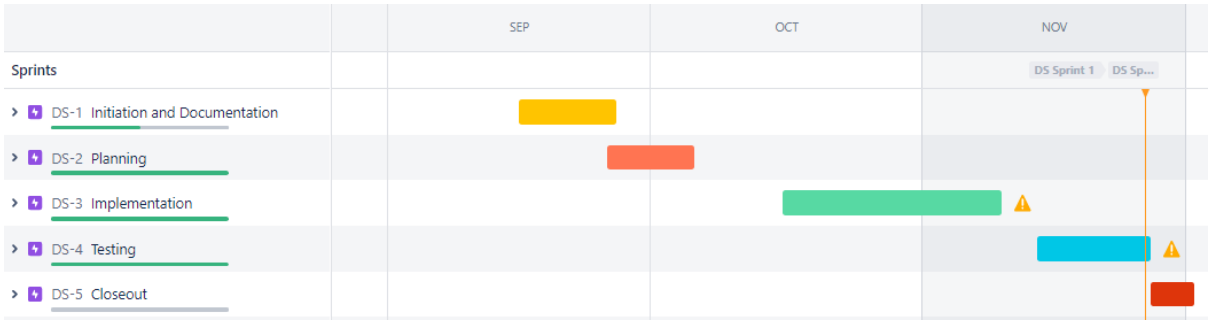
2.9 Basic SE Information

2.9.1 Software Life Cycle Methodology

Agile Methodology

2.9.2 WBS Using Jira

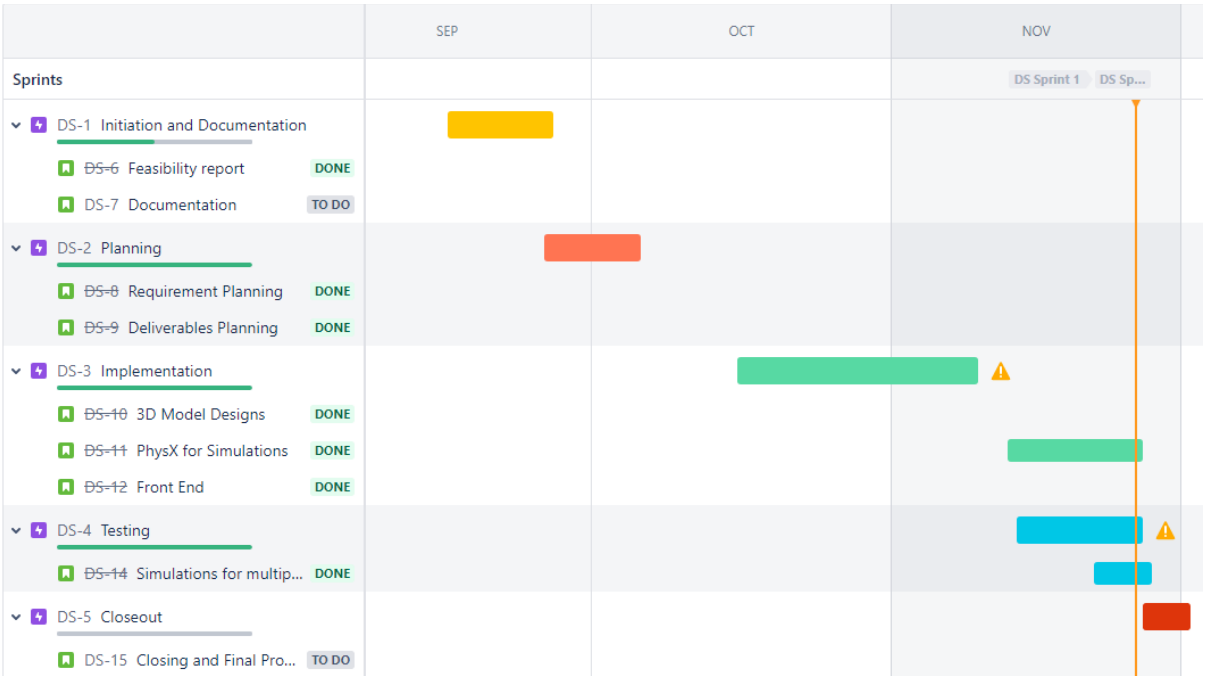
Road Map



Issues Log

Event	Recipient	Actions
Issue Created	All Watchers, Current Assignee, Reporter	•••
Issue Updated	All Watchers, Current Assignee, Reporter	•••
Issue Assigned	All Watchers, Current Assignee, Reporter	•••
Issue Deleted	All Watchers, Current Assignee, Reporter	•••
Issue Moved	All Watchers, Current Assignee, Reporter	•••
Issue Commented	All Watchers, Current Assignee, Reporter	•••
Issue Comment Edited	All Watchers, Current Assignee, Reporter	•••
Work Logged On Issue	All Watchers, Current Assignee, Reporter	•••
Issue Worklog Updated	All Watchers, Current Assignee, Reporter	•••
