BACHELOR'S THESIS PROJECT

TITLE: Simulation of a TaxiBot Operation for First Person Training

DEGREE: Double Bachelor's Degree in Aerospace Systems Engineering and Telecommunications Systems Engineering

AUTHOR: Gabriel Sebastián Font Rojas

DIRECTOR: Jordi Pons Prats

CODIRECTOR: Francisco Javier Mora Serrano

DATE: September 2021

Title: Simulation of a TaxiBot Operation for First Person Training

Author: Gabriel Sebastián Font Rojas

Director: Jordi Pons Prats

Codirector: Francisco Javier Mora Serrano

Date: September 2021

Overview

The TaxiBot is a semi-robotic electric vehicle capable of towing aircraft from the boarding gate at the terminal to a runway holding position. The TaxiBot is controlled by the pilot using the regular tiller for nose wheel steering, rudder pedals and brakes within the cockpit. This is one of multiple solutions to reduce fuel consumption on ground operations. This project focuses on the development of a piece of software that simulates the use of a TaxiBot to perform a taxi procedure on a recreated airport. The purpose of the experience is to practice the operation in a risk-free virtual environment.

The TaxiBot is relatively new in aviation and it is not featured in any openly available flight simulation software today. The design phase of the experience entails several important decisions. The reasoning and criteria behind these decisions and the requirements or optional tasks that are derived as a consequence are all valuable information. Along this document, a description of the process that was followed in order to develop the experience, from creating or acquiring assets to scripting the forces that move the simulated vehicles and creating the evaluation systems, is presented.

The same virtual environment could be expanded to provide different experiences, test new vehicles or procedures (as is the case of the TaxiBot) without needing to make another fresh start.

To validate the experience, several users with different expertise levels on the subject tested the application and provided feedback to evaluate different key points like their engagement, what knowledge was gained following the training session and which aspects they would like to see improved.

The experience was well received in the validation session. However, a door remains open to keep researching the subject as a more systematic validation method could provide numerical data on the effectiveness of similar experiences. Additionally, a few proposed functionalities that were not carried out or newer suggestions could be implemented at a later stage to expand on the ideas and developments accomplished in this project.

Título: Simulación de Rodaje con el Taxibot para Entrenamiento en Primera Persona.

Autor: Gabriel Sebastián Font Rojas

Director: Jordi Pons Prats

Codirector: Francisco Javier Mora Serrano

Fecha: Septiembre de 2021

Resumen

El TaxiBot es un vehículo eléctrico semirrobótico capaz de remolcar aviones desde la puerta de embarque de la terminal hasta una posición de espera en la pista. El TaxiBot en teoría es transparente al piloto ya que se controla con el timón y los frenos habituales de la cabina. Es una de las múltiples soluciones para reducir el consumo de combustible en las operaciones en tierra. Este proyecto se centra en el desarrollo de un software que simula el uso de un TaxiBot para realizar un procedimiento de taxi en un aeropuerto recreado. El objetivo de la experiencia es practicar la operación en un entorno virtual sin riesgos.

El TaxiBot fue desarrollado por Israel Aerospace Industries y consiguió una primera certificación en 2014. Por ser relativamente nuevo en la aviación, no aparece en ningún software de simulación de vuelo disponible en la actualidad. La transparencia para el piloto de usar un TaxiBot es un argumento de venta y las diferencias con respecto a un rodaje convencional ameritan entrenamiento específico para pilotos y operadores. La fase de diseño de la experiencia conlleva varias decisiones importantes. El razonamiento y los criterios en los que se basan estas decisiones, así como los requisitos o las tareas opcionales que se derivan como consecuencia, constituyen una información valiosa. A lo largo de este documento se presenta una descripción del proceso que se ha seguido para desarrollar la experiencia, desde la creación o adquisición de activos hasta el scripting de las fuerzas que mueven los vehículos simulados y la creación de los sistemas de evaluación. El mismo entorno virtual podría ampliarse para ofrecer diferentes experiencias, probar nuevos vehículos o procedimientos (como es el caso del TaxiBot) sin necesidad de volver a empezar.

Para validar la experiencia, varios usuarios con diferentes niveles de experiencia en el tema probaron la aplicación y proporcionaron comentarios para evaluar diferentes puntos clave como su compromiso, los conocimientos adquiridos tras la sesión de formación y los aspectos que les gustaría ver mejorados.

La experiencia fue bien recibida en la sesión de validación. Sin embargo, queda una puerta abierta para seguir investigando el tema, ya que un método de validación más sistemático y con más participantes puede aportar datos numéricos sobre la eficacia de esta experiencia u otras similares. Además, algunas funcionalidades propuestas que no se llevaron a cabo o sugerencias más novedosas podrían implementarse en una etapa posterior para ampliar las ideas y desarrollos realizados en este proyecto.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1. Motivation

Aviation has a high environmental impact [1] so new systems to reduce fuel consumption and emissions are constantly being developed. As shown in Figure1.1, around 30% of emissions in a large airport can be traced to taxi operations [2]. Alternative ground propulsion systems that can mitigate ground movement related consumption as well as provide other advantages are being implemented in several airports [3, 4].

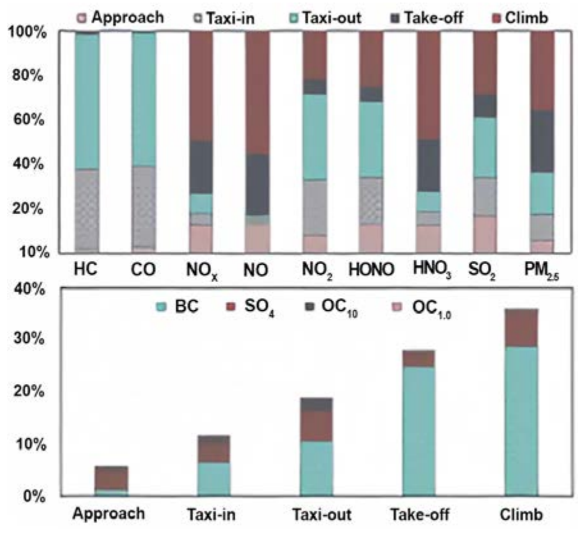


Figure 1.1. Xu et al. (2020). Aircraft Emissions in Shanghai Pudong International Airport [2].

The TaxiBot is one of these proposed solutions to the mentioned problem as it is a vehicle that can tow medium to large aircraft at conditions comparable to those of a regular taxi procedure. Instead of using jet engines and burning fuel, the TaxiBot uses a hybrid electric engine, which reduces noise levels and total fuel consumption considerably. The first TaxiBot model has been approved and certified for some of the most used family of aircraft. The benefits and drawbacks of TaxiBot utilization have been analyzed and each new deployment on an existing airport is studied laboriously [5]. However, while the technology has been tested, the TaxiBot has just been deployed in very few airports around the world. Pilots must learn and accustom themselves to the differences when performing the procedure with the presented change and airports need to make studies of the effect it could have on capacity and safety.

In this project, a first person simulation of the TaxiBot experience is developed in order to delve deeper into these aspects, attempting to solve some of the questions and also discover unexpected issues or benefits that may arise.

Several flight simulation programs exist nowadays and they recreate all the maneuvers in a regular flight extremely well. Most courses in pilot training employ flight simulators as they can provide very similar environments without any risk involved at a fraction of the cost. Some airlines are even implementing Virtual Reality technologies to train pilots, cabin crew, and operators.

The experience we wish to accomplish could be used to train pilots in the use of this new technology. The following research will focus on testing the experience, to study its effectiveness as a learning tool.

Most of these technical aspects were present in the initial project proposal. It was initially considered that the taxi procedure could performed entirely from the point of view of the TaxiBot operator. The operator controls the vehicle during pushback and after disconnecting at the holding position, so they could still make use of a training experience in a virtual environment. There has been at least one experiment to use tugs (driven by operators with reduced visibility and ergonomics of regular tugs) to tow aircraft to the runway [6]. This practice was later discontinued without an official statement behind the motives.

On a personal note, I have enjoyed game development as a hobby for a few years, so this project seemed very appealing. I have experience working with Unity (a popular Game Engine with free and paid options), programming applications in C#, and 3D modeling with the help of a couple of programs. I would like to test how technology has made the development of such a piece of software accessible to a single developer in the current age. Shortcuts should be taken when possible to facilitate the process (in programming, modeling, animating and other tasks). Newer solutions will be implemented with the intention of commenting on the feasibility of such an endeavor.

1.2. Objectives

The main objective of this project is to develop a simulator of TaxiBot (tractor vehicle) Operations for First Person Training with the aim of evaluating and improving the human factors in terms of ergonomics and required skills. This objective can be broken down in the following sub-objectives:

1. Review of the background and state of the art, including precedent simulators, current training material, procedures for conventional and tractor vehicle operations, and design of interactive experiences with educational purposes.
2. Modular design of the application to provide a suitable framework for straightforward replication of the simulator for different airports and vehicles. This design choice will also ease adding other functionalities at a later phase.
3. Implementation of a specific use case, including:
    - 3.1. First person vision of the TaxiBot driver and the aircraft pilots.
    - 3.2. Visualization and execution of the nose landing gear locking process, taxi procedure from boarding gate up to the holding position, unlocking and return route.

3.2. Potential inclusion different factors like different environments, vehicle dynamics, visibility and weather conditions, different plane sizes.

4. Evaluation of the best compromise between realism, model details, fluidity, cost, and portability of the simulation.

1.3. Research Methodology

The main adapted methodology in this project is the Design Science Research Methodology (DSRM) divided into 5 stages: (1) Identification of the observed problem and motivations, (2) Definition of the objectives for a potential solution, (3) Design and development, (4) Demonstration, and (5) Evaluation (Peffers et al., 2006) [7], (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007) [8], (Walls, Widmeyer, & El Sawy, 1992) [9], (Hevner, March, Park, & Ram, 2004) [10]. Figure 1.2 shows the project stages, detailing the different tools used in the corresponding established activities.

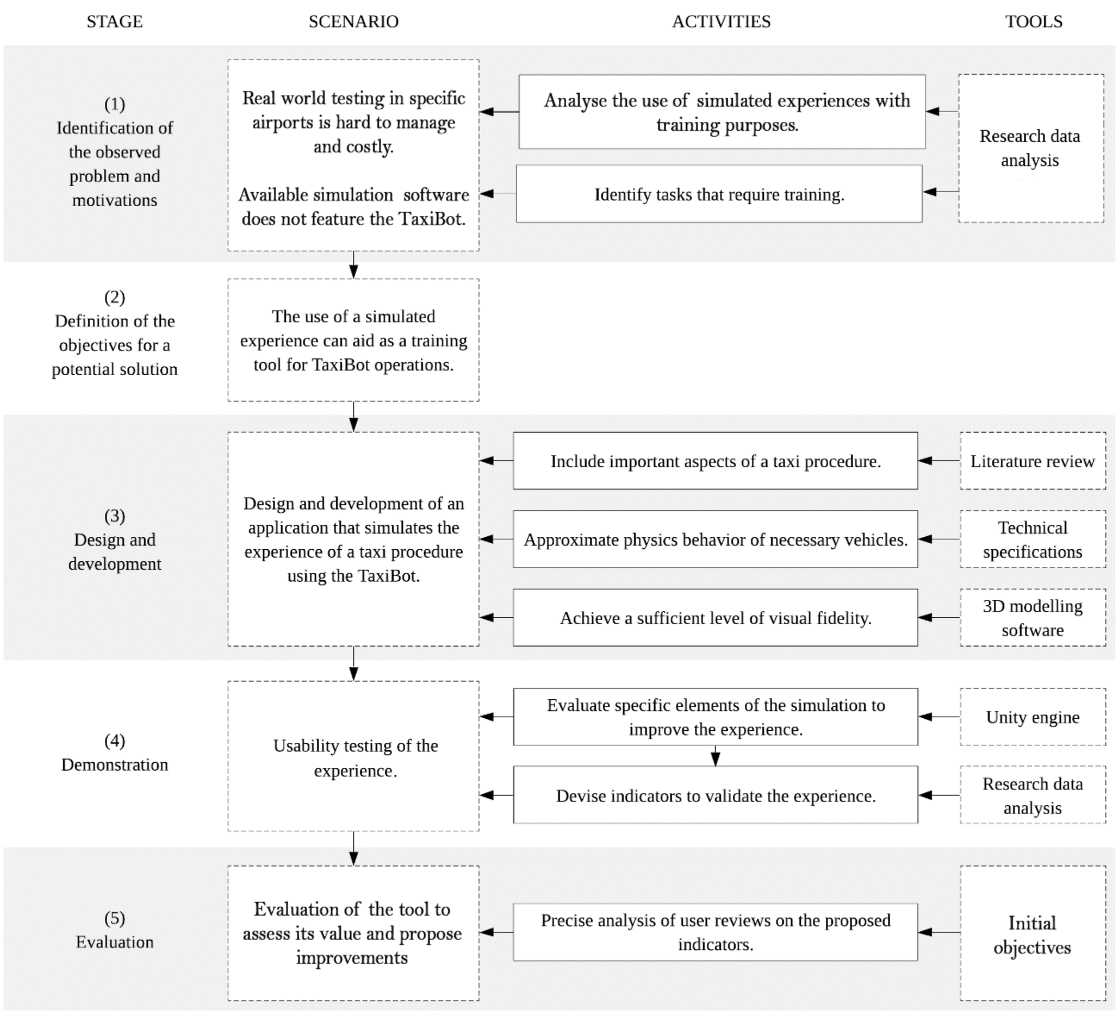

STAGE
SCENARIO
ACTIVITIES
TOOLS
(1) Identification of the observed problem and motivations
Real world testing in specific airports is hard to manage and costly.
Available simulation software does not feature the TaxiBot.
Analyse the use of simulated experiences with training purposes.
Identify tasks that require training.
Research data analysis
(2) Definition of the objectives for a potential solution
The use of a simulated experience can aid as a training tool for TaxiBot operations.
(3) Design and development
Design and development of an application that simulates the experience of a taxi procedure using the TaxiBot.
Include important aspects of a taxi procedure.
Approximate physics behavior of necessary vehicles.
Achieve a sufficient level of visual fidelity.
Literature review
Technical specifications
3D modelling software
(4) Demonstration
Usability testing of the experience.
Evaluate specific elements of the simulation to improve the experience.
Devise indicators to validate the experience.
Unity engine
Research data analysis
(5) Evaluation
Evaluation of the tool to assess its value and propose improvements
Precise analysis of user reviews on the proposed indicators.
Initial objectives

Figure 1.2. Research methodology project stages (own image).

In the first stage, an analysis of available resources was conducted in order to check the use of simulated experiences like virtual reality applications as training tools. As for background, the use of flight simulation software for training and entertainment is an established subject. The identified problem is that use of the TaxiBot is relatively new and there are few resources openly available about it. For this stage, documents on the topics of simulation and serious games were considered. Additionally, unsuccessful attempts to contact Smart Airport Systems, one of the companies in charge of TaxiBot deployment were made. Information about current training methods for operators and pilots will prove to be rather useful for the rest of the project.

The main objectives were defined in the second stage. These objectives point to the development of an application that can provide an engaging experience that can be used as a training tool. Furthermore, a requirement that the application should be expandable was considered, since many aspects of the design and development phases are not specific to TaxiBot operation.

In the third stage, specifications about the procedure, vehicles, and environment were gathered. Using this information, specific aspects can be defined in order to design a competent experience. During development of the application, specifications that were not foreseen became necessary so further literature review was required.

In the fourth stage, the training experience was implemented and tested by a few users with different levels of experience on the subject. In the fifth stage, a standardized questionnaire was adapted and employed to evaluate the training experience in terms of its technical features and usability. The results obtained were discussed in order to determine positive aspects and possible improvement opportunities.

1.4. Document Structure

This document is structured in the following way:

Chapter 1: Introduction

In this chapter, basic ideas that motivate the project and its objectives are established.

Chapter 2: State of the Art

This chapter offers a dive into the state of the subject today. A study of the available technologies that answer to a similar call is also carried out. This part will work as a foundation for the following chapter.

Chapter 3: Experience Design

This chapter establishes the foreseen requirements of the experience and the design of the systems that will be implemented in the application.

Chapter 4: Unity Implementation

This chapter includes all the information related to the integration of features. The steps taken to create the virtual environment, physics simulation and supplementary systems are disclosed in detail. It includes integration of existing solutions, development of new solutions and obstacles found along the process.

Chapter 5: Validation

The process through which relevant data is gathered in order to measure the value of the experience.

Chapter 6: Conclusions and Future Lines of Research

This section contains the conclusions made throughout the development of the project, based on the initial objectives and the process itself.

CHAPTER 2. STATE OF THE ART

2.1. Taxi Operations

Autonomy is a popular proposed approach to reduce taxi times, delays, and human errors. Research and development is being done to improve these factors. A concept of the future of taxi operations proposes modifying the TaxiBot or a similar vehicle to implement trajectory-based surface operations without modifying the aircraft [11]. This process led to design of a Human Machine Interface to aid pilots in performing operations. With its use, one pilot would taxi and another one would monitor tasks remotely.

The TaxiBot is an important milestone to tackle the subject of autonomy. A remotely controlled or unmanned TaxiBot is achievable but a long development and verification phase is expected [5]. Although testing of an unmanned vehicle was estimated to start in 2020, no statement has been released yet.

The TaxiBot entered regular operation at Frankfurt Airport in 2015. The vehicle has been deployed in Indira Gandhi International Airport (Delhi) and Kempegowda International Airport Bengaluru (Bangalore) for a few years. Indira Gandhi completed 1000 movements using the TaxiBot in May, 2021 [12]. Amsterdam Airport Schiphol has been studying and testing a deployment since July 2020. A program to achieve sustainable taxiing by 2030 expects several TaxiBots to join the fleet at the start of 2022. Developments on the subject of autonomy or control from a remote location is also slowly being carried out. However, no operation or test has been disclosed yet.

2.2. The Use of Simulation in Aviation

The use of flight simulation software in the aviation industry is extremely common. The FS1 Flight Simulator was launched in 1979 for the Apple II. It placed the user in the cockpit of a biplane fighter so the experience could be considered to have a first person view. Throughout the following decades, more advanced software was developed to aid in training, development or simply user entertainment.

Many companies offer full flight simulator devices that can re-create aircraft procedures and motion to achieve an experience incredibly similar to their real-world counterpart. The training effectiveness of flight simulators has been proven decades ago [13] and their use is well established in pilot training.

These devices are often very expensive and the software underneath is proprietary, so implementation of new technology or control over different experiences is limited.

In April 2021, the European Union Aviation Safety Agency (EASA) approved the first Virtual Reality based Flight Simulation Training Device (FSTD) [14]. The device was developed by VRM Switzerland and it is currently used to train rotorcraft pilots. The adoption of VR enables more cost-effective experiences that can be used to complement full flight simulators and other training tools.

As opposed to approved FSTDs, amateur flight simulation is very popular and several options are readily available for ordinary users and not just pilots in training.

2.3. Software Examples

Examples of relevant tools that make use of simulation and that can be used with training purposes are provided in this section.

2.3.1. Tecknotrove: Tecknosim Pushback Simulator 2017

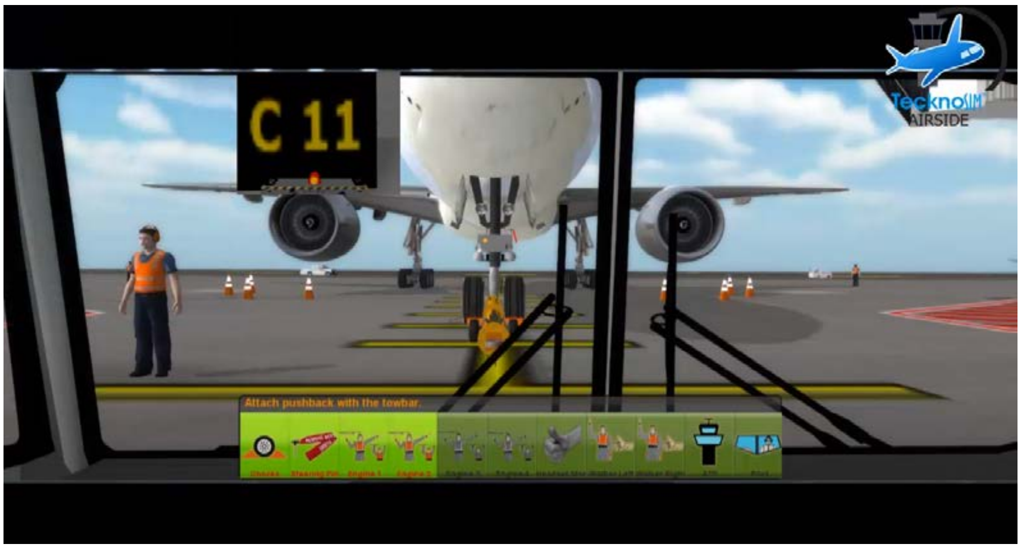


Figure 2.1. Tecknotrove System (I) Pvt Ltd. (2017). Pushback operator first person view. https://www.youtube.com/watch?v=-TzDYp65zo4 [15]

This simulator was designed specifically to train pushback operators. The user controls the vehicle through the use of a recreated dashboard of a pushback tractor. The operator seat is mounted on an electric motion platform to transfer movement to the user and therefore, increase simulation fidelity.

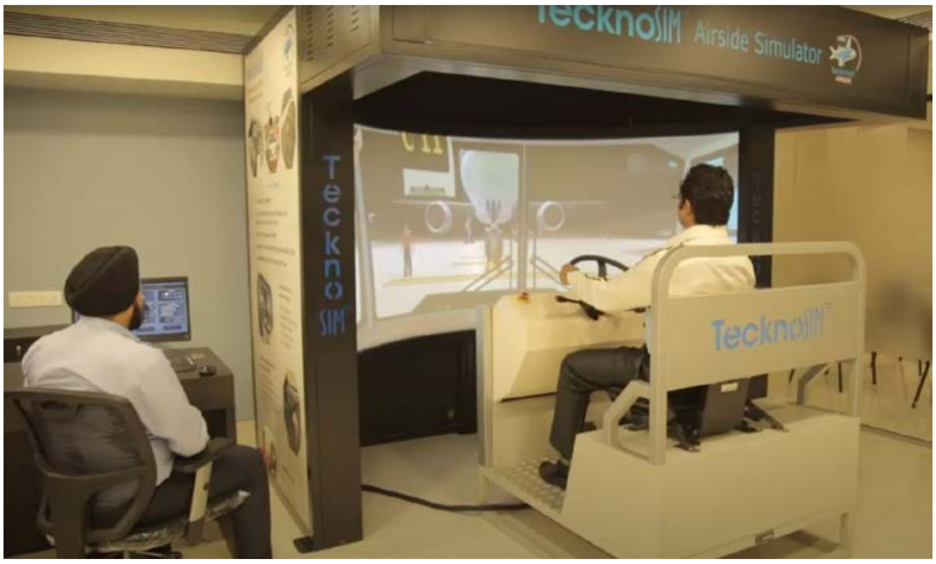


Figure 2.2. Tecknotrove System (I) Pvt Ltd. (2017). Electric motion platform and curved projection display. https://www.youtube.com/watch?v=-TzDYp65zo4 [15]

Certain aspects of the experience scenario can be configured such as the direction to push the aircraft, whether the maneuver includes towing, time of day and weather. An instructor can be present to supervise the experience on a connected workstation. The software can provide performance evaluation reports that can disclose specific infractions that might have been committed by the user.

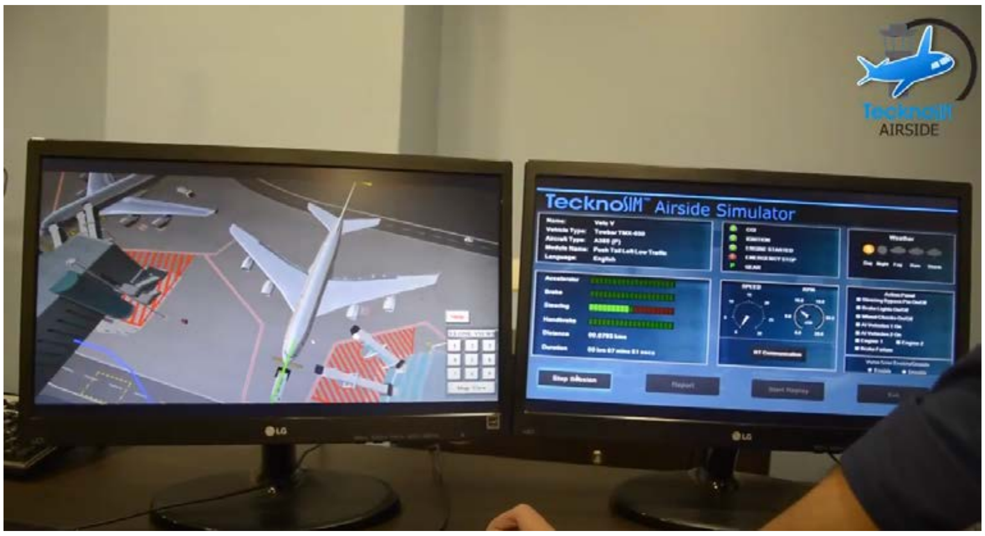


Figure 2.3. Tecknotrove System (I) Pvt Ltd. (2017). Supervisor workstation. https://www.youtube.com/watch?v=-TzDYp65zo4 [15]

2.3.2 KLM VR Training

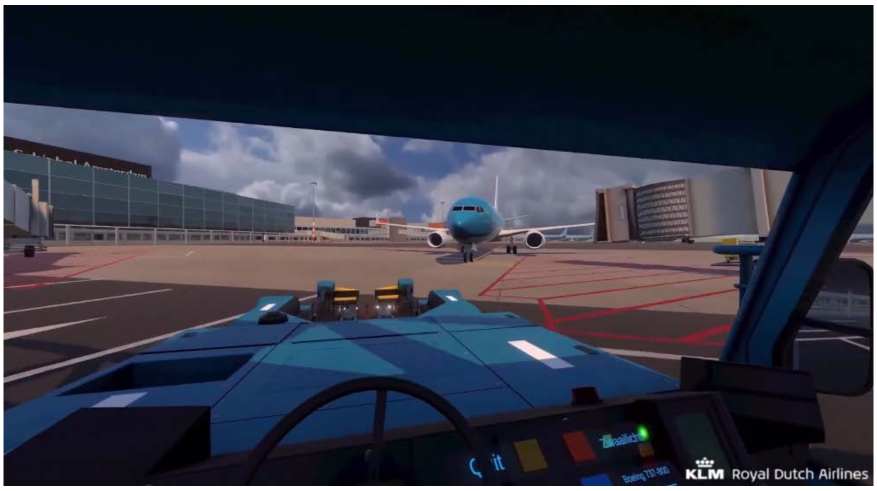


Figure 2.4. AIRFRANCE KLM-XR Center of Excellence. (2021). Pushback training. https://www.youtube.com/watch?v=C6EiRQjvZjo [16]

KLM Royal Dutch Airlines has been using Virtual Reality to train cabin and ground crews using different applications. The company has been striving towards obtaining EASA certification in order to replace some components of the current training processes [17].

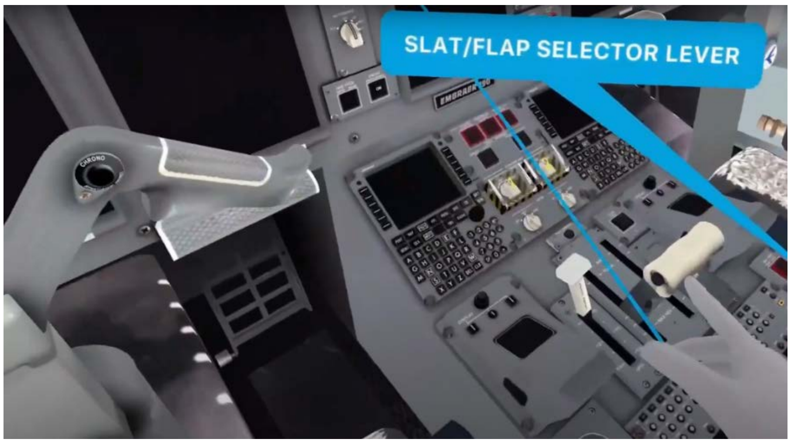


Figure 2.5. AIRFRANCE KLM-XR Center of Excellence. (2021). Pilot training. https://www.youtube.com/watch?v=C6EiRQjvZjo [16]

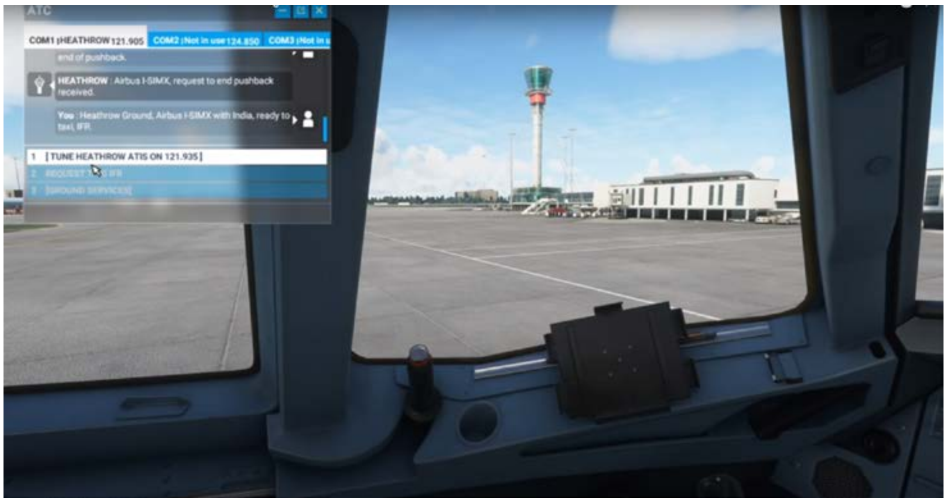
In the demo reels that have been posted online, KLM presents experiences for pushback training, pilot training, and emergency trainings for cabin crew.

The main purpose of the VR experiences for pilots is to familiarize the trainee with the cockpit and controls so that their use of the simulator time is more effective.

2.3.3. Microsoft Flight Simulator

Microsoft has been releasing entries in the Flight Simulator series since 1982. The latest was released in August 2020 and it is the most advanced amateur flight simulator today considering all technological features. The game applies cloud computing (Microsoft Azure) to generate the topography of the entire Earth using data from Bing Maps.

Microsoft Flight Simulator offers extremely detailed recreations of airports and aircraft. It has several hand-crafted airports where flight procedures have been scripted. In these airports, the user piloting the aircraft can interact with a virtual air traffic controller to receive instructions on common procedures like approach, landing, taxiing and takeoff.



end of pushback.

HEATHROW : Airbus I-SIMX, request to end pushback received.

You : Heathrow Ground, Airbus I-SIMX with India, ready to taxi, IFR.

Figure 2.6. 320 Sim Pilot. (2020). Communications display in Microsoft Flight Simulator. https://www.youtube.com/watch?v=fLQgAN79YqE [18]

This feature was particularly relevant to this project because understanding and following instructions is essential for any procedure at any commercial airport.

Microsoft Flight Simulator also offers VR support, dynamic time of day, weather systems and it can be synchronized with real-world air traffic. All in all, it is one of the most technically advanced games currently available.

CHAPTER 3. EXPERIENCE DESIGN

In this chapter, the requirements that should be kept in mind when developing the tool are established. Based on the objectives, requirements and aspects of an actual taxi procedure using the TaxiBot, several decisions must be taken to attain a polished experience while trying to keep the development process convenient as well.

3.1. Requirements and Optional Systems

The series of functional requirements that have been foreseen for the simulator are the following.

- Immersive environment recreation. To increase immersion, the main elements of the experience, like the plane or planes and the TaxiBot will need high quality assets. The airport will need correct markings on the taxiways and runways throughout the procedure.

- The physics of the simulated vehicles should resemble the behavior of the real vehicles they represent.

- The experience must be able to track the steps and route taken by the user in order to evaluate possible errors they might have committed. Depending on the severity of the infraction, the simulation could come to an unsuccessful end. Performance analysis should be simple and automatic.

- An integrated solution to allow the user to check the relevant aerodrome charts. Since just the taxi phase is contemplated, at least the Ground Movement Chart must be included.

- A networking solution that allows another user to evaluate the pilot in real time should be implemented.

A few optional systems that can be implemented to enhance the experience are listed below.

- Control over the time of day and weather conditions during the experience.

- Support for different peripherals to enable more accurate control. For example, a joystick or a steering wheel provide analog controls that are not possible simply using a keyboard.

3.2. Target Audience

The amount of previous knowledge required to take part in the experience and find it useful will determine the overall level of difficulty. In order to carry out a taxi procedure, the pilot and co-pilot need to follow a very long sequence of steps. During the pushback phase, the aircraft will probably be pushed away from the airport gate by a pushback tractor from the parking position to a taxi position. On completion of pushback, the crew must ensure that all the systems are working correctly. Starting the APU, starting the engine or engines, turning the taxi lights on and disabling parking brakes are all things that take place right before taxiing and they require the pilot to use the intricate control panels within the cockpit. The pilot also needs to be able to communicate with the ground crew, first officer, and air traffic controller. An exceedingly important aspect is that the pilot needs to be able to move in an airport environment using the regular provided charts and instructions to understand the required path.

The assistance that the experience can provide to a beginner will determine the entry level of knowledge of the user. The objectives, setting, and mission should be explained in a short tutorial prior to the procedure. The experience will be a piece of interactive audiovisual media, so a written document explaining the use of signs and markings in an airport, specifications or separation criteria could be overwhelming. In order to help maintain user engagement, a different solution should be studied to aid beginners to find their route and undergo the experience.

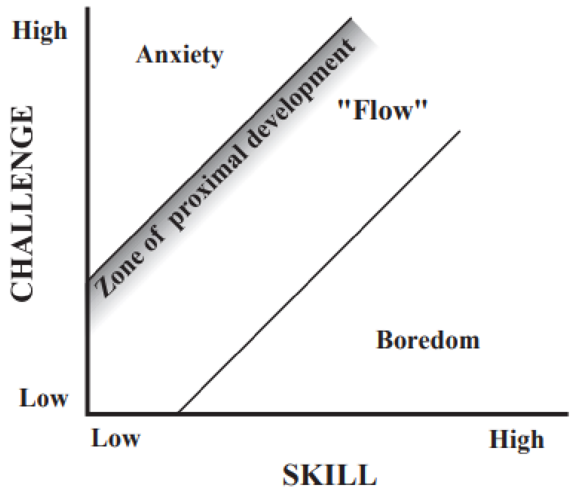


Figure 3.1. Kiili, K (2004). Three channel model of flow [19].

The proposed three channel model of flow (Kiili, 2004, p. 16) [19] in Figure 3.1 asserts that if a challenge is lower than the skill level of the user, the player might feel bored. The player might grow confused or anxious if the challenge is greater than expected, on the other hand. The author proposes a zone of proximal development, which is the ideal flow state where skill level increases with the presented challenge. This zone of proximal development might be extended if guidance or help from other players is provided. This statement served to define one of the main features of the designed application.

3.3. Specifications

It is important to distinguish what is already established from what is still undetermined because a decision needs to be made with the purpose of approaching the initial objectives of the project. In this section, a review of the existing information is presented.

3.3.1. TaxiBot Specifications

In order to simulate the dynamics of the involved vehicles, a good deal of data about them is essential. The very first specifications that are required to re-create the TaxiBot are measurements, weight, and power. Frankfurt Airport has arranged a project to reduce ground traffic emissions. As one of the measures, the TaxiBot was integrated as part of the deployed fleet. Some key points of information can be found on the site of the E-PORT AN project, which manages Electromobility at Frankfurt Airport [20].

These specifications apply to the Narrow Body model:

- Net weight: 27 t
- Tractive power: 8 t
- 2 generators (260 kW), 2 Scania DC9 diesel (2 x 294 kW / 2 x 394 PS).
- Driving speed (max. load): 42 km/h
- Taxiing at 23 knots, same as current airplane taxi speed.
- Protection of the nose landing gear (NLG) from exceeding maximum allowed fatigue load at all times.
- No change in Nose Landing Gear life time.

Following unsuccessful attempts to contact Israel Aerospace Industries, the main company responsible for TaxiBot manufacturing, some further information will most likely be required. When a new parameter is needed, the specifications of a similar pushback/tow tug (in size and power) will be considered [21].

Another source of data would be videos that can be found online featuring a TaxiBot operation partially or completely. These videos of the real vehicle in motion can provide some validation since a comparison between the represented dynamics and the real dynamics is more empirical in nature and valid nevertheless.

3.3.2. Specifications of the Featured Aircraft

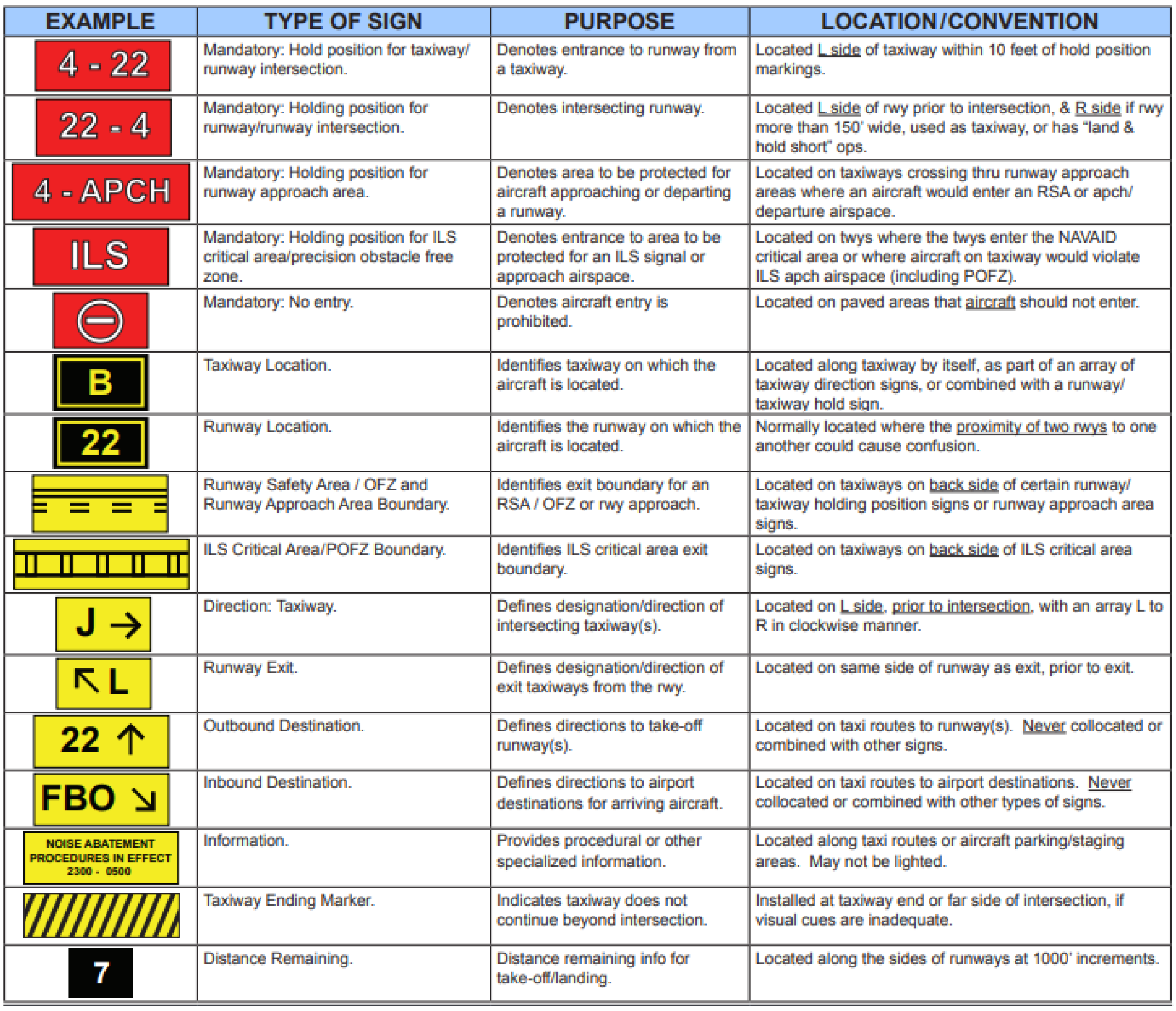
The model of aircraft that is selected for the experience in development is not very important, as long as its size and weight are within the upper and lower limits for the TaxiBot Narrow Body.

In any case, some important aspects about the aircraft are measurements, weight, turning radius on ground, and if the airplane is able to operate on the designated airport according to its category. The interior of the aircraft will have to be recreated faithfully since the user will play the role of pilot in the experience.

3.3.3. Airport Specifications

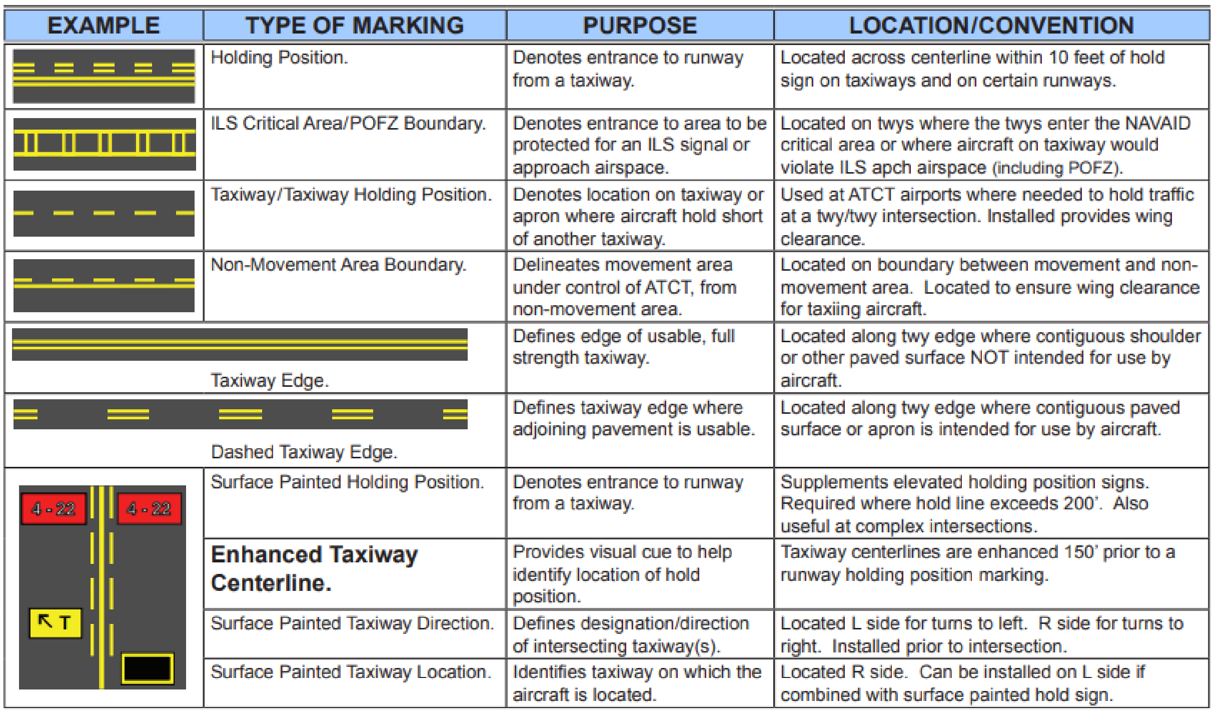
Being aware of the position of your aircraft and understanding the route you must follow is a crucial aspect of ground movements. The environment recreation must therefore include the relevant surface markings and signs.