Issue Worklog Deleted	All Watchers, Current Assignee, Reporter	•••

Plan



2.9.3 Coding Practices Used

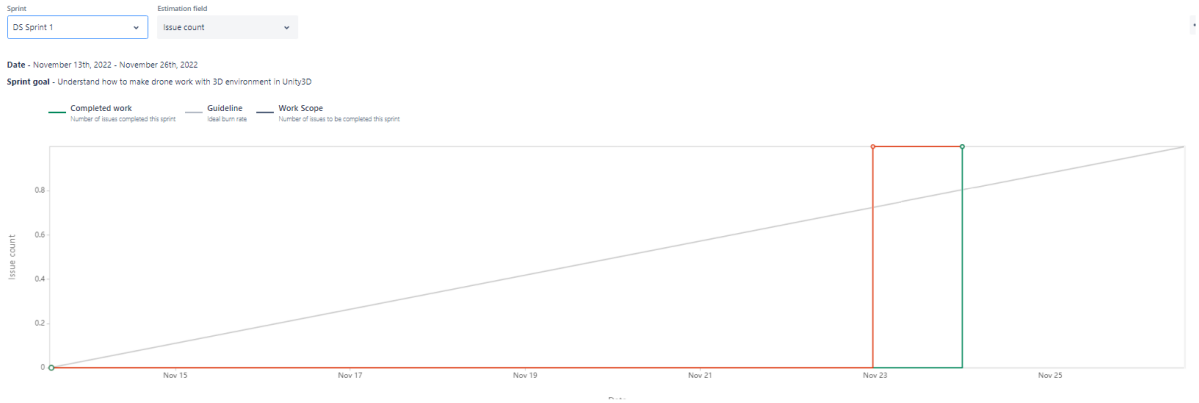
- Code Simplicity
- Limited use of global variables
- Meaningful naming convention used
- Proper indentations used
- Exception handling
- Well documented code

2.9.4 Software Change Management (SCM)

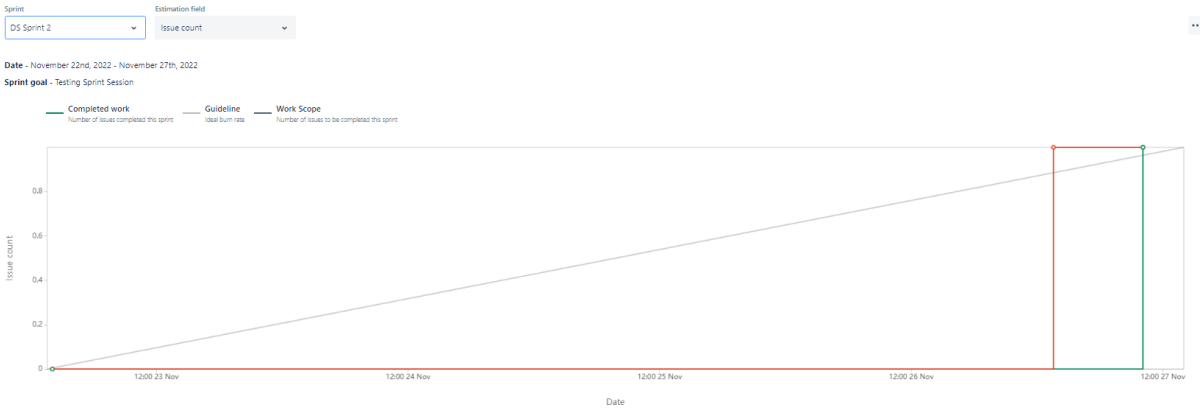
Version 1- Flight Simulator with basic colliders and drag.
Version 2- Flight Simulator with previous features + thrust system.
Version 3- Flight Simulator with landscape map.
Version 3a- Flight Simulator with flap system and braking system.
These were stored and worked on using GitHub. No branches were made, only Main Branch changes. Versioning is used. No releases yet.