Principles of surface markings and signs placed beside the taxiways and runways of a licensed aerodrome can be found in the ICAO International Standards and Recommended Practices (SARPS) [22] or the FAA Standards for Airport Markings documents [23]. A quick reference guide of the latter document is presented below.



EXAMPLE	TYPE OF SIGN	PURPOSE	LOCATION/CONVENTION
4 - 22	Mandatory: Hold position for taxiway/ runway intersection.	Denotes entrance to runway from a taxiway.	Located L side of taxiway within 10 feet of hold position markings.
22 - 4	Mandatory: Holding position for runway/runway intersection.	Denotes intersecting runway.	Located L side of rwy prior to intersection, & R side if rwy more than 150' wide, used as taxiway, or has "land & hold short" ops.
4 - APCH	Mandatory: Holding position for runway approach area.	Denotes area to be protected for aircraft approaching or departing a runway.	Located on taxiways crossing thru runway approach areas where an aircraft would enter an RSA or apch/ departure airspace.
ILS	Mandatory: Holding position for ILS critical area/precision obstacle free zone.	Denotes entrance to area to be protected for an ILS signal or approach airspace.	Located on twys where the twys enter the NAVAID critical area or where aircraft on taxiway would violate ILS apch airspace (including POFZ).
⊖	Mandatory: No entry.	Denotes aircraft entry is prohibited.	Located on paved areas that aircraft should not enter.
B	Taxiway Location.	Identifies taxiway on which the aircraft is located.	Located along taxiway by itself, as part of an array of taxiway direction signs, or combined with a runway/ taxiway hold sign.
22	Runway Location.	Identifies the runway on which the aircraft is located.	Normally located where the proximity of two rwys to one another could cause confusion.
	Runway Safety Area / OFZ and Runway Approach Area Boundary.	Identifies exit boundary for an RSA / OFZ or rwy approach.	Located on taxiways on back side of certain runway/ taxiway holding position signs or runway approach area signs.
	ILS Critical Area/POFZ Boundary.	Identifies ILS critical area exit boundary.	Located on taxiways on back side of ILS critical area signs.
J →	Direction: Taxiway.	Defines designation/direction of intersecting taxiway(s).	Located on L side, prior to intersection, with an array L to R in clockwise manner.
↖ L	Runway Exit.	Defines designation/direction of exit taxiways from the rwy.	Located on same side of runway as exit, prior to exit.
22 ↑	Outbound Destination.	Defines directions to take-off runway(s).	Located on taxi routes to runway(s). Never collocated or combined with other signs.
FBO ↘	Inbound Destination.	Defines directions to airport destinations for arriving aircraft.	Located on taxi routes to airport destinations. Never collocated or combined with other types of signs.
NOISE ABATEMENT PROCEDURES IN EFFECT 2300 - 0500	Information.	Provides procedural or other specialized information.	Located along taxi routes or aircraft parking/staging areas. May not be lighted.
	Taxiway Ending Marker.	Indicates taxiway does not continue beyond intersection.	Installed at taxiway end or far side of intersection, if visual cues are inadequate.
7	Distance Remaining.	Distance remaining info for take-off/landing.	Located along the sides of runways at 1000' increments.

Figure 3.2. FAA. (2009). Airport Sign and Marking Quick Reference Guide Part 1. https://www.faa.gov/airports/runway_safety/publications/media/QuickReferenceGuideProof8.pdf [23]



EXAMPLE	TYPE OF MARKING	PURPOSE	LOCATION/CONVENTION
	Holding Position.	Denotes entrance to runway from a taxiway.	Located across centerline within 10 feet of hold sign on taxiways and on certain runways.
	ILS Critical Area/POFZ Boundary.	Denotes entrance to area to be protected for an ILS signal or approach airspace.	Located on twys where the twys enter the NAVAID critical area or where aircraft on taxiway would violate ILS apch airspace (including POFZ).
	Taxiway/Taxiway Holding Position.	Denotes location on taxiway or apron where aircraft hold short of another taxiway.	Used at ATCT airports where needed to hold traffic at a twy/twy intersection. Installed provides wing clearance.
	Non-Movement Area Boundary.	Delineates movement area under control of ATCT, from non-movement area.	Located on boundary between movement and non-movement area. Located to ensure wing clearance for taxiing aircraft.
	Taxiway Edge.	Defines edge of usable, full strength taxiway.	Located along twy edge where contiguous shoulder or other paved surface NOT intended for use by aircraft.
	Dashed Taxiway Edge.	Defines taxiway edge where adjoining pavement is usable.	Located along twy edge where contiguous paved surface or apron is intended for use by aircraft.
4 - 22 4 - 22	Surface Painted Holding Position.	Denotes entrance to runway from a taxiway.	Supplements elevated holding position signs. Required where hold line exceeds 200'. Also useful at complex intersections.
	Enhanced Taxiway Centerline.	Provides visual cue to help identify location of hold position.	Taxiway centerlines are enhanced 150' prior to a runway holding position marking.
↖ T	Surface Painted Taxiway Direction.	Defines designation/direction of intersecting taxiway(s).	Located L side for turns to left. R side for turns to right. Installed prior to intersection.
	Surface Painted Taxiway Location.	Identifies taxiway on which the aircraft is located.	Located R side. Can be installed on L side if combined with surface painted hold sign.

Figure 3.3. FAA. (2009). Airport Sign and Marking Quick Reference Guide Part 2. https://www.faa.gov/airports/runway_safety/publications/media/QuickReferenceGuideProof8.pdf [23]

Airport charts can be procured from the corresponding AIP (Aeronautical Information Publication) in order to understand where the indicated markings and signs should go according to the pertaining documentation.

Fortunately, consumer satellite imagery has reached a sufficient resolution that many airports in the world can have their surface markings recreated exactly. As an example, the image below showing a section of Barcelona–El Prat Airport was captured using Google Earth.

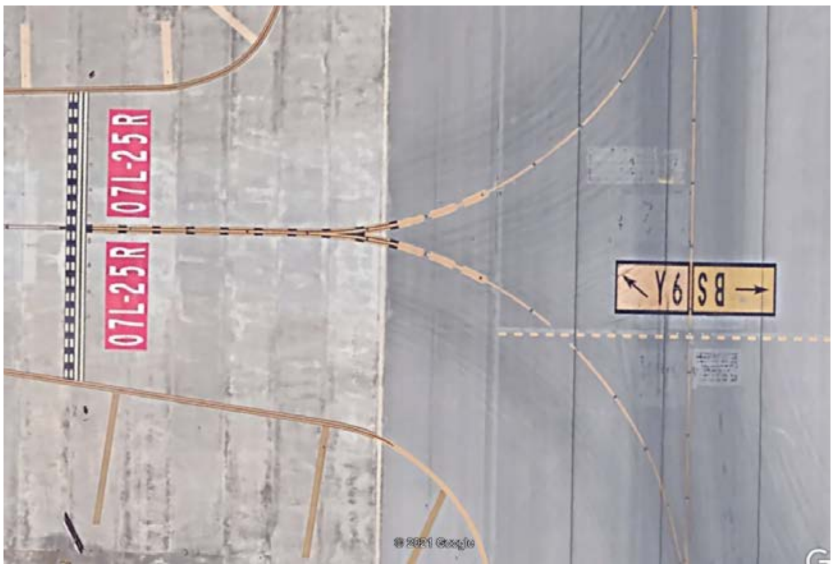


Figure 3.4. Example of taxiway markings in Barcelona-El Prat Airport. (Image captured using Google Earth).

The required markings will be traced to create textures that can be placed on the virtual environment with their actual scale and position. The real airports might not follow the documentation exactly as most of it is recommended and not enforced. In addition, marks can be deleted or obscured by tire tracks, or sections of pavement can be replaced. In my opinion, the environment should be closer to the actual airport to increase the training value of the experience. In any case, the intention will be to include the markings that are most relevant to the featured procedure if time does not allow every marking to be reproduced.

To reproduce other signs, lights, and their accurate positions, pictures, charts or other available resources could be used.

3.4 Decisions and Criteria

3.4.1. Location Selection

To increase its value as a learning tool, the experience should not be a perfectly straightforward example of a taxi procedure. To add some difficulty, the airport where the experience will take place should enable obstacles or events that can come into play and disturb the operation. Based solely on this assessment, an airport with a higher complexity is preferable. However, unless a 3D model of such an airport can be acquired easily, a higher complexity airport will require a longer period of time spent modeling.

Initially, Barcelona-El Prat Airport was considered because its size, runways and taxiways can serve as a remarkable environment for an intricate taxi procedure. However, a 3D model of this airport could not be procured for free or within pricing expectations, so the environment would have to be created from the start.

Therefore, Valencia Airport (also known as Manises Airport) was selected because it is a smaller, single runway airport and the estimated time to create the 3D model and textures is much shorter. Valencia Airport was the eighth busiest Spanish airport in 2019 and the third considering only single runway aerodromes. The final traffic number came to 8.539.403 passengers in 2019 [24] (this year was chosen because air travel was heavily affected by the COVID-19 pandemic from 2020 onwards).

The taxiways and gates could be considered to be more complex than those of an average single runway airport since it used to also serve the shorter runway 04/22. Only runway 12/30 is currently in use so the experience will feature a flight departing from this runway. The runway is 3215 meters long and 45 meters wide. The taxiway N (parallel to the runway) is 23 meters wide and the paved area is 44 meters wide (measurements taken from Google Earth). Both measurements for the runway and taxiway correspond to code letter E. The taxiway to runway distance is 200 meters. The airport can then operate with heavy aircraft like the Boeing 747, B777, B787, Airbus A340 and others.

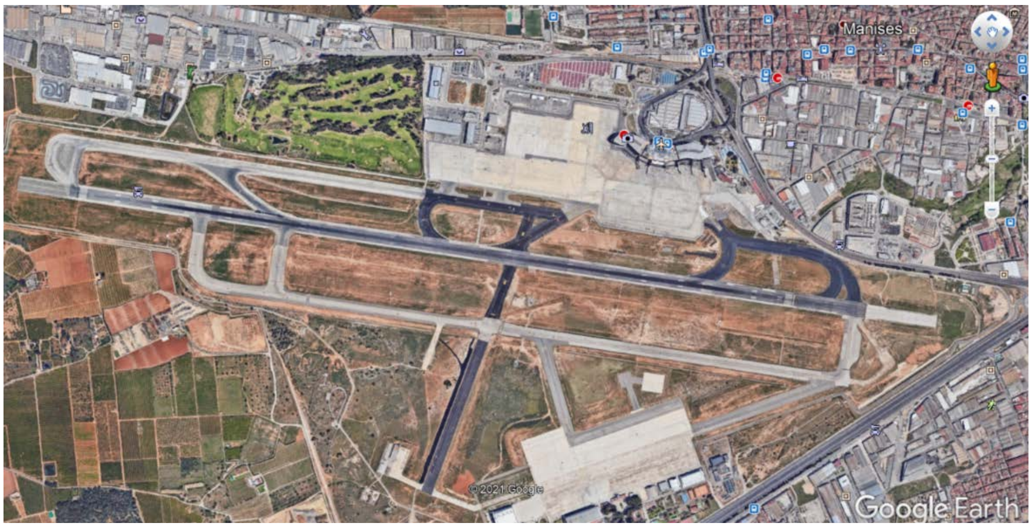


Figure 3.5. Layout of Valencia Airport. Captured using Google Earth.

3.4.2. Taxi Procedure

The taxi procedure that the pilot performs in the experience will preferably be complex and feature unlikely but possible events. The educational value of the experience will depend on several design decisions and an effort to identify them is contained in this chapter.

The airport has already been selected because of its size and complexity. Between the two possible directions of runway 12/30, 12 was selected because the number of possible conflicts is larger when compared to the other configuration. A section of the Aerodrome Ground Movement Chart shown below illustrates one of the possible hurdles of the experience. Following taxiway H4 instead of turning right on N2, a pilot might accidentally cause a dangerous runway incursion. This is called a hot spot, a location with a history or potential risk of collision or incursion.

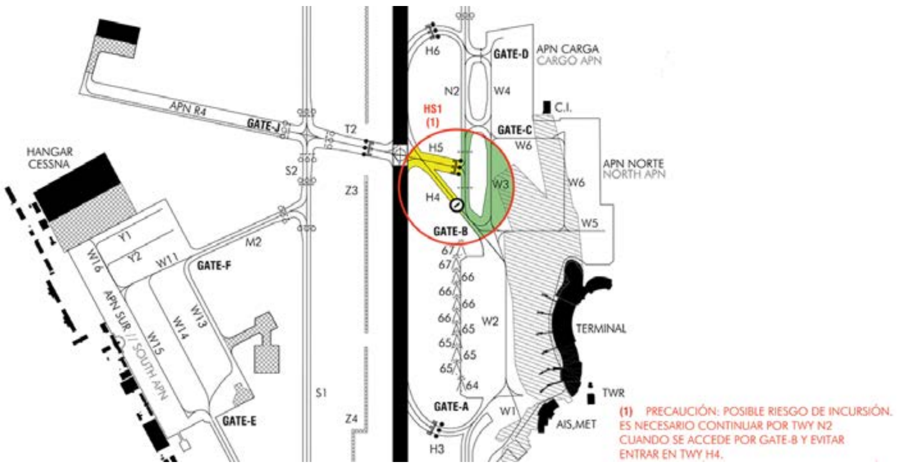


Figure 3.6. AIP España. Detail of a possible conflict zone in the Ground Movement Chart. https://aip.enaire.es/AIP/contenido_AIP/AD/AD2/LEVC/LE_AD_2_LEVC_GMC_1_en.pdf [25]

An indicative duration of 5 to 10 minutes was selected in order to keep the user engaged throughout the experience. There are no specific limitations on the subject of taxi speeds. Instead, the aircraft manufacturer will recommend speeds that allow for safe stopping distances, turns, and avoiding brake degradation. For example, the Boeing 737 Flight Crew Training Manual [26] states that a normal taxi speed is approximately 20 knots, with a maximum of 30 knots in straight sections and around 10 knots in high angle turns.

The TaxiBot specifications are quite clear in this regard. The maximum speed of the vehicle under load is set to 23 knots. The distance from the terminal to the holding position H8 is approximately 2500 meters. Considering a speed of 20 knots in straight areas of the route and 10 knots on turns, the time an aircraft would take in an uninterrupted taxi procedure to move this distance is almost exactly 5 minutes.

Different events that a pilot might experience and add difficulty to the procedure could be implemented to add learning potential. These include low visibility, the presence of obstacles on the taxiway or runway, coming across other aircraft or vehicles and getting instructions to change the initial planned route for the procedure.

The instructions to perform the procedure will use correct aviation English terminology and vocabulary when possible. While this aspect makes the experience more realistic, it brings the problem of reducing the target audience since some previous knowledge on aviation communication is expected of the user. If the airport zones, taxiways, runway and obstacles are all modeled correctly and the program can place the TaxiBot and aircraft that the user is controlling within those zones, the entire route from starting position to the holding position can be changed rather quickly. For this implementation, the base selected taxi procedure is the following:

“[Callsign], taxi to and hold short of RWY 12 via Gate C N H8. Contact tower on 118.55 when ready.”

The user would then need to understand the instructed route from the terminal gate to the holding position. The pushback procedure is performed by the TaxiBot operator with help of the ground crew since the pilot cannot see behind the aircraft. To be able to simulate this part of the procedure correctly, the same user or another would need to play the role of TaxiBot operator initially and they should have a more complete view instead of the view from the cockpit. The pilot would take over from then on and follow the appropriate centerlines to piece together the route as is shown below.

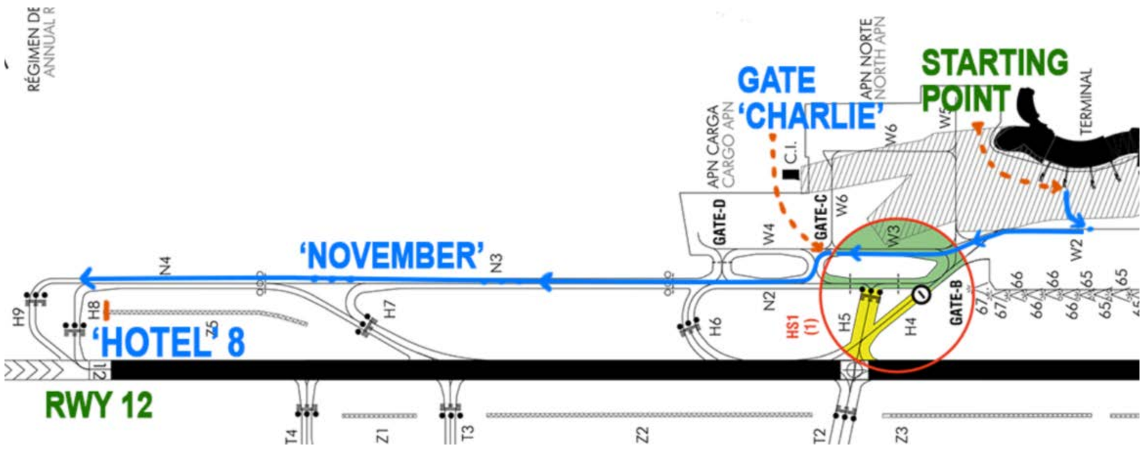


Figure 3.7. Route of the designed operation.

A problem that presents itself as a result of the use of the TaxiBot is the fact that taxiways in most airports are not prepared to have a vehicle that needs to return to the apron. A pushback tractor disconnects from the aircraft while still on the apron and it can move safely back to a hangar or another aircraft that requires pushback. A TaxiBot however, should be able to disconnect from the plane on the holding position and return safely to the apron. The taxiway will most likely be in use already for a different operation, so an alternative needs to be in place. An aeronautical study to find an optimal solution needs to be made in any airport that is considering a TaxiBot implementation. But, as it escapes the scope of this project, I will simply add a service road parallel to the taxiway. According to ICAO SARPs Table 3-1 [22], the correct minimum distance from the existing taxiway centerline for object separation is 43.5 meters to maintain the code letter E.

3.4.3. Challenging the User

As mentioned in Section 3.2, the challenge of the experience should be proportional to the skill level of the user, otherwise they would get uninterested (not enough challenge) or lost (too much challenge). Additionally, including unlikely, but possible events only rises the number of opportunities where the user can learn a new aspect of taxi procedures. Different aircraft or ground vehicles can be encountered throughout the apron and taxiway. In order to maintain separation, a flight controller will need to issue new instructions and the pilots should be able to comply. Additionally, visibility could be reduced or strong winds could affect control of the aircraft. Many scenarios that are described in documents by the ICAO could be recreated. For example, Annex 2 to the Convention on International Civil Aviation (Rules of the Air) [27] contains several rules and information that a pilot should have internalized, like right-of-way and overtaking, even in the absence of traffic controller instructions.

Threats are events that can increase operational complexity. Anticipated threats, like weather conditions or regular traffic can be expected. Unexpected threats cannot be foreseen by the crew, such as a sudden mechanical failure, unauthorized traffic or obstacles, or poor runway conditions. The crew must apply their skills obtained in training in all of these cases and avoid committing an error.

These conditions all make for interesting extensions of the experience, but whether or not they are implemented will be determined by the development schedule. The most important aspects will be the ones that are essential to fulfill the main objectives.

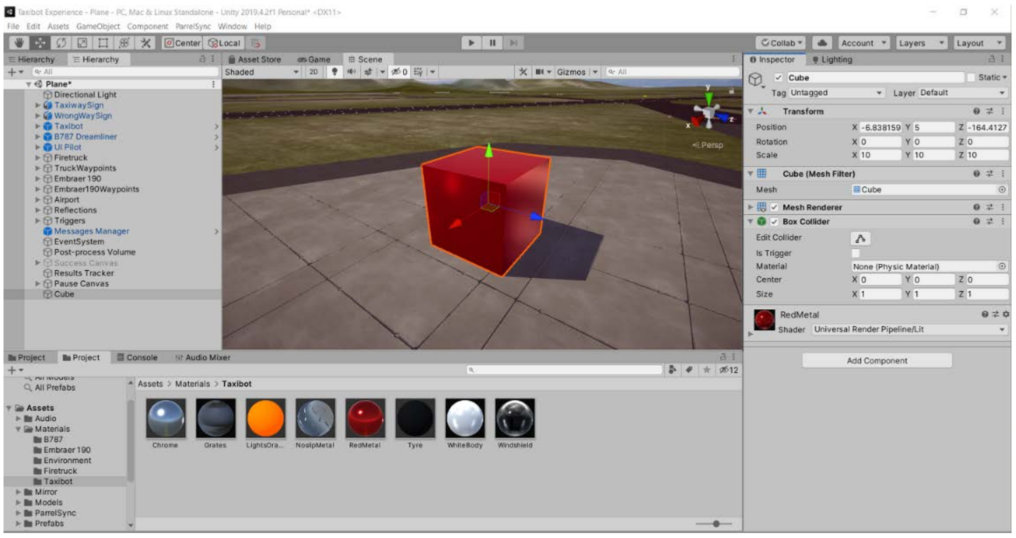
CHAPTER 4. UNITY IMPLEMENTATION

4.1. Relevant Technological Tools

Development of an experience in any engine can be a laborious project and it depends on how much work can be saved. There are several software available that can provide the services that were required to create this specific implementation. In this case, I used the ones I was most comfortable with to be more efficient.

4.1.1. Unity

Unity is one of the most popular game engines in the market. It can be used to develop applications in 2D or 3D for a large number of platforms. It is often considered to be flexible and easy to get started. Due to its versatility and because it counts with a free version, countless video games, serious games, and applications have been released.



Cube ✓ Static
Tag Untagged Layer Default
Transform
Position X -6.838159 Y 5 Z -164.4127
Rotation X 0 Y 0 Z 0
Scale X 10 Y 10 Z 10
Cube (Mesh Filter)
Mesh Cube
✓ Mesh Renderer
✓ Box Collider
Edit Collider
Is Trigger
Material None (Physic Material)
Center X 0 Y 0 Z 0
Size X 1 Y 1 Z 1
RedMetal
Shader Universal Render Pipeline/Lit

Figure 4.1. View of the Editor in Unity.

The Editor provides an environment to make use of all the tools offered by Unity. The project that contains the required pieces of a game or experience is created within this Editor.

An extremely useful feature of Unity is its Asset Store. Users can share for free or sell their assets and packages. Because Unity is exceedingly popular, it is very common to find convenient elements on the asset store, documentation, or relevant tutorials online.

A very short description of how Unity works is provided below.

4.1.1.1. Assets

Assets are resources that can be imported into Unity in order to be used in the experience. External files like 3D models, images for textures or User Interface (UI), audio files, and scripts can be imported if they are in supported formats.

4.1.1.2. GameObjects

GameObject is the fundamental class of all entities that are used in Unity. Any instance can contain several components, which add specific functionalities to that object. For example, if a developer intends to implement a ball object, it could use these components and more:

- A transform component is always attached to an object. It contains information about the position, rotation and scale of said object.
- A spherical 3D model in the Mesh Renderer. This component determines the geometry that is rendered and is therefore visible.
- A sphere collider to check for collision with other objects. This could be necessary if gravity affects the ball and the developer does not want the ball falling through the floor object (which would also require a collider).
- Rigidbody is a component that puts the motion of the object under the physics engine used by Unity. Its mass, friction and other parameters can be changed using this component. The behavior of the ball can be controlled through the application of forces
- A script is a piece of code (current versions of Unity support C# natively) that can control objects, trigger events, read user input. If the developer would like to kick the ball when the space key is pressed, it would be through a script. If the key is pressed, set the velocity of the ball at that initial moment to a given three-dimensional vector. Conveniently, one object can have more than one script attached to it.

4.1.1.2. Scenes

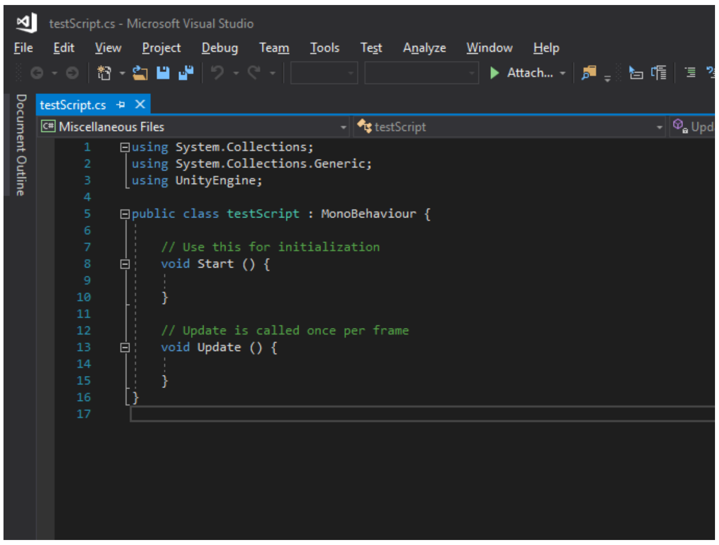
Scenes are assets that can define a virtual environment. A project could have multiple scenes and each scene could contain a map, an object controlled by the user, obstacles, lights, cameras, UI panels and more. The normal approach to develop an experience is to create a new scene, make and import the required assets, create the GameObjects within that scene and configure their components.

Afterwards, those GameObjects or groups of GameObjects (with their components and properties) can be saved as *prefabs* in order to save them and make the job of implementing or instancing them on other scenes much easier.

4.1.1.3. Runtime

The Unity editor can compile the application in order to test it almost instantly (depending on hardware). The behavior of all objects when they are loaded, when they are enabled, on every time step of the physics simulation and on every frame of the running application can be determined by scripts. Several built-in functions and parameters of the Unity libraries are indispensable to know for any application. In the normal approach, scripts contain classes that inherit from MonoBehaviour. This allows the scripts to be placed in GameObjects, change variables from the inspector and access Unity events like Start(), Awake(), FixedUpdate(), Update() and other functions. These functions are called when the object that contains them is loaded, when it is initialized, every 20ms (by default) or every frame of the engine (as fast as possible) respectively.

4.1.2. Visual Studio



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

public class testScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Figure 4.2. Empty MonoBehaviour script in Visual Studio 2017.

An Integrated Development Environment is an application software that provides tools and services that assist in software development. Unity currently supports Visual Studio, VSCode, JetBrains Rider, and a few other IDEs. By default, a Unity installation will also install Visual Studio. These IDEs enable script edition, access to Unity libraries documentation, and debugging tools. Due to the nature of GameObjects with countless possibilities of components, the autocomplete feature and access to documentation on the use of built-in parameters is nearly indispensable.

4.1.3. C# in Unity

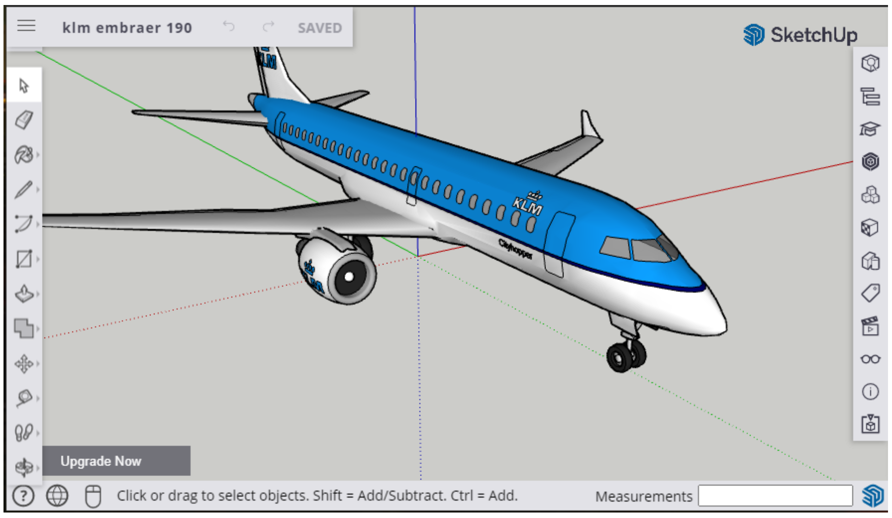
Unity uses C#. It is usually implemented as an Object-oriented scripting language although Unity has also released a different architecture that allows for different functionalities and increased parallelization called DOTS. Some principles of Object Oriented Programming (OOP) should be kept in mind but the usual Unity approach is more straight-forward and it revolves around GameObjects and components. Both GameObjects and components are objects, but the implementation and ease with which a developer can build an application is simpler and achieves better performance when compared to an application built using “pure” OOP. In any case, basic proficiency in coding is required as almost all scripts will require:

- Variables, which contain values like integers, strings of characters, Booleans, and references to other objects.
- Functions, which are pieces of code that can read and manipulate variables.
- Classes that create the structure and contain definitions of variables and functions.

Unity currently supports C# 8.0 and it uses Roslyn, an open-source C# compiler to test or build applications that run in almost any platform like Windows, Linux, OS X or Android.

4.1.4. SketchUp

SketchUp is a 3D modeling computer program that is based on drawing two-dimensional faces that can be manipulated to create 3D geometry. It is one of the most user friendly options available at the moment and it includes good tutorials to help newcomers. It offers a free version as well as a paid version and it is available as a web-based application on any browser.



Click or drag to select objects. Shift = Add/Subtract. Ctrl = Add.

Figure 4.3. SketchUp – free web based version.

Similar to Unity, an advantage of SketchUp is its community. The program makes use of the 3D Warehouse, a place where users can sell their models or offer them completely for free. Thanks to its popularity, countless assets like real buildings, vehicles, sections of world terrain can be found for free. There are other alternatives online to buy 3D models or get them for free, but there are usually far fewer free models.

The professional version of SketchUp lets the user export the model to .fbx, which is the preferred format for Unity. Otherwise, there are free online converters that can provide this service. Models found in the 3D Warehouse will always be in the same .skp format, so they can be modified in SketchUp to solve issues, add or remove elements.

Because of its simplicity, SketchUp itself is not very powerful and it does not support several features that other experiences might require. For example, it cannot create animations (or rigging or posing) and it does not perform any physics simulation.

4.1.4. Adobe Photoshop

Adobe Photoshop is the most recognized image editor available. Since 2013, it changed its licensing scheme to software as a service. A user can subscribe to the Adobe Creative Cloud for a monthly fee instead of a large single payment purchase.

Numerous images will be created for the experience and they will be placed as textures or UI elements. Most image editing software will likely be able to serve this purpose.

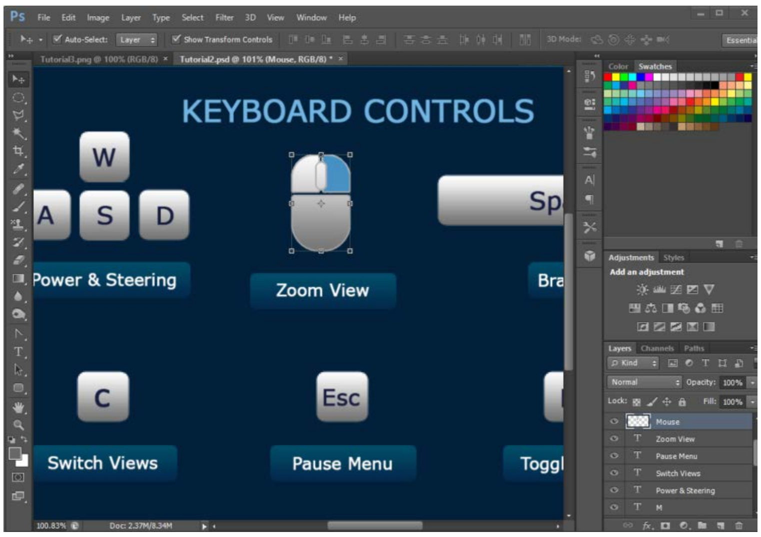


Figure 4.4. Use of Photoshop to create User Interface elements.

4.2. Personal Background

This section is more personal as it delves into the reasoning behind these choices and it relates to my previous knowledge of all the development aspects. If a different developer decides to create a similar experience, the software selection, time investment, and several other factors will most likely be different.

I started making videogames as a hobby at a young age but I have never taken part in any professional project. While I have used other engines before, most of my time was spent learning Unity. I consider myself to be competent in its use but far from an expert.

I have experience creating architectural renderings as it was my first job. As a result, I am also proficient in 3D modeling and texture work. I have selected SketchUp because it will be the fastest alternative when it comes to recreate a floor plan without small details (such as an airport). The countless models found on the 3D Warehouse will save a lot of time. All required models will be static, so there is no need to use another program that supports animation, cloth physics or soft-body dynamics.

There are many roles involved in the creation of simulation software, games, or other experiences. The number of skills that are required and the time it takes to learn them and keep them up to date is the main limiting factor on solo development. In the process that follows, I will try to document the effort each major task takes and how different systems could be implemented at a different stage, if I had more time or teammates.

4.3. Assets

4.3.1. Modeling and Adapting

A 3D model of the TaxiBot could not be procured. Fortunately, the Narrow Body model is very similar to other towbar-less tractors, so a model of the Komatsu WZ4000 was used instead. Its size and proportions were modified to better resemble the TaxiBot.

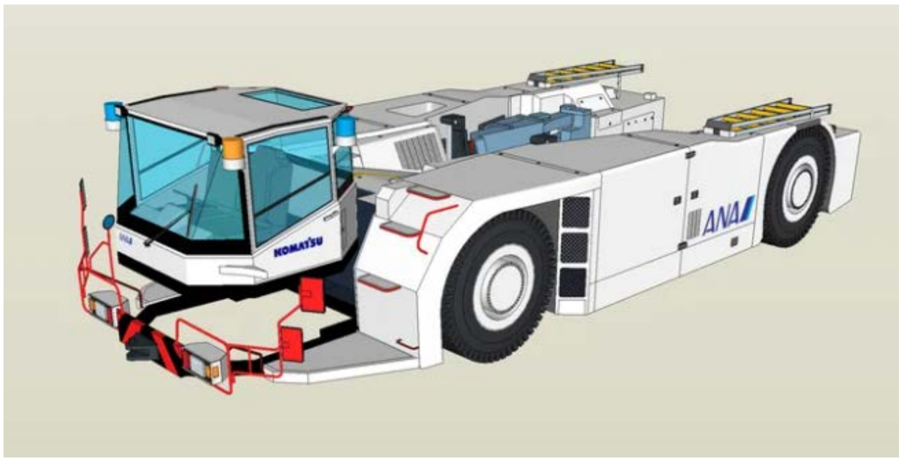


Figure 4.5. Komatsu WZ4000 model to be used as the TaxiBot [28]. Found on the SketchUp 3D Warehouse.

In most modeling software, a face is defined by its vertices and a normal direction. Using the directions, an “inside” face and an “outside” face can be defined. This allows any rendering program to save performance as it only needs to draw the face facing outwards. Models in the .fbx format present this behavior in Unity. In SketchUp however, faces are often generated incorrectly and it is not problematic within the program itself as it can display both sides correctly textured.

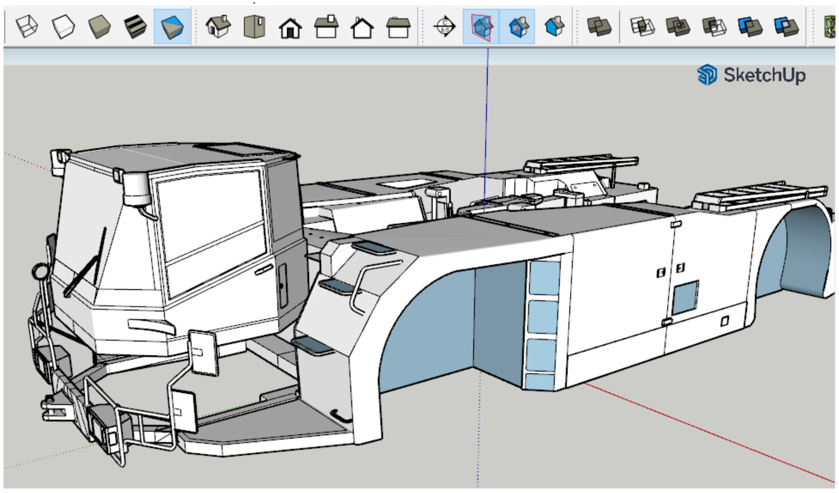


Figure 4.6. Normal direction issues in the downloaded TaxiBot model.

A look at the model without the applied materials can reveal the issue. The blue faces will not render in Unity and the object will not be complete. Many models available in the 3D warehouse need to be corrected in order to import them into Unity. In this TaxiBot substitute, the interior was also added since the original model was missing faces and a camera could not be placed inside the cabin. The wheels were removed because they need to move. The easier solution will be to keep them as separate GameObjects in Unity, so it is simpler to keep the models separate as well.

As for the aircraft, there are countless options available on the Warehouse. For example, there are over one thousand free results from the query “Airbus A320”. The Boeing 787-9 Dreamliner was selected simply because a good model was available and it can operate on Valencia Airport. On retrospect, one of the aircraft models that have been certified for TaxiBot use would have been a better choice. These include the Airbus A320 family and the Boeing 737.

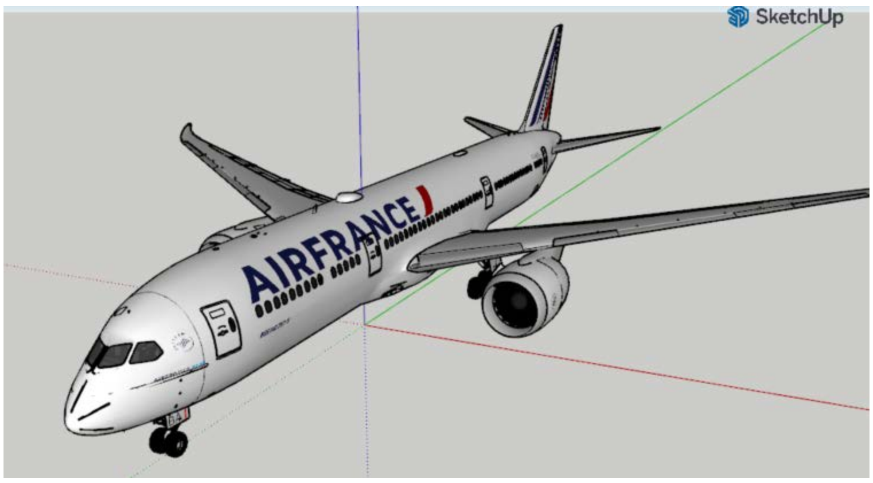


Figure 4.7. Air France Boeing 787-9 3d model [29].

Since the experience will undoubtedly place a camera that simulates the point of view of the pilot, the interior of the cockpit will also be needed. A model of the cockpit that includes the flight display and other instruments was fortunately also available from the 3D Warehouse [30].

The model had to be modified slightly because its scale was not entirely precise. The inner walls, windows, floor and seats were modeled to fit both the plane and instruments 3D models. A reference 360º panoramic image of the cockpit was used when reference was required [31].

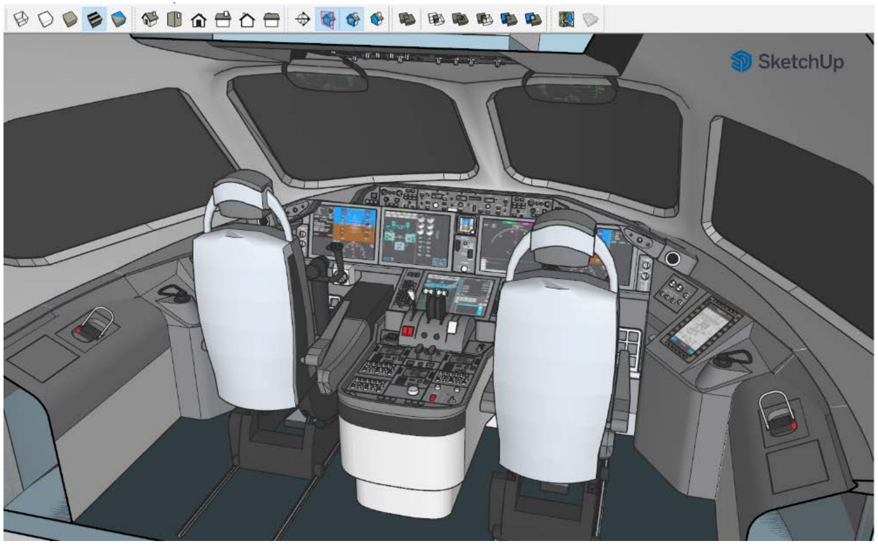


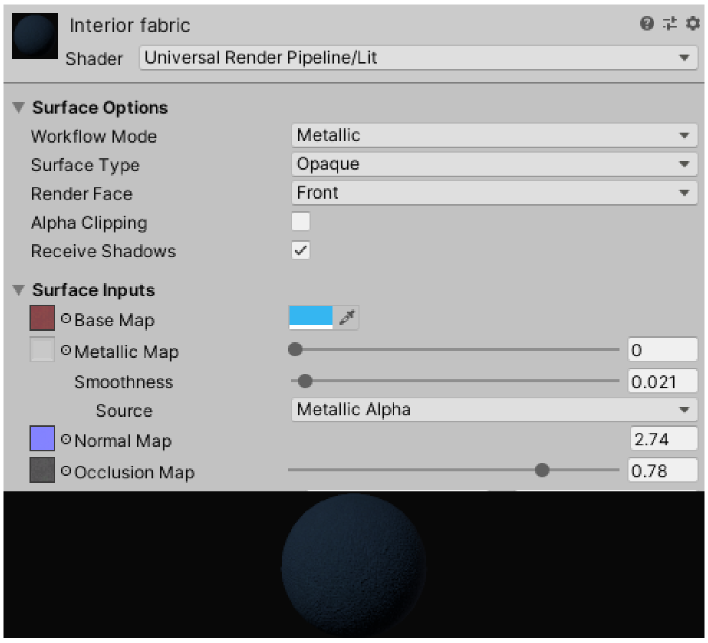
Figure 4.8. View of the cockpit in the merged 3D model.

Another modification that was implemented was separating the model in groups. The polygons that make up the wings, engines, fuselage, cockpit, and different landing gears all belong to separate groups. This could allow Unity to move them independently. A regular technique in simulation software or videogames is to replace a “healthy” piece of a vehicle with a damaged version of that piece when it detects a collision. The main cockpit controls are also separate groups, so if desired, an animation that relates the user input to the virtual yoke, tiller or pedals can be implemented.

Many small elements in the models were removed because they would not be visible and they were only increasing the file size and rendering load. The original plane model was 16MB and the cockpit model was 7MB. The final model that merges them together where I added interior details is 11MB in size. In total, this 53% size reduction was achieved removing unnecessary textures, details on the fuselage like pitot tubes, and remaking the seats and smaller objects to reduce polygon count.

4.3.2. Importing Models

Textures are images that are added to a 3D surface to define color and details. Several maps like the diffuse map, the normal map, reflection map and others determine how the surface looks under a certain light and at a certain angle. Many texture examples can be found in the Unity Asset Store or other free sources online (e.g., https://textures.com).



Interior fabric
Shader Universal Render Pipeline/Lit

Surface Options

Workflow Mode	Metallic
Surface Type	Opaque
Render Face	Front
Alpha Clipping	☐
Receive Shadows	✓

Surface Inputs

Base Map	🟦
Metallic Map	0
Smoothness	0.021
Source	Metallic Alpha
Normal Map	2.74
Occlusion Map	0.78

Figure 4.9. Parameters of a Universal Render Pipeline Material.

When importing a model, the unit conversion should be set appropriately or the scale of the object might be ridiculously wrong. After the 3D models are imported into Unity, materials that use these texture maps need to be created and assigned accordingly.

4.3.3. Environment Recreation

Initially, the option to procure the model of an airport with correct markings in place was considered. Unfortunately, none that meet the criteria was found. A simple runway and taxiway system that was found on the 3D Warehouse was modified to include a small apron.
This model served as a testing ground during the first stages of development.