2.9.5 JIRA Report

1. Sprint 1



2. Sprint 2



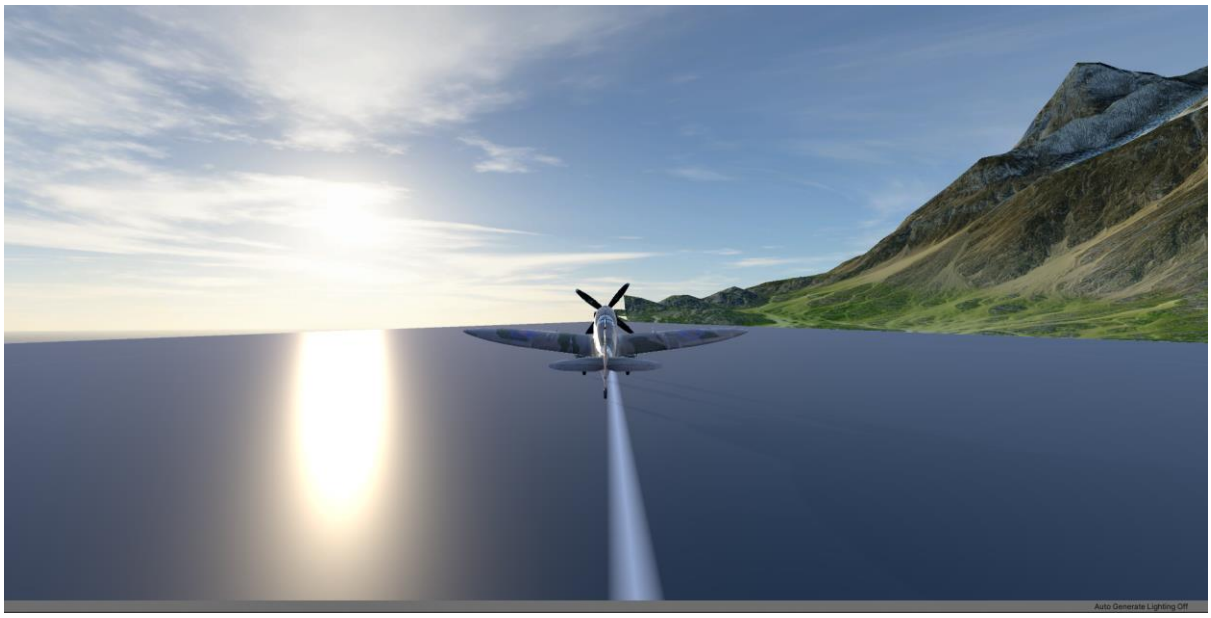
3. Specific Requirements

3.1. External Interface Requirements

3.1.1. User Interfaces

The user interface design is simple and clear. One can very easily view the live feed and operate the plane. In this product, Flight Simulator an individual can create and upload their aero-plane models to the system.

Sample Screenshots:



3.1.2. Hardware Interfaces

- Computer
- Keyboard /Controller
- Virtual Headset

3.1.3. Software Interfaces

- Unity3d
- Blender
- Nvidia PhysX

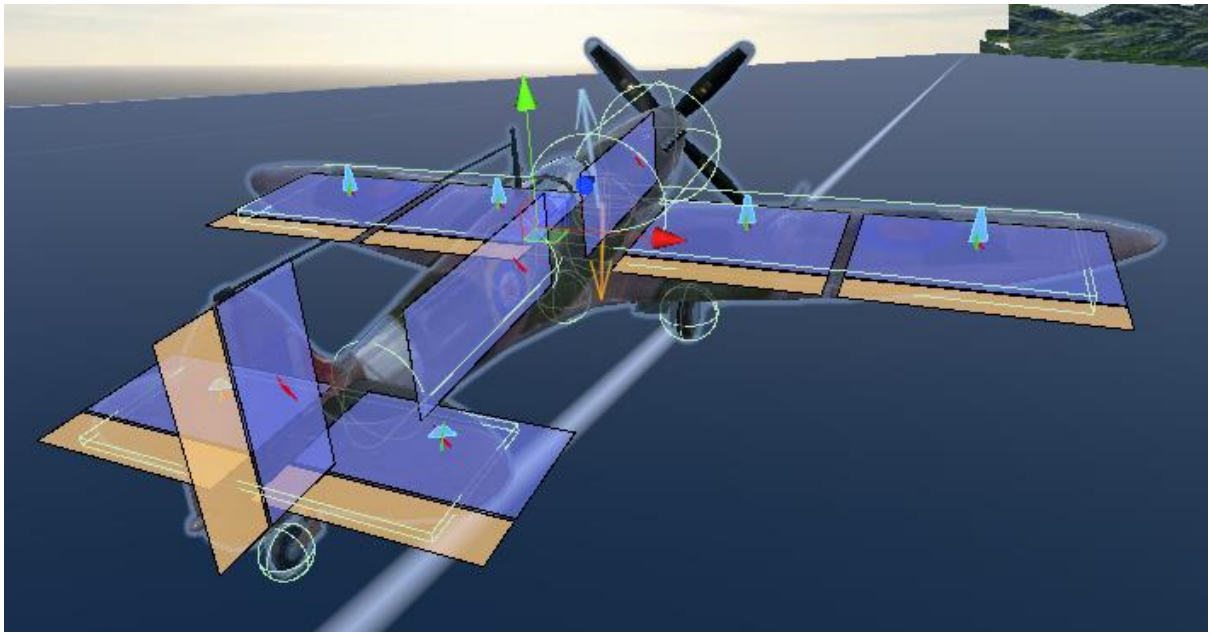
3.1.4. Communications Interfaces

None

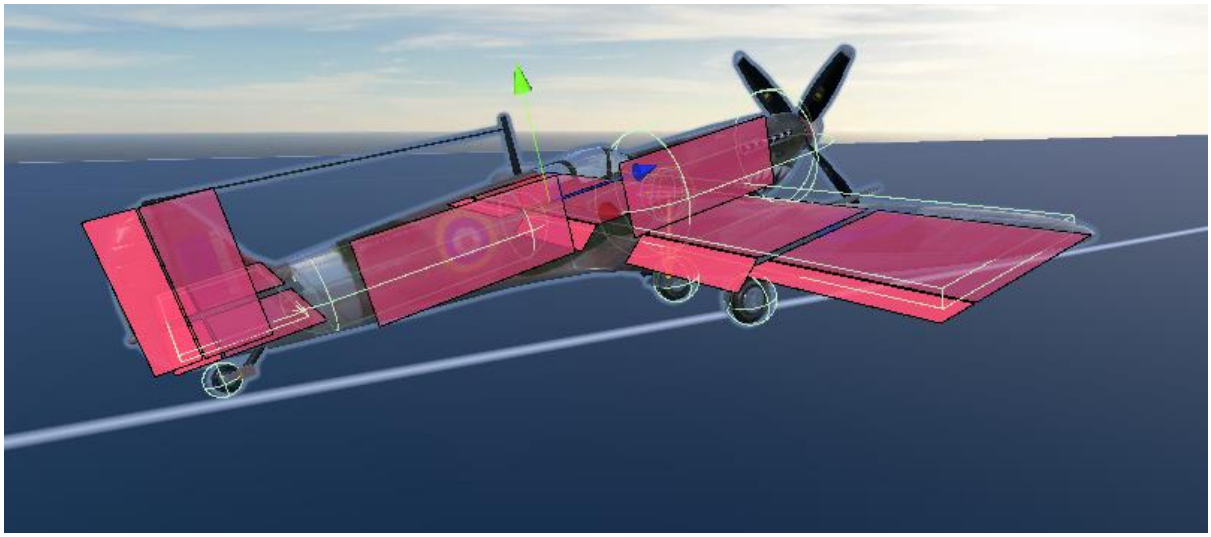
3.2. Functional Requirements

The Flight Simulator being developed is generic. It can be used to simulate any kind of aircraft with the given thrust and drag data(active aero).

Aero Not Active



Active Aero



Flight


Auto Generate Lighting Off