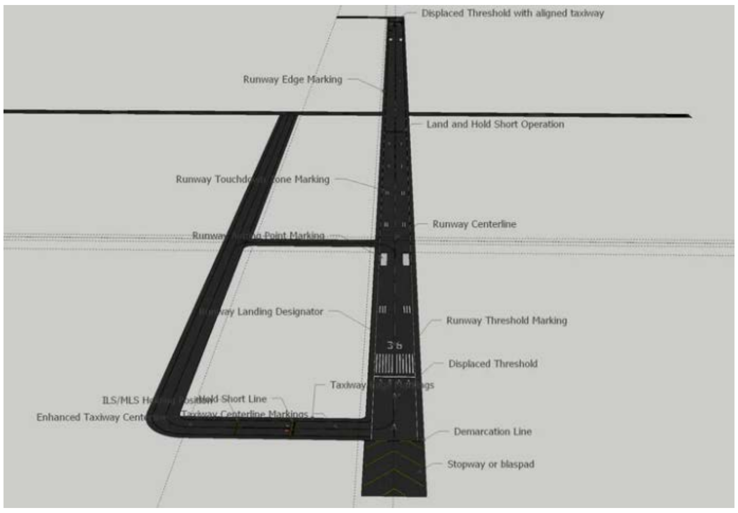

Displaced Threshold with aligned taxiway
Runway Edge Marking
Land and Hold Short Operation
Runway Touchdown Zone Marking
Runway Centerline
Runway Aiming Point Marking
Runway Landing Designator
Runway Threshold Marking
Displaced Threshold
Taxiway
Hold Short Line
Taxiway Centerline Markings
Enhanced Taxiway Center
Demarcation Line
Stopway or blaspad

Figure 4.10. Initial test environment model found on the 3D Warehouse [32].

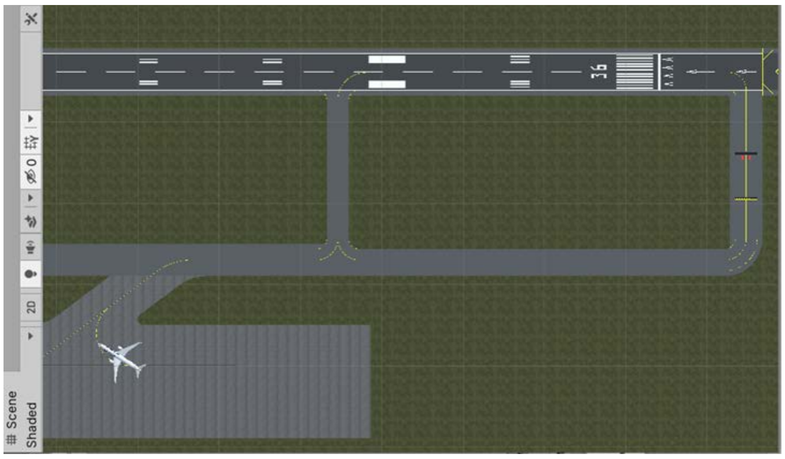


Figure 4.11. Initial test environment modified and imported into a Unity scene.

The resolution of Google Earth images is sufficiently high to be used directly as textures on most areas of the recreated environment. However, thin lines like centerlines and smaller text seem blurry, especially when viewed at an angle.

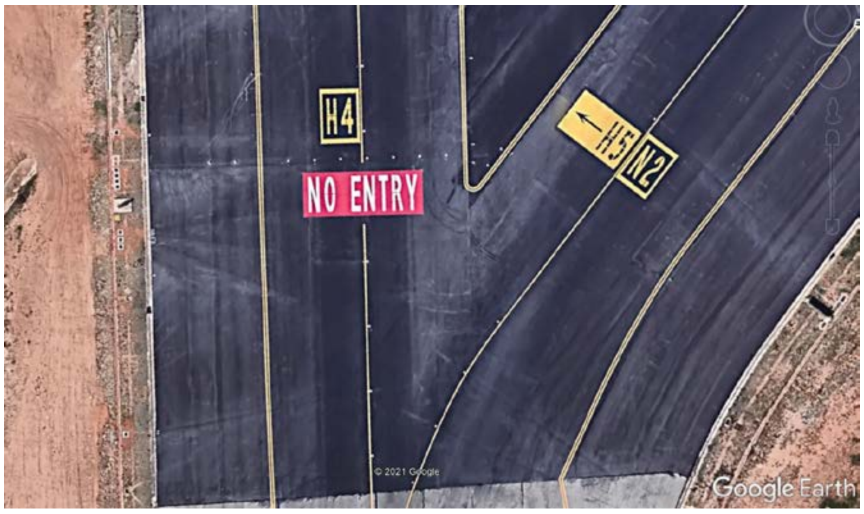


Figure 4.12. Taxiway markings at Valencia Airport. Captured using Google Earth.

Several markings were therefore recreated in a higher resolution using Adobe Photoshop. These images are simple shapes and text, overlaid with a concrete texture. A normal map aligned to the concrete can be added to the Unity material to increase its realism.

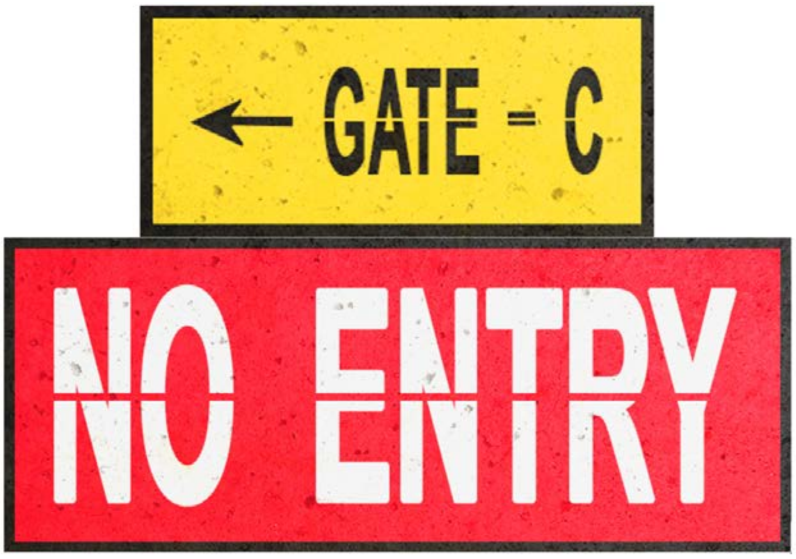


Figure 4.13. Examples of textures for markings created by me.

The airport model was created from scratch and it took around 60 hours to complete. A larger airport would have certainly taken longer.

In SketchUp, a two dimensional plane was textured with a composite image comprised of several Google Earth captures. The most important lines were simply traced over, making sure to keep correct alignments.

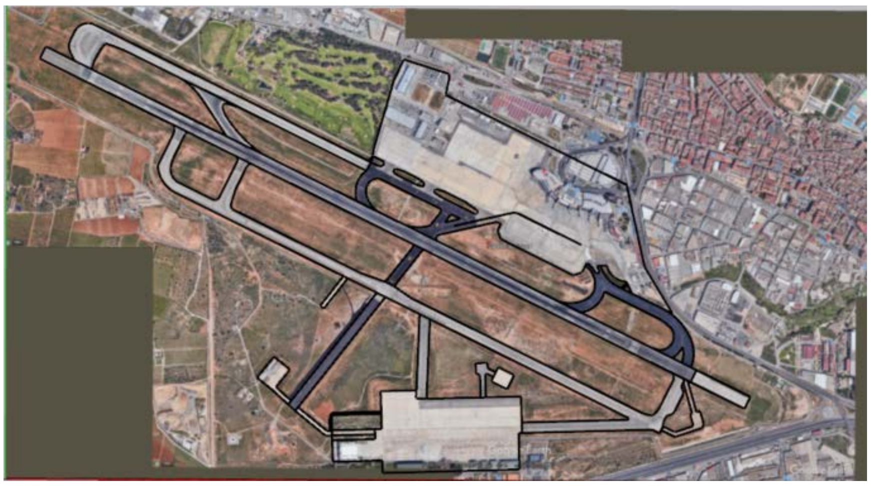


Figure 4.14. Composed Google Earth image imported into SketchUp to model the airport layout.

This method ensures that all measurements will be accurate and so will the position and scale of the different markings on the apron, taxiways and runway. These markings make use of the textures that were created previously. Although not all of them were implemented due to time constraints, the markings that seem most important to complete the procedure described in chapter 2 were included. The service road that was mentioned in the design chapter was placed 55 meters away from the taxiway centerline to add a generous safety margin.

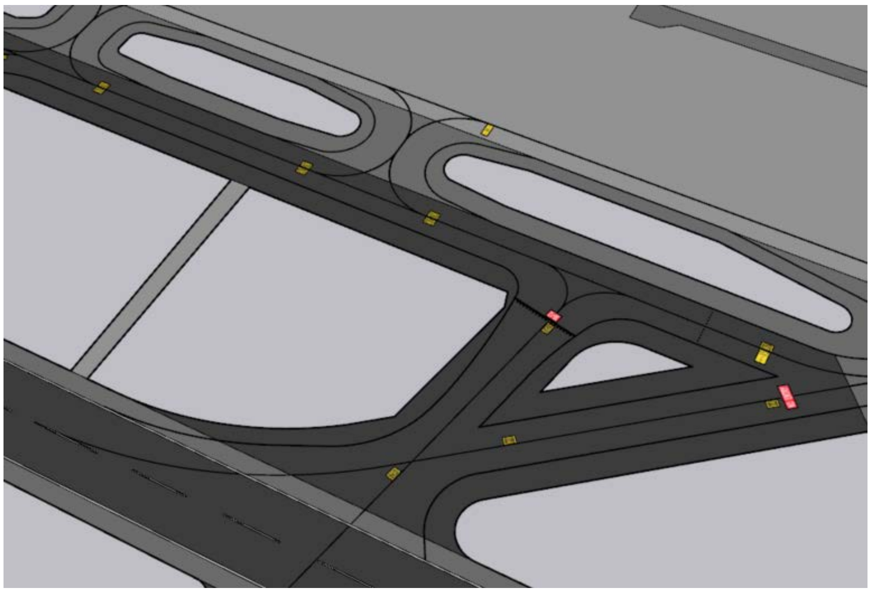
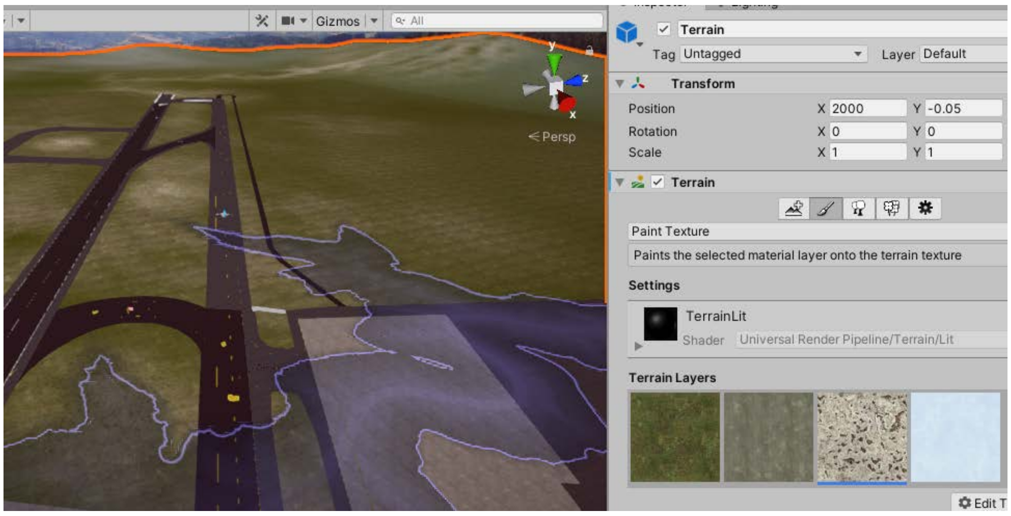


Figure 4.15. Airport 3D model in progress.

The model was imported into Unity and textures for the different surfaces were assigned. A terrain object was created to add some elevation and it was painted with grass, dirt and sand textures. These assets are included in the Terrain Tools Sample Package, created by Unity and available for free.



Terrain
Tag Untagged Layer Default
Transform
Position X 2000 Y -0.05
Rotation X 0 Y 0
Scale X 1 Y 1
Terrain
Paint Texture
Paints the selected material layer onto the terrain texture
Settings
TerrainLit
Shader Universal Render Pipeline/Terrain/Lit
Terrain Layers
Edit T

Figure 4.16. Building Unity terrain after importing layout.

A model of the terminal building could be found on the 3D Warehouse. It is the same model that is currently visible in Google Earth. A few materials were created and assigned to achieve a surprisingly good result for a model with such a low polygon count [33].

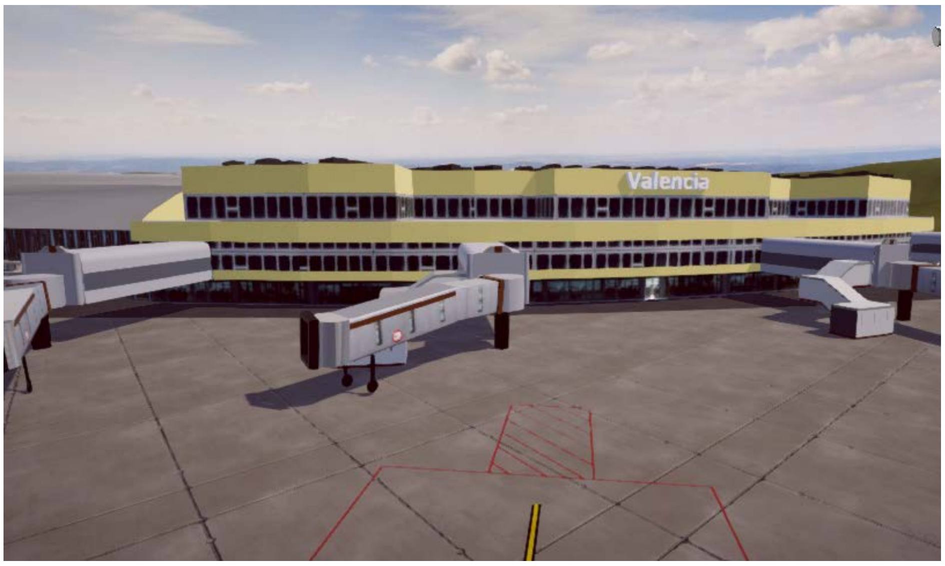


Figure 4.17. Imported terminal 3D model [33].

A few weeks after this airport model was completed, a user of an X-Plane community forum posted a free scenery package that features high resolution textures and night lighting. I could not find any way to convert these files to any format that could be imported into Unity so this option was unfortunately discarded.

4.4. Physics Simulation

Unity has built-in tools to make the development of the physics simulation quite straight-forward.

Both the Boeing 787 and the TaxiBot objects require colliders to detect when they come into contact with other colliders and Rigidbodies to define mass, center of gravity, and allow them to receive forces. In object-oriented projects such as this, the Nvidia PhysX is used.

For the TaxiBot, the Rigidbody component was configured with a mass of 27.000 kg. The mesh collider component uses the same imported 3D model but the convex option is enabled. This option simplifies the mesh to its bounding convex shape as it saves on performance. The four wheel objects themselves use a different 3D model and they only serve visual purposes.

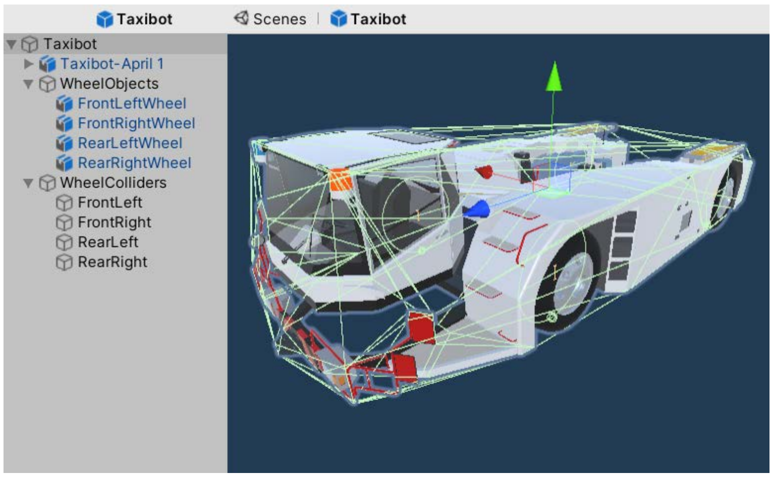
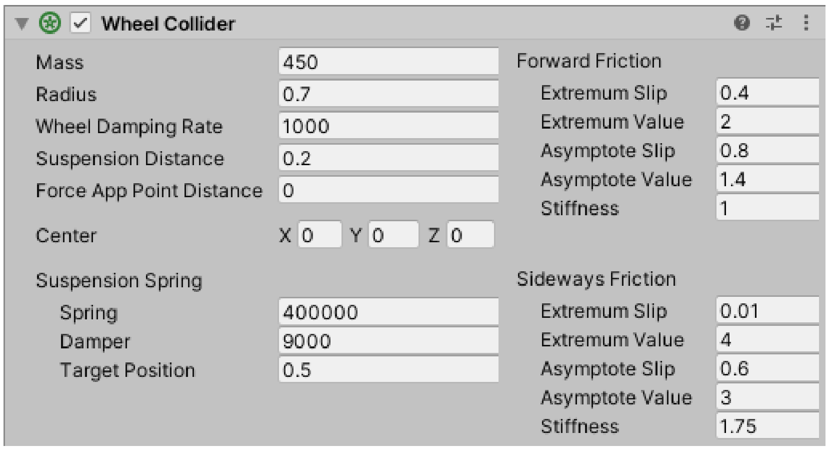


Figure 4.18. TaxiBot prefab showing object hierarchy.

Objects with the Wheel Collider component are used to define physical characteristics of the wheels. This collider component is included in the Unity libraries and it suffers from a few shortcomings. Other, more complete wheel components and scripts can be found on the Asset Store but at this moment, none of them are free. Parameters like the wheel mass, suspension distance and spring constant, forward and sideways friction all need to be playtested thoroughly. These values in Figure 4.19 make the vehicle closely replicate the behavior of the TaxiBot that is displayed in videos.

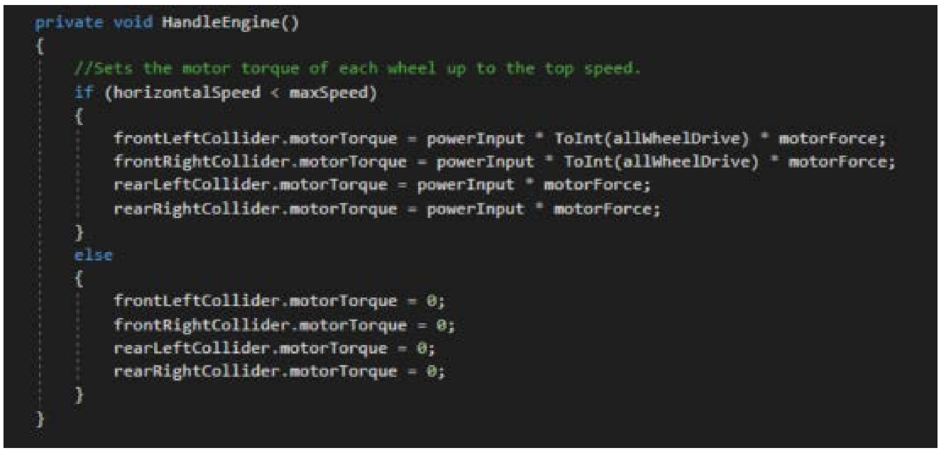

Wheel Collider

Mass	450	Forward Friction	
Radius	0.7	Extremum Slip	0.4
Wheel Damping Rate	1000	Extremum Value	2
Suspension Distance	0.2	Asymptote Slip	0.8
Force App Point Distance	0	Asymptote Value	1.4
		Stiffness	1
Center	X 0 Y 0 Z 0		
Suspension Spring		Sideways Friction	
Spring	400000	Extremum Slip	0.01
Damper	9000	Extremum Value	4
Target Position	0.5	Asymptote Slip	0.6
		Asymptote Value	3
		Stiffness	1.75

Figure 4.19. Selected parameters for the Wheel Collider objects.

The script that controls the TaxiBot will need to read the user inputs and apply forces to the wheels to simulate the behavior of the vehicle. A function reads whether the user is pressing the keys to go forward or backward (W and S respectively in this implementation) and normalizes it, positive meaning forward and negative meaning reverse. A similar operation is performed to read steering input, encoding right as the positive and left as the negative direction (D and A). Braking can be a single value from 0 to 1 (Space key). If a keyboard is used, all of these parameters could be flags because a key is either pressed or it is not. However, if a different control device is in place, the experience could make use of analog input methods. The values mentioned are all floating point numbers to enable this possible feature.

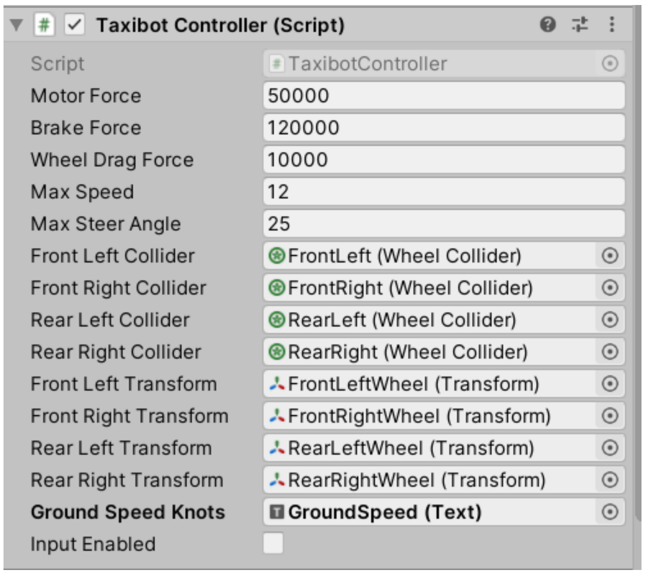
Different functions that handle the engine, steering and brakes are in place. They change the torque and steering angle (built-in parameters of the Wheel Collider class) according to the previously explained input variables. The TaxiBot specifications mention that the vehicle can operate using traction in two wheels or all four wheels. A flag to change this parameter was included.



```
private void HandleEngine()
{
    //Sets the motor torque of each wheel up to the top speed.
    if (horizontalSpeed < maxSpeed)
    {
        frontLeftCollider.motorTorque = powerInput * ToInt(allWheelDrive) * motorForce;
        frontRightCollider.motorTorque = powerInput * ToInt(allWheelDrive) * motorForce;
        rearLeftCollider.motorTorque = powerInput * motorForce;
        rearRightCollider.motorTorque = powerInput * motorForce;
    }
    else
    {
        frontLeftCollider.motorTorque = 0;
        frontRightCollider.motorTorque = 0;
        rearLeftCollider.motorTorque = 0;
        rearRightCollider.motorTorque = 0;
    }
}
```

Figure 4.20. Torque assignment using built-in Wheel Collider objects.

The initial engine parameters were obtained translating the power specifications to torque considering a wheel of around 1.4 meters in diameter. Afterwards, tuning was required to achieve the dynamics shown in reference videos. The maximum speed of 12 meters per second derive from the maximum speed of 23 knots under load. The maximum steering angle was a value that could not be procured form the specifications. The value that was obtained is one that ensures that the maximum steering radius is approximately that of the unassisted Boeing 789. This value is closely related to the clamping mechanism of the aircraft nose wheel and how the joint behaves.



Taxibot Controller (Script)

Script	TaxibotController
Motor Force	50000
Brake Force	120000
Wheel Drag Force	10000
Max Speed	12
Max Steer Angle	25
Front Left Collider	FrontLeft (Wheel Collider)
Front Right Collider	FrontRight (Wheel Collider)
Rear Left Collider	RearLeft (Wheel Collider)
Rear Right Collider	RearRight (Wheel Collider)
Front Left Transform	FrontLeftWheel (Transform)
Front Right Transform	FrontRightWheel (Transform)
Rear Left Transform	RearLeftWheel (Transform)
Rear Right Transform	RearRightWheel (Transform)
Ground Speed Knots	**GroundSpeed (Text)**
Input Enabled	☐

Figure 4.21. Parameters of the TaxiBot controller script.

Public variables or those with a specified *SerializeField* attribute can be loaded from the inspector window in the editor. To manage the forces that affect the wheels, the script must have access to the Wheel Collider objects. The script also references the wheel objects that only have a renderer. A function can change the position and rotation of the wheels using the transforms of these objects.

Up until this moment, the TaxiBot has just been implemented as a very simple car. The interaction with the plane is one of the most important aspects as the accuracy of the simulation is one the main attributes to validate the experience.

Each of the different groups that make up the model has its own mesh collider. This allows the functionality of moving each object independently, separating them from the plane object or collapsing the entire model. The plane object has an assigned mass of 180.000 kg, which is a perfectly normal takeoff weight. The plane object also makes use of Wheel Collider components on the landing gear.

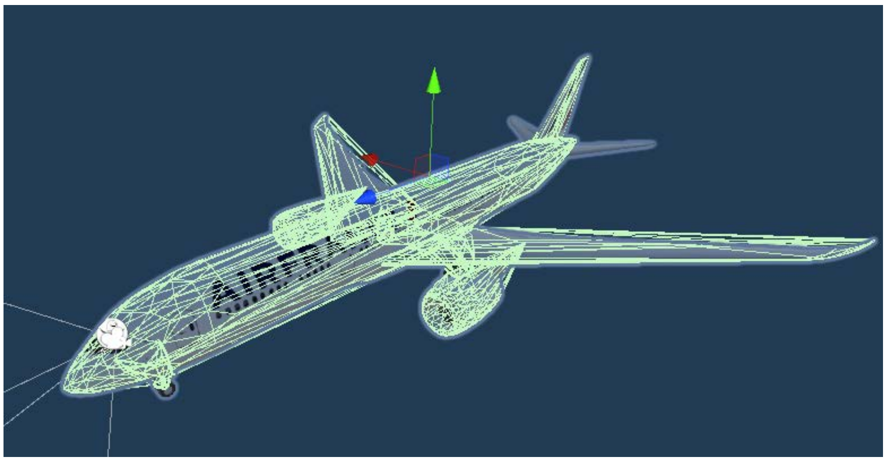


Figure 4.22. The imported Boeing 787. Each part major part of the aircraft is a different collider.

A Hinge Joint component is in place to connect it to the TaxiBot Rigidbody. The position and axis need to be configured, as well as some limits. Lufthansa asserts that the TaxiBot wheels are steered according to the detected nose landing gear steering angle [34]. The maximum angles that the joint can take are defined by the aircraft specifications. The fact that steering is no longer directly driven by the nose wheel makes the maximum angle not exactly 70 degrees, so the turning radii were emulated instead. The minimum nose turning radius of 32 meters [35] was employed to define the TaxiBot wheel steering and Hinge Joint limits.

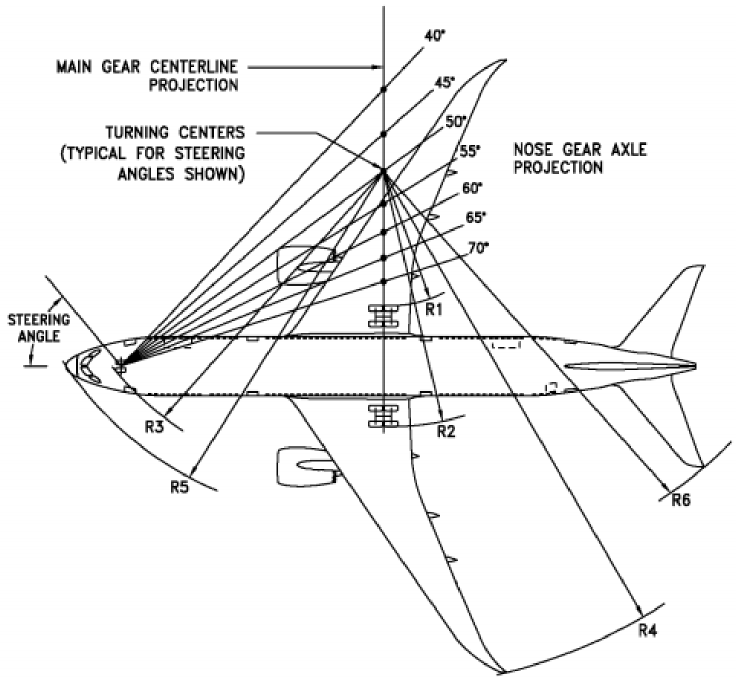

MAIN GEAR CENTERLINE PROJECTION
TURNING CENTERS (TYPICAL FOR STEERING ANGLES SHOWN)
STEERING ANGLE
NOSE GEAR AXLE PROJECTION
40°
45°
50°
55°
60°
65°
70°
R1
R2
R3
R4
R5
R6

Figure 4.23. Different turning radii for the Boeing 787 [35].

One of the shortcomings of using the built in Wheel Controller objects emanates from the fact that wheels can “go to sleep” if they are resting for a few seconds. This is a feature of all physics objects in Unity and it is used to save on resources. When the simulation detects that it should move again, it starts calculating the net force, velocity and rotation again. The translation of forces that allow the TaxiBot to push or pull the airplane works sufficiently well in movement. However, at the start of the simulation, the landing gear wheels are resting and the forces applied by the TaxiBot can never move them. This bug has been known for a few years and it requires a workaround. In this case, a similar script that reads the power input was used and assigned to the Boeing 787 GameObject. If the TaxiBot is accelerating, then a very minor torque is also applied to the airplane wheels that contact the ground.

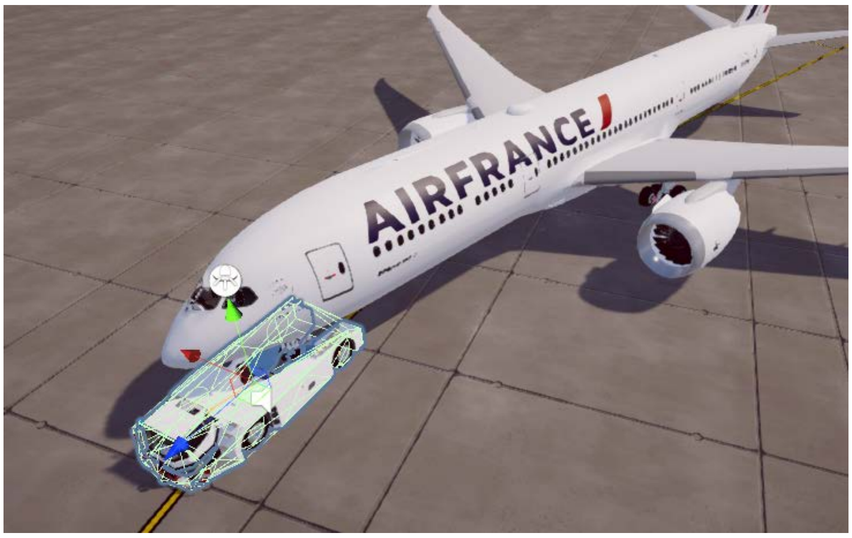


Figure 4.24. TaxiBot and Boeing 787 models connected through a hinge component.

While the development of these scripts was straightforward, a meaningful amount of time was spent tuning every parameter. In total, approximately 30 hours of work were spent writing the described scripts, tuning and testing.

4.5. Events

Besides following the initial instructions to complete the taxi route, elements that disrupt the procedure were designed and implemented. It is preferable to implement events that might happen in a real environment in order to add learning potential to the experience.

4.5.1. Emergency Vehicle

Both a pilot and a ground vehicle operator have a responsibility to avoid collisions. However, controllers exercise their best judgment to avoid conflicts and their instructions are mandatory. In this situation, an emergency scenario is depicted. As the Boeing 787 approaches Gate-D, the ground traffic controller will issue an instruction to stop and give way to a ground vehicle using exit H6.

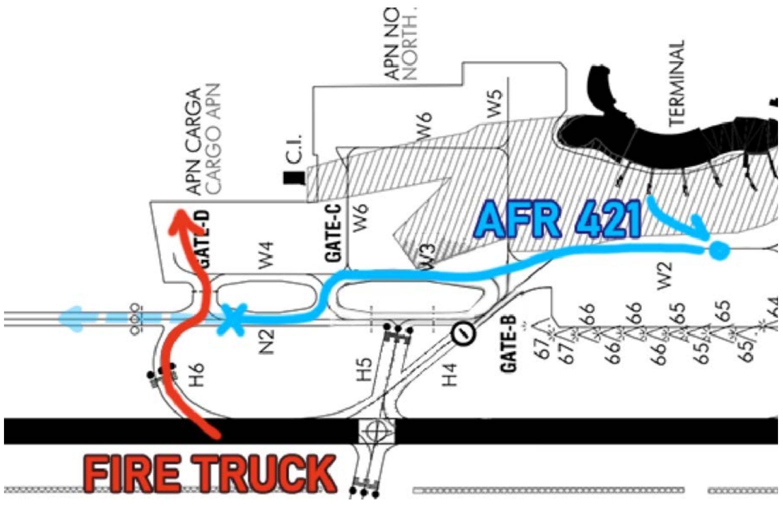


Figure 4.25. Design of the first obstacle.

As for the implementation, the 3D model of an aircraft rescue and fire fighting vehicle was downloaded from the 3D Warehouse, imported into Unity, and simple materials (solid colors and reflectivity) were added.

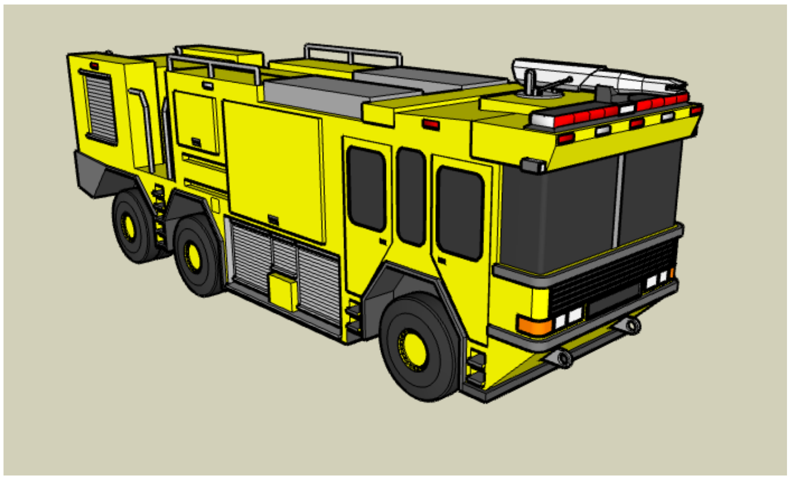


Figure 4.26. Airport Fire Engine found on the 3D Warehouse [36].

Like before, Rigidbody and collider components were added to the object. A different GameObject was created to contain several children objects that hold coordinates of the points that define the desired path.

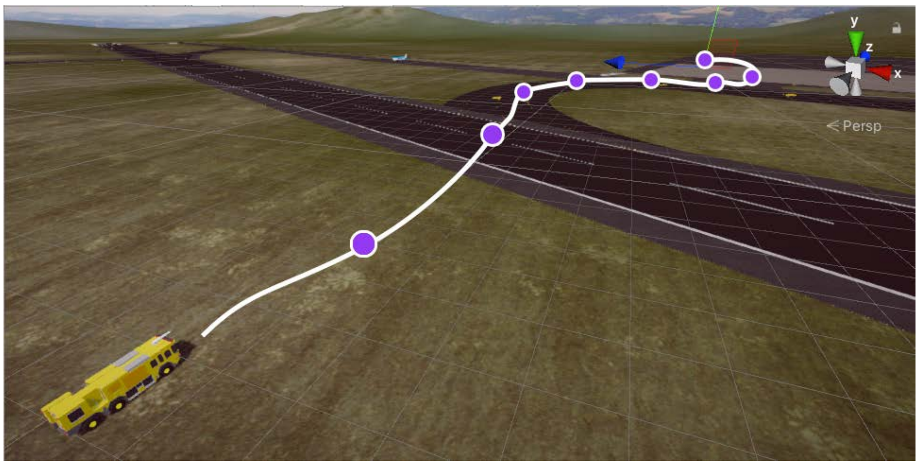


Figure 4.27. Visualization of the intersecting route.

The script that controls the vehicle is quite simple. On proximity, the object is activated and the velocity parameter is set to the speed variable (in this case, 20 m/s) aiming towards the next point. The object rotates at a predetermined angular velocity to align itself with its trajectory so it does not switch directions suddenly. This behavior is very far from an adequate vehicle simulation, but at a distance, it is acceptable for the purpose it serves.

Static or dynamic objects can detect collisions in Unity using the built-in collider components. Instead of using it to define the boundaries of a physical body, a developer can set a collider object as a trigger. The component then calls a different function on the script attached to the GameObject depending on the event that occurs, like entering, staying, and leaving the trigger volume. In this case, triggers are used to enable the movement of the vehicle in its trajectory, detect if the user does not stop promptly, and if the aircraft comes into conflict with the emergency vehicle.

In a real airport environment, the specific distances are determined by the controller, who should take into account the course of both vehicles and the required distance to brake safely. In this scenario, the event is completely scripted and the user will always be instructed to stop around 150 meters away from H6. The implementation of the traffic controller system will be disclosed later on.

4.5.2. Leading Aircraft

The second event that was implemented is a leading aircraft on taxiway N3. In this case, the traffic controller needs to consider several factors to determine the minimum separation, like braking distance, relative velocities, reaction times and safety margins.

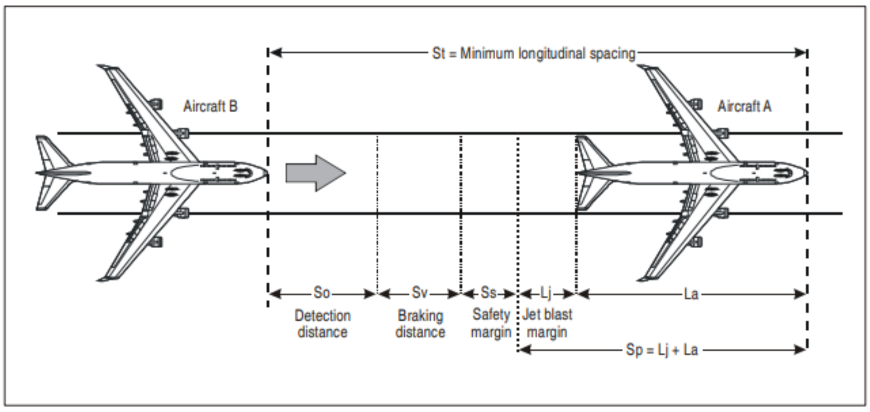


Figure 4.28. Factors to consider in longitudinal spacing [37].

According to ICAO regulation, spacing between two taxiing aircraft should never be under 50 meters. In most airports however, distances are just determined based on the types of leading and trailing aircraft. In this case, the Embraer 190 was selected. Its maximum takeoff weight is 51.800 kg, making it a medium aircraft according to the ICAO wake turbulence category. The trailing aircraft is the Boeing 787, a heavy aircraft. A common minimum separation distance in these cases is 200 meters [38]. In any case, the traffic controller should issue instructions with sufficient anticipation.

The Embraer 190 3D model was easily procured from the 3D Warehouse and imported into Unity. Its Rigidbody mass is set to 50.000 kg and it uses a script very similar to the one on the fire truck.

Figure 4.29. Embraer 190 3D model downloaded from the SketchUp 3D Warehouse [39].

In this case, the trajectory defined by the list of points follows the taxiway N, turns on H8 and takes off from RWY 12. In addition to the list of coordinates, there is also a list of speeds the aircraft reaches in each stretch. The aircraft starts taxiing at a slower velocity than the maximum speed of the TaxiBot under load, forcing the user to slow down. Afterward, it rolls at 30 knots, slows down for the curves, holds for a moment and accelerates until it takes off at 140 kts.

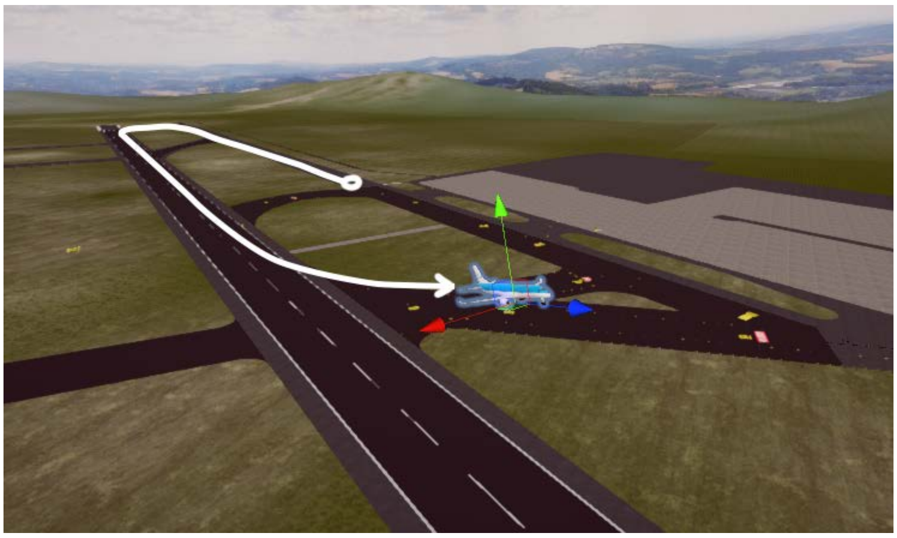


Figure 4.30. Taxi and take-off trajectory of the aircraft.

In the experience, the traffic controller issues an instruction to the user (AFR 421) to stop, instructs the smaller plane (KLM 190) to continue taxiing, waits a few seconds, and finally instructs AFR 421 to continue taxiing.

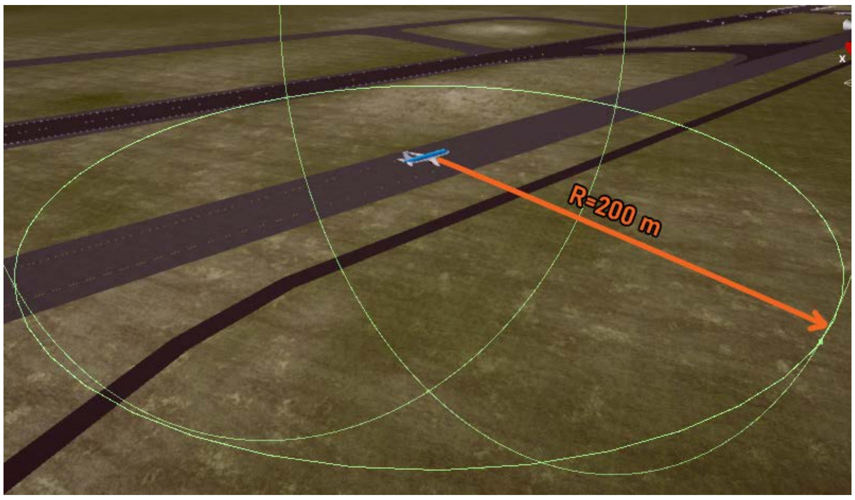


Figure 4.31. Trigger zones to check if separation is above 200 meters.

Triggers to activate the aircraft and detect if separation minima is not maintained are in place. A circular zone that detects if separation is under 200 meters follows the plane object through its trajectory, and it is important to understand the reasoning behind this distance. Nonetheless, if the controller issues an instruction to stop, a trigger should detect if the user stops appropriately, considering reaction time and braking distance.

4.6. Automatic Ground Movement Control

In order to guide the pilot to perform the taxi procedure correctly, the experience should include a system that tracks the position of the user controlled vehicles. In an airport environment, the pilot interacts with the traffic controller that is in charge of ground movements via radio. The simulated traffic controller will send similar messages that can help the user and use proper aviation vocabulary.

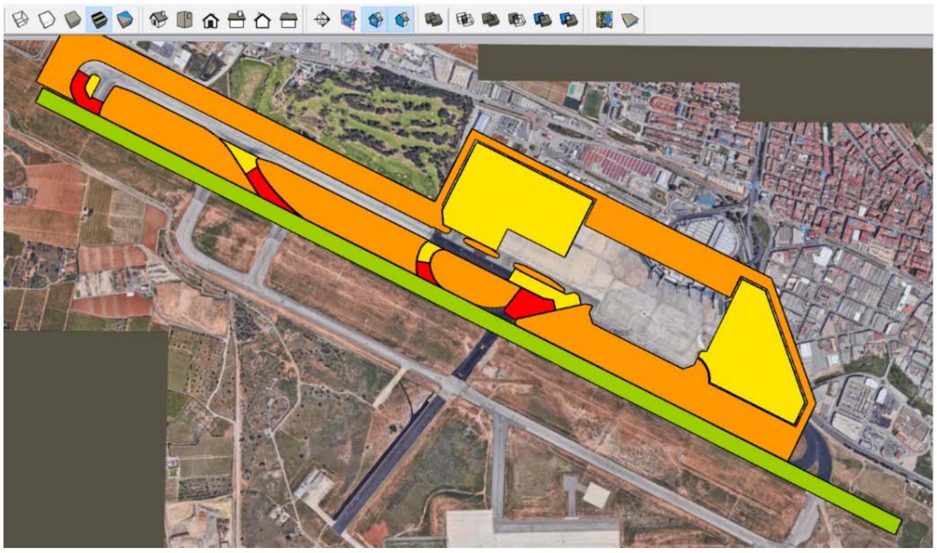


Figure 4.32. Different airport zones modeled in SketchUp.

Different areas of the airport were modeled using SketchUp and the original composite satellite image obtained from Google Earth. These simple models are imported into Unity and labeled using a simple script that defines a zone name and calls a function when a collision with either the TaxiBot or the Boeing 787 is detected. The solution in place uses a parent object that contains all static zones of the airport. When the function is called, the parent object uses a long conditional statement that can send messages to the pilot according to the zone that calls it.

For this purpose, a messaging system was added and placed in the Head-Up Display (HUD). The user can receive and send messages using a panel on the left. The simulated traffic controller cannot read the messages, but the feature could be more useful in a different version of the experience.

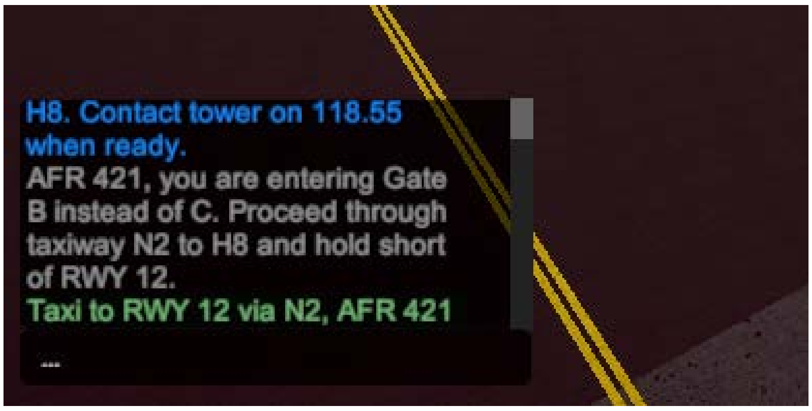

H8. Contact tower on 118.55 when ready.
AFR 421, you are entering Gate B instead of C. Proceed through taxiway N2 to H8 and hold short of RWY 12.
Taxi to RWY 12 via N2, AFR 421
...

Figure 4.33. Examples of controller and user messages.

The implementation of the messages panel was simple and many resources can be found online as a chat box is very common in games. The script adds strings to an array and it creates text objects as children of a Content Box, which is another built-in Unity component. When the user clicks on the input field, the input to control the TaxiBot is disabled until the user writes and sends the message, or clicks away from the panel.

19 written messages for the traffic controller were drafted. A few examples of these messages are displayed below.

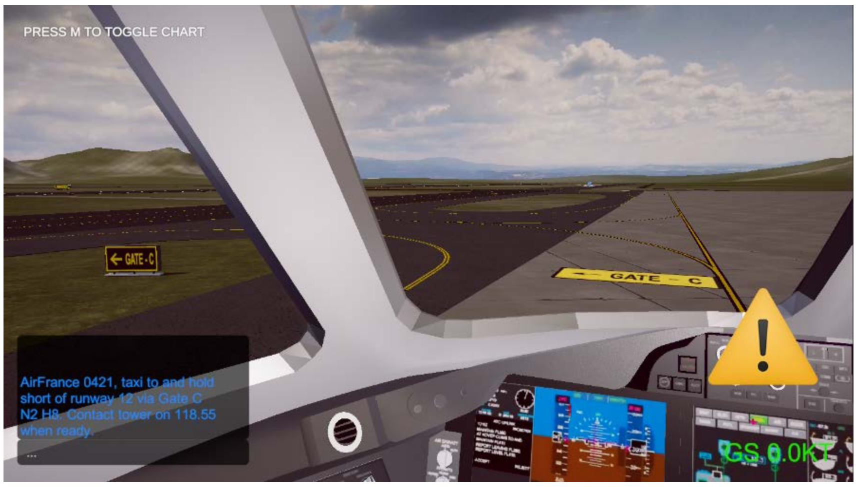
- AFR 421, stop at once. You are about to collide with the terminal.
- AFR 421, exit the APRON through Gate C.
- AFR 421, you have caused an excursion. Return to the taxiway if you do not need assistance.
- AFR 421 stop immediately. You have not been cleared to enter the runway.

These messages were later recorded and the audio files are played at the same moment the respective message is sent. An Audio Source component was added to the parent airport zones object and a function swaps and plays the requested Audio Clip on command.

It is important to note that many mistakes that can be committed in the experience are more dangerous than the traffic controller suggests. A taxiway excursion can damage the landing gear very seriously and the flight would likely need to be evacuated. Failing the experience would require the user to start a new attempt entirely. Instead, a scoring system was designed to aid the pilot in recognizing if a mistake has been made.

Points that reward users who accomplish a task correctly is one of the basic gamification strategies. A scoring system that makes use of the detection and instructions scripts was implemented. The points themselves are arbitrary, but the consequences of their inclusion will be discussed in the validation stage.

Apart from route mistakes, disregarding separation criteria and leaving the taxiway unexpectedly, other aspects like centerline accuracy and time to finish the procedure also influence score to a smaller degree.

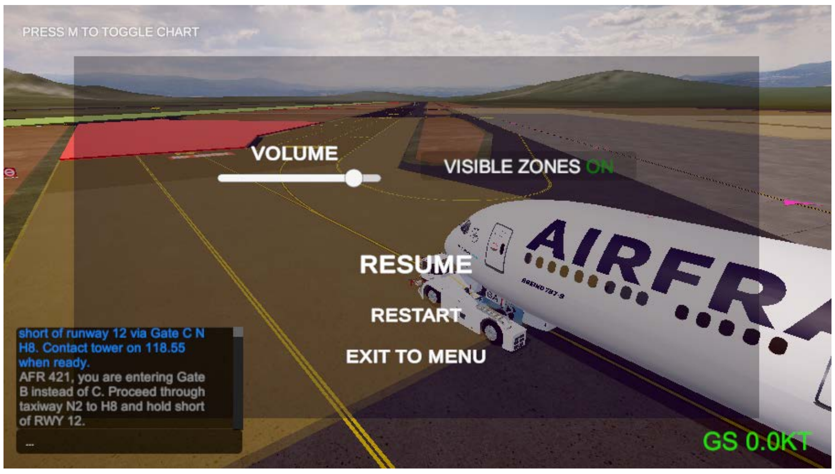


AirFrance 0421, taxi to and hold short of runway 12 via Gate C N2 H8. Contact tower on 118.55 when ready.

Figure 4.34. Observing airport signs in first person perspective.

If the pilot is off the centerline by 3 meters or more, a warning sign appears on the HUD to alert of the potential danger. The Boeing 787 is a class E aircraft because its main landing gear is 9.80 meters wide. Considering the taxiway is 23 meters wide, the remaining margin to each side is 6.5 meters. On top of that, the shoulders are paved a further 10 meters in both sides for safety. While keeping the plane centered on the straight sections is easy enough, getting used to the position of the nose wheel while turning is certainly challenging.

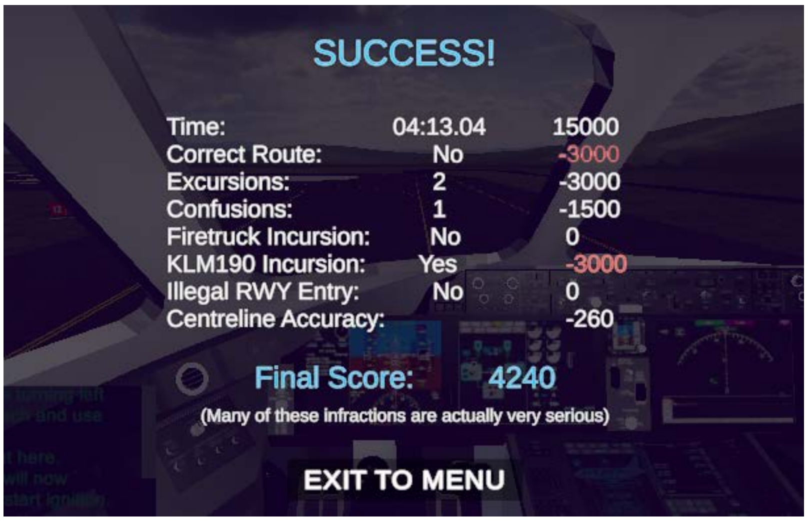
A toggle to display the mapped airport zones to the user was added in a pause menu. Being able to see the zones to avoid and the requested route simplifies the experience to users with little knowledge of airport maneuvering and situational awareness.



short of runway 12 via Gate C N H8. Contact tower on 118.55 when ready.
AFR 421, you are entering Gate B instead of C. Proceed through taxiway N2 to H8 and hold short of RWY 12.

Figure 4.35. Functionality to toggle visibility of the infraction zones.

Once the user stops at the position where the TaxiBot will disconnect (checking the presence of the TaxiBot in the specified zone and that its velocity is approximately zero), the score and all the factors behind it are displayed.



SUCCESS!

Time:	04:13.04	15000
Correct Route:	No	-3000
Excursions:	2	-3000
Confusions:	1	-1500
Firetruck Incursion:	No	0
KLM190 Incursion:	Yes	-3000
Illegal RWY Entry:	No	0
Centreline Accuracy:		-260

Final Score: 4240

(Many of these infractions are actually very serious)

Figure 4.36. Results panel with the score obtained throughout the experience.

If all areas of an airport are modeled, imported and labeled, different ground operations could be implemented. To create a new trajectory, the changes to make are the starting and final positions, creating links between the erroneous zones and the script in charge of scoring, and setting the appropriate messages. In the end, this evaluation approach is very limited. Each new mission needs to be developed attentively, and they constitute a significant time investment, especially if there are many messages in text or audio form. The fact that the user cannot communicate with the traffic controller removes a very important aspect of airside operations.

4.7. Networking

During development and initial tests, I noticed that people with low qualification in airport procedures were lost and they could not find the described route. In these cases, I would explain the use of centerlines and indicate their position on the ground movement chart. With minimal indications, the experience was much more manageable by a larger audience. Pilots or vehicle operators can ask the traffic controller for assistance if they are lost or confused in an airport. The automatic controller system cannot provide dynamic indications, so a different solution was explored.

While it was not one of the requirements initially, networking was implemented to connect two or more users to the same virtual environment. The networking functionality creates several opportunities that were not possible before.

4.7.1. Additional Tools

The official Unity multiplayer solution is under development at the moment. Many functionalities have not been implemented yet, so it is not recommended and an alternative is required.

Mirror is an API (Application Programming Interface) built for Unity, by third-party developers. It is open source and it can be imported as a package for free. Its use in Unity is based on pre-built components that can be added to GameObjects to add functionalities like synchronizing variables, sending commands to execute functions on the server or a client, enabling and flagging sections of code as “server only” or “client only”.

ParrelSync is an editor extension for Unity that allows a developer to create clones of a project and synchronizing changes between them almost instantly. With it, a user can open two instances of the same project, and test the networking solution between them. It is not necessary but it saves several minutes of testing time. The alternative would be building a copy of the project and running it for every test. ParrelSync is free to use and available under MIT License as well.

4.7.2. Network Implementation

Mirror provides a few components that are required to create the network and define its parameters. A copy of the same Valencia Airport scene was created, a Network Manager object was added, and some prefabs were updated to contain the following components.

- Transport Component:

  The script that configures the protocol in use to send and receive data through the network. Mirror supports several transports that are based on different protocols. UDP and TCP are the most popular data transmission protocols. In this case, the default transport kc2k, a C# implementation of KCP, was used. KCP is a reliable protocol based on TCP (Transmission

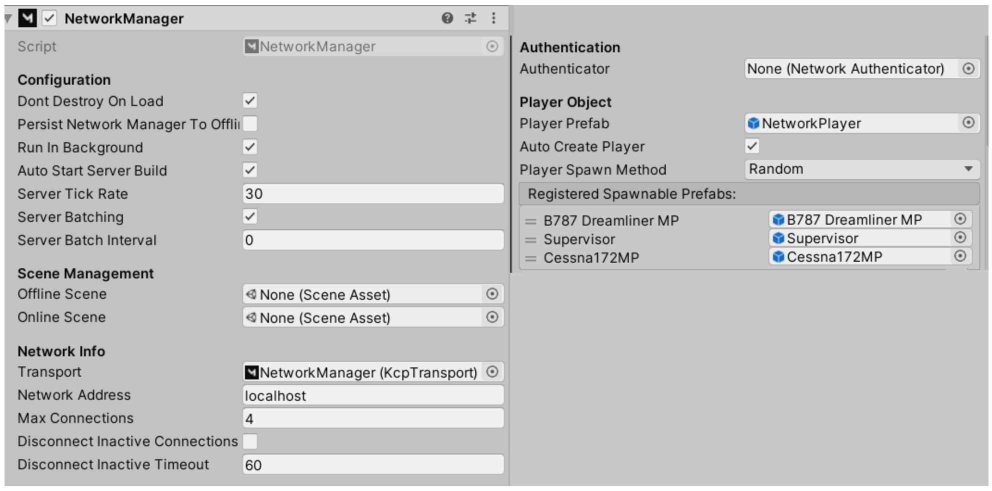
Control Protocol) except it prioritizes network latency at the cost of higher bandwidth usage. Depending on requirements of the experience, a different transport based on UDP would fare better as it can send update date at a higher rate. Such a solution would improve the synchronization of position and velocity.

- Network Identity:

  This component is required to spawn and track instances of the object across the network with separate identifier codes.

- Network Manager:

  The component that handles the application state, prefab spawning, scene management and network debugging. Some parameters to configure it are server update rate, network address, maximum number of simultaneous connections, and a list of “spawnable” objects.



NetworkManager

Script	NetworkManager
Configuration	
Dont Destroy On Load	✓
Persist Network Manager To Offli	
Run In Background	✓
Auto Start Server Build	✓
Server Tick Rate	30
Server Batching	✓
Server Batch Interval	0
Scene Management	
Offline Scene	None (Scene Asset)
Online Scene	None (Scene Asset)
Network Info	
Transport	NetworkManager (KcpTransport)
Network Address	localhost
Max Connections	4
Disconnect Inactive Connections	
Disconnect Inactive Timeout	60
Authentication	
Authenticator	None (Network Authenticator)
Player Object	
Player Prefab	NetworkPlayer
Auto Create Player	✓
Player Spawn Method	Random
Registered Spawnable Prefabs:	
B787 Dreamliner MP	B787 Dreamliner MP
Supervisor	Supervisor
Cessna172MP	Cessna172MP

Figure 4.37. Parameters of the Network Manager component.

- Network Transform:

  This component synchronizes position, rotation, and scale of the GameObject over the network using the required components mentioned earlier. Update rate and interpolation factors can be set. However, the component is very limited in options, and a more advanced system could be employed.

- Network Manager HUD:

  The default component to provide a simple display that allows users to input a network address and choose to play the roles of host, client or server. Integrating this elements into the main menu is simple but as many other aspects in this project, it takes time. The default configuration is used.

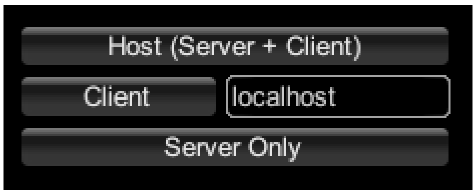


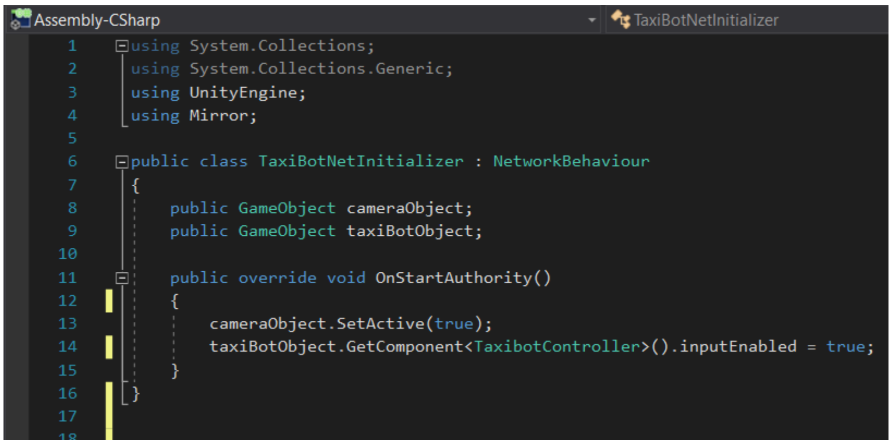
Figure 4.38. Network Manager HUD panel as it appears during runtime.

These components and their use is very similar to the now deprecated Unity multiplayer tool package UNet. The topology used in the implementation was client hosted, in which one of the clients will also run the server tasks.

4.7.3. Network Profiles

The acting Network Manager for the scene needs to contain references to all the objects with network functionalities it needs to spawn. These objects are new prefabs that contain Network Identities and Transforms. A very quick solution to implement object authority was created.

When the client sends an order to the server to instance a prefab, the Network Manager can identify the client as the object owner. If the Network Transform component has the ‘Client Authority’ option enabled, execution of different blocks of code can be specific to the owner and to every other client connected to the network.



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

public class TaxiBotNetInitializer : NetworkBehaviour
{
    public GameObject cameraObject;
    public GameObject taxiBotObject;

    public override void OnStartAuthority()
    {
        cameraObject.SetActive(true);
        taxiBotObject.GetComponent<TaxibotController>().inputEnabled = true;
    }
}
```

Figure 4.39. Network Initializer script for the TaxiBot. Every object with network functionality makes use of a similar script.

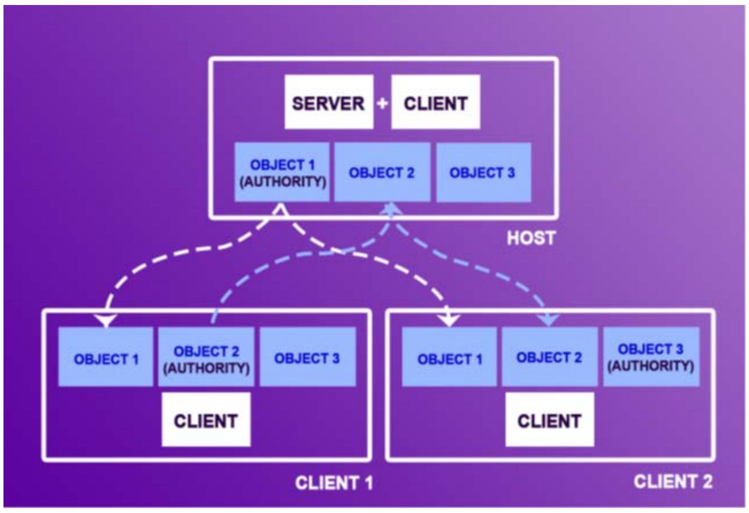
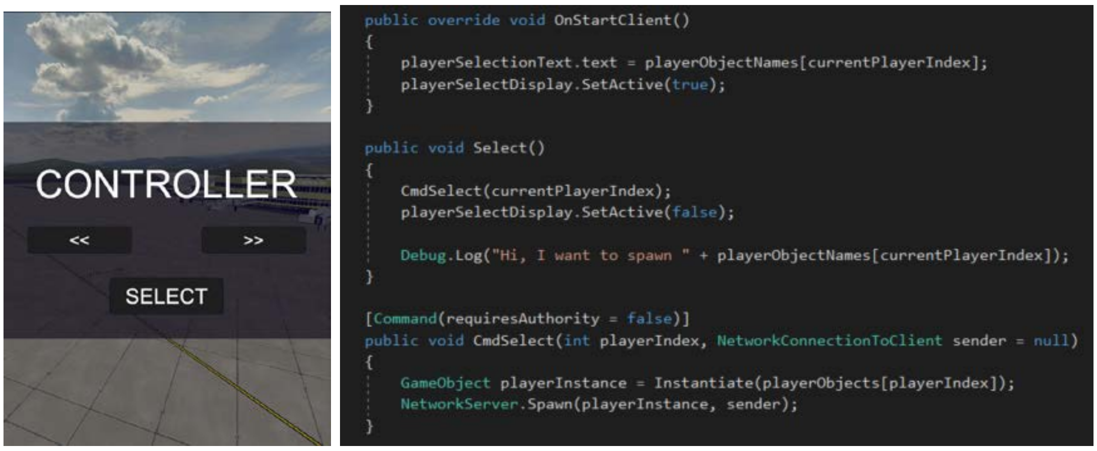


Figure 4.40. Network topology and authority over objects (own image).

The implemented solution consists of a short additional script assigned to the network prefabs. Its only functionality is to enable certain objects or components on the owner client. With this solution, the TaxiBot network prefabs and its corresponding basic prefab share the same script for movement and references. A similar script is in place for the air traffic controller profile. The profile is simply a toggle between a bird's eye view and a free camera.

Right after connecting, a UI panel for the user to select a profile is in place. This object can send the command to spawn a selected prefab. The functions required to display the enable the panel upon connection, select a profile and send the command to instance the prefab to the current host make use of classes and functions only available through Mirror.



```
public override void OnStartClient()
{
    playerSelectionText.text = playerObjectNames[currentPlayerIndex];
    playerSelectDisplay.SetActive(true);
}

public void Select()
{
    CmdSelect(currentPlayerIndex);
    playerSelectDisplay.SetActive(false);

    Debug.Log("Hi, I want to spawn " + playerObjectNames[currentPlayerIndex]);
}

[Command(requiresAuthority = false)]
public void CmdSelect(int playerIndex, NetworkConnectionToClient sender = null)
{
    GameObject playerInstance = Instantiate(playerObjects[playerIndex]);
    NetworkServer.Spawn(playerInstance, sender);
}
```

Figure 4.41. Left: Profile selection panel. Right: Functions required to select and send instancing message to the host.

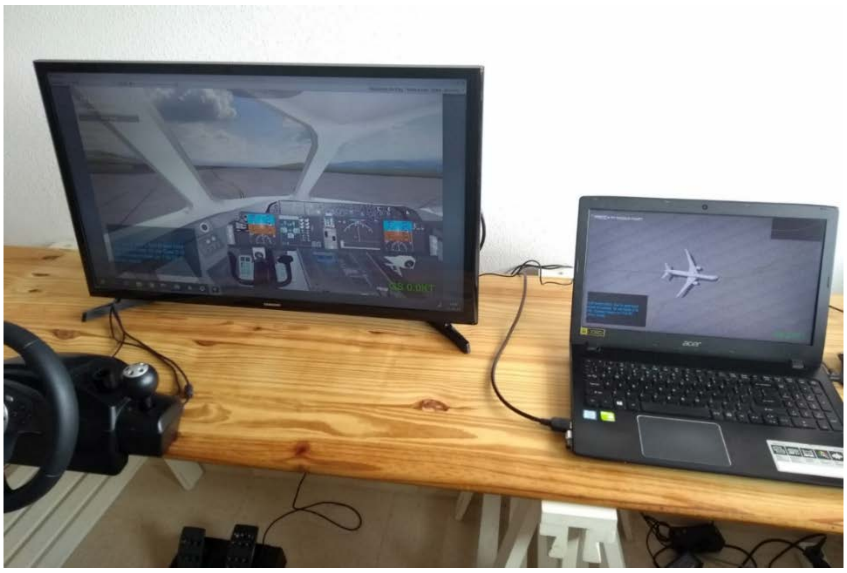


Figure 4.42. Testing two profiles, pilot and controller connected to the same virtual environment.

This system was designed keeping in mind that it should be easy to implement a new profile in the same environment. To test this, I added a new vehicle monitoring the time it takes me. I decided it would be a small aircraft, so a free model of a Cessna 172 was downloaded from the 3D Warehouse.

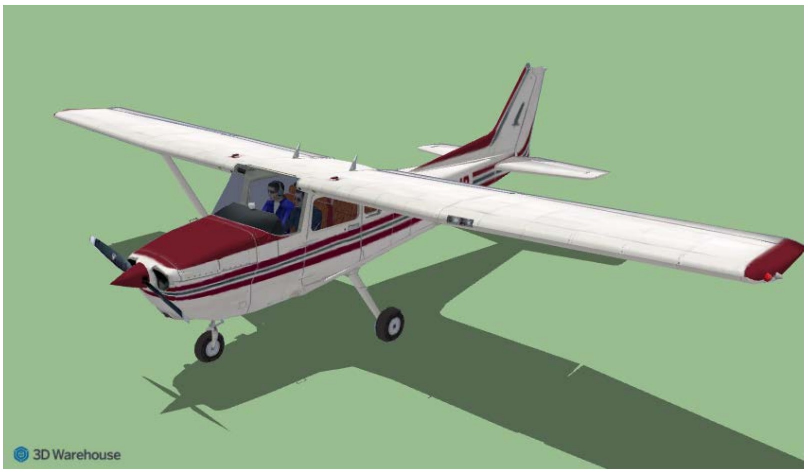
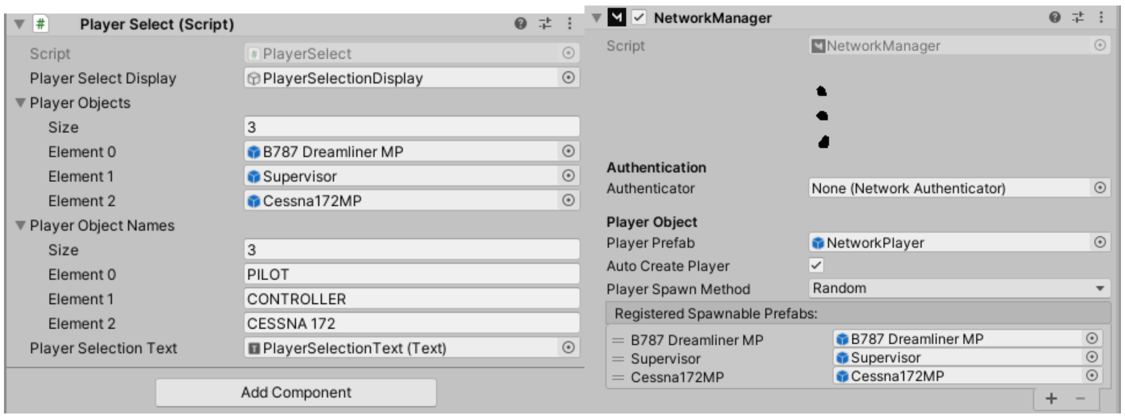


Figure 4.43. Cessna 172 3D model downloaded from the SketchUp 3D Warehouse [40].

The new model was swiftly imported into Unity and its materials were assigned. A script using mouse and keyboard controls to fly the aircraft was procured online. The code is copyrighted under MIT License [41], so it can be used for the purposes of this project. Some parameters like forces and mass needed to be configured but it was an overall simple process.

An initializer script was created and similarly to the TaxiBot and traffic controller network profile, all it does is enable user inputs, camera, and cursor element. The prefab also needs to have a Network Identity and a Network Transform with the Client Authority option enabled.

Afterward, references and its profile name were added to the Player Select panel arrays. Finally, the prefab needs to be registered in the current scene Network Manager



Player Select (Script)

Script	PlayerSelect
Player Select Display	PlayerSelectionDisplay
Player Objects	
Size	3
Element 0	B787 Dreamliner MP
Element 1	Supervisor
Element 2	Cessna172MP
Player Object Names	
Size	3
Element 0	PILOT
Element 1	CONTROLLER
Element 2	CESSNA 172
Player Selection Text	PlayerSelectionText (Text)

Add Component

NetworkManager

Script	NetworkManager
Authentication	
Authenticator	None (Network Authenticator)
Player Object	
Player Prefab	NetworkPlayer
Auto Create Player	✓
Player Spawn Method	Random

Registered Spawnable Prefabs:

B787 Dreamliner MP	B787 Dreamliner MP
Supervisor	Supervisor
Cessna172MP	Cessna172MP

Figure 4.44. Left: Add prefab and name to the profile select panel. Right: Add prefab to the Network Manager registered list.

From the moment the 3D model and the script were found online to the moment the Cessna 172 was flying in the virtual environment, the process only took 35 minutes.

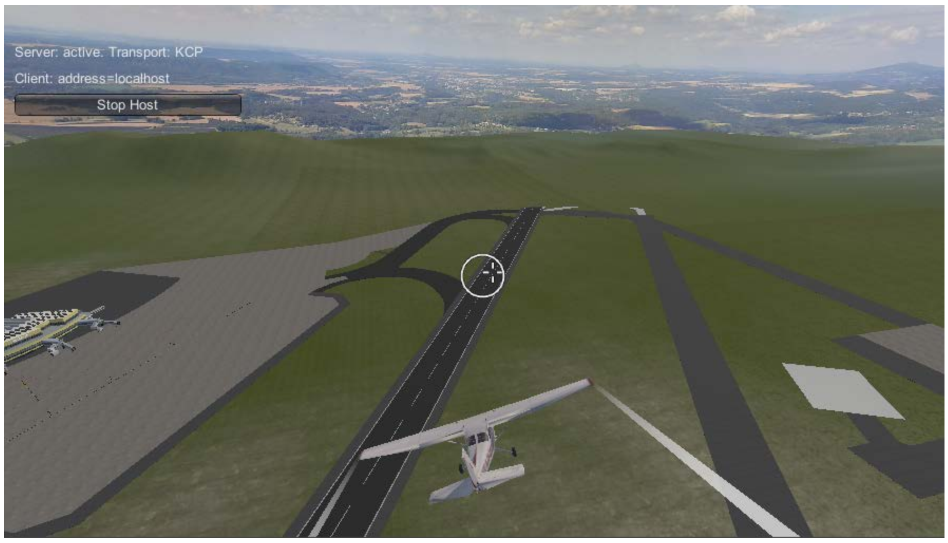


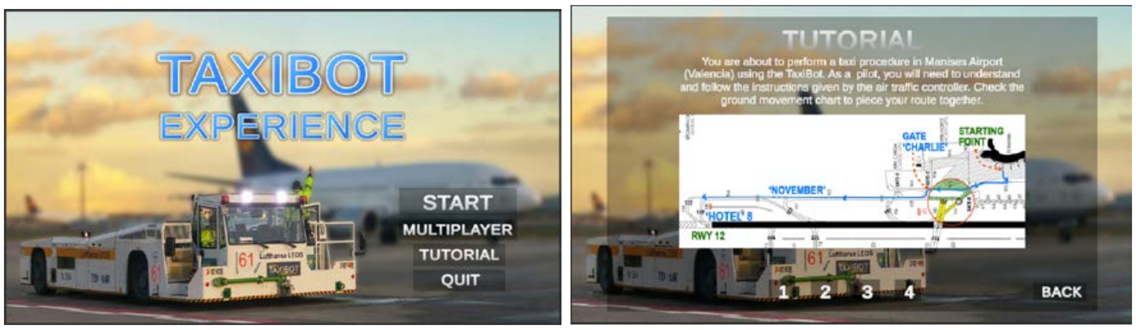
Figure 4.45. Cessna 172 flying over the virtual environment.

4.8. Additional Functionalities

While I do not consider them to relate directly to the objectives, several more functionalities were implemented to improve usability and user experience.

4.8.1. Main Menu

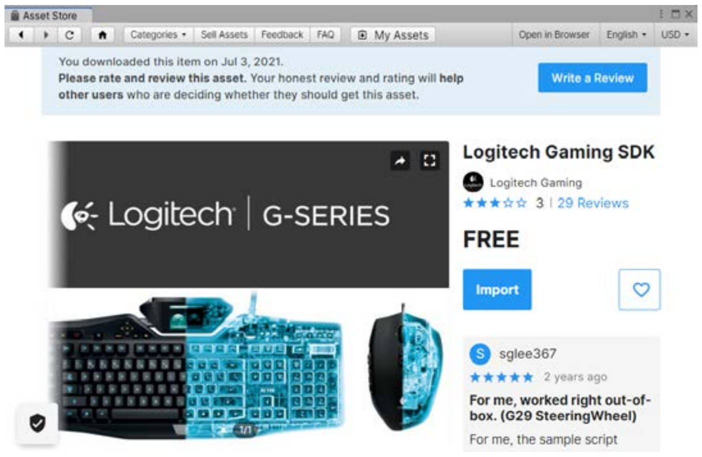
The menu was created rather fast. It consists of a simple canvas with buttons that perform functions like move to a specific scene (in this case, simple environment and multiplayer environment). The Tutorial button enables a new panel and hides the previous one. Each of the four buttons only changes the active image and text instructions.



You are about to perform a taxi procedure in Manises Airport (Valencia) using the TaxiBot. As a pilot, you will need to understand and follow the instructions given by the air traffic controller. Check the ground movement chart to piece your route together.

Figure 4.46. Left: main menu view. Right: First tutorial slide. (Original photo by IAI). https://www.ainonline.com/aviation-news/air-transport/2016-10-10/taxibot-ready-cut-cost-and-carbon-footprint-taxiing

4.8.2. Steering Wheel and Pedals Support



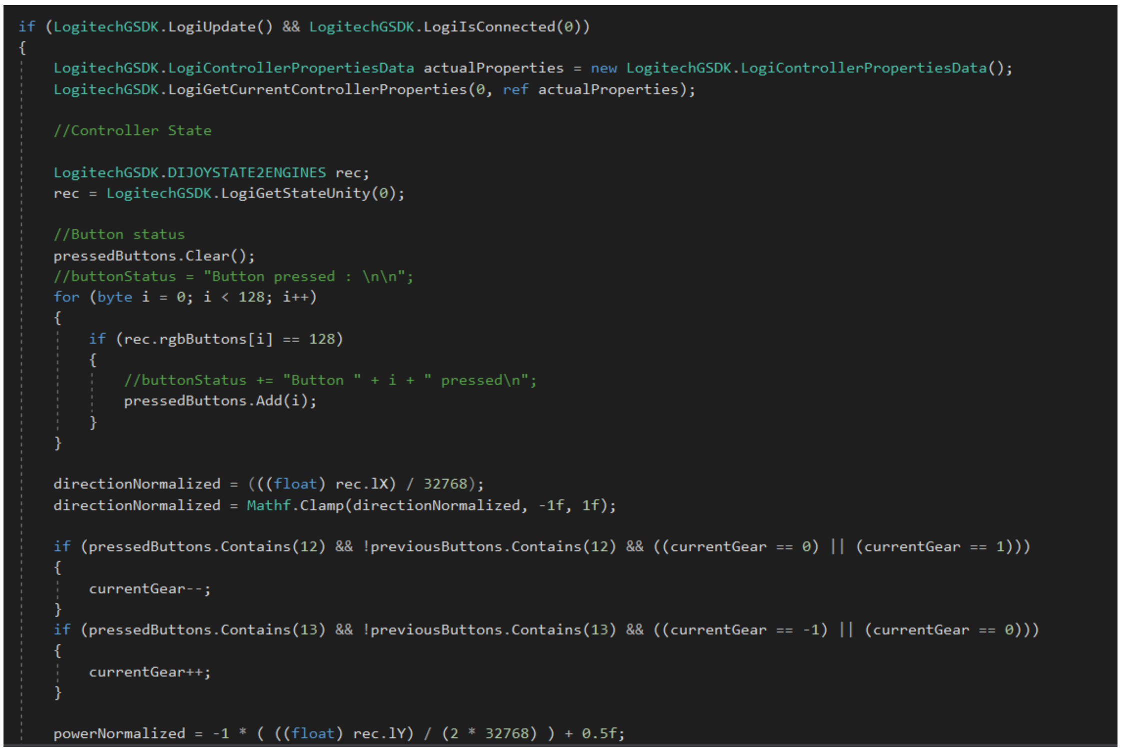
You downloaded this item on Jul 3, 2021.
Please rate and review this asset. Your honest review and rating will **help other users** who are deciding whether they should get this asset.

Logitech Gaming SDK
Logitech Gaming
★★★☆☆ 3 | 29 Reviews
FREE

sglee367
★★★★★ 2 years ago
For me, worked right out-of-box. (G29 SteeringWheel)
For me, the sample script

Figure 4.47. Logitech Software Developer Kit available in the Unity Asset Store. https://assetstore.unity.com/packages/tools/integration/logitech-gaming-sdk-6630

I could get hold of a Logitech Driving Force Pro steering wheel and pedals [42] for free, so I decided to investigate how to approach its implementation. A free Software Developer Kit package to integrate Logitech devices was available from the Unity Asset Store. I simply followed the installation instructions to see working examples. After studying an implementation example, all that was required was to assign the variables defined in Section 4.4 to the correct axes and buttons, as well as create a flag to determine if the inputs should be read from the mouse and keyboard or from a connected steering wheel.



```
if (LogitechGSDK.LogiUpdate() && LogitechGSDK.LogiIsConnected(0))
{
    LogitechGSDK.LogiControllerPropertiesData actualProperties = new LogitechGSDK.LogiControllerPropertiesData();
    LogitechGSDK.LogiGetCurrentControllerProperties(0, ref actualProperties);

    //Controller State

    LogitechGSDK.DIJOYSTATE2ENGINES rec;
    rec = LogitechGSDK.LogiGetStateUnity(0);

    //Button status
    pressedButtons.Clear();
    //buttonStatus = "Button pressed : \n\n";
    for (byte i = 0; i < 128; i++)
    {
        if (rec.rgbButtons[i] == 128)
        {
            //buttonStatus += "Button " + i + " pressed\n";
            pressedButtons.Add(i);
        }
    }

    directionNormalized = (((float) rec.lX) / 32768);
    directionNormalized = Mathf.Clamp(directionNormalized, -1f, 1f);

    if (pressedButtons.Contains(12) && !previousButtons.Contains(12) && ((currentGear == 0) || (currentGear == 1)))
    {
        currentGear--;
    }
    if (pressedButtons.Contains(13) && !previousButtons.Contains(13) && ((currentGear == -1) || (currentGear == 0)))
    {
        currentGear++;
    }

    powerNormalized = -1 * ( ((float) rec.lY) / (2 * 32768) ) + 0.5f;
```

Figure 4.48. Code snippet to map user control variables to encoded values in any Logitech steering wheel.

CHAPTER 5. VALIDATION

During development of the experience, many aspects were being tested continuously to ensure that the features, quality, and performance of the application were appropriate. This aspect drove some decisions beyond the initial design in more than one situation (steering wheel and pedals and networking).

The purpose of the validation section is to analyze the extent to which the objectives have been met. To do so, the experience was tested and evaluated by users of different levels of knowledge on the subject. A survey was designed to obtain mostly qualitative data on the experiences of each user.

The setup consisted of a computer using a steering wheel and pedals for the test user performing the operation playing the role of pilot. Meanwhile, a different computer was connected using the network functionality to play the role of controller monitoring the user. This arrangement allowed me to solve doubts in case the automatic controller messages and chart were insufficient.

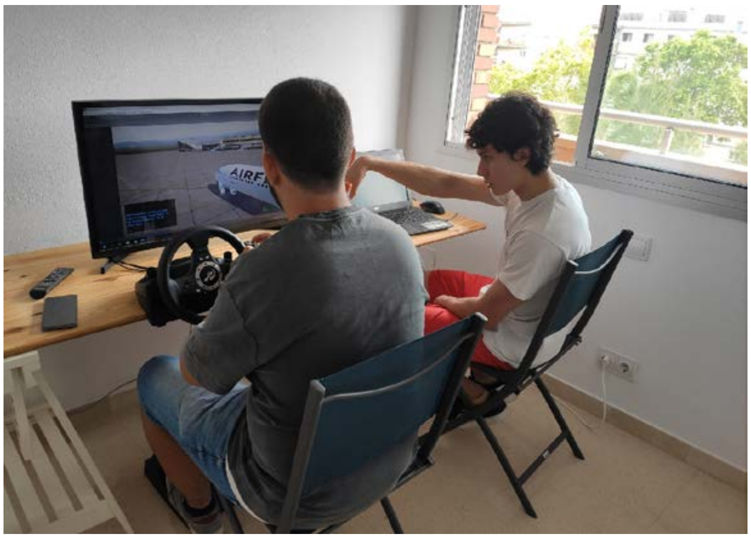


Figure 5.1. Validation session with the roles of test pilot and traffic controller.

The results of the questionnaire are shown below. The percentages are calculated over a total of 16 users who tested the experience and answered the questionnaire minutes later.

- The average time to finish the experience was approximately 8 minutes and 35 seconds. The fastest time was just under 5 minutes, performed by a user on a second attempt.
- 12.5% of users did not complete the experience (two users). One of them explained that the most difficult aspect was relating the position and direction of the aircraft to the provided chart. The other test user explained that it would take too long to attempt the procedure from the start a second time.
- Most of the test users considered they did not have much knowledge on the subject of airport ground movement.

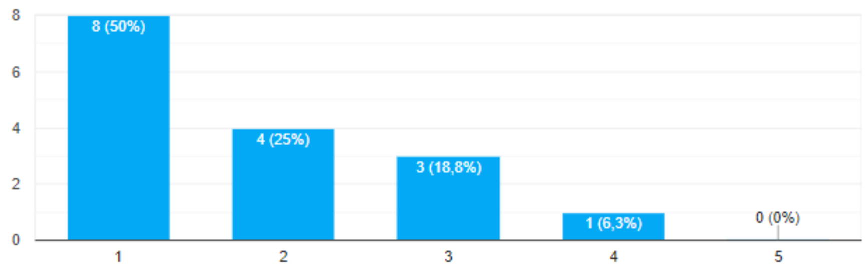


Figure 5.2. Knowledge about the subject prior to the experience. 1 means very unfamiliar, 5 means very knowledgeable.

However, the majority of users stated that they were familiar with videogames or other software that makes use of mouse and keyboard control schemes.

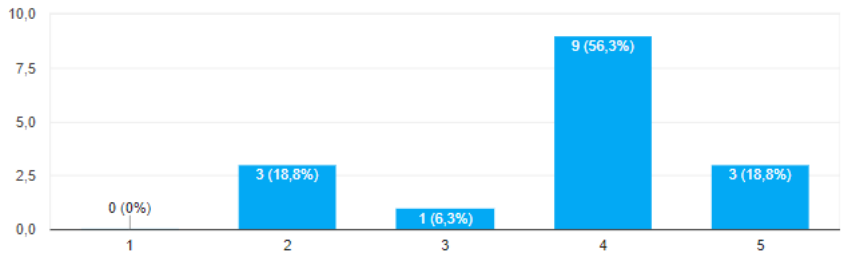


Figure 5.3. Use of mouse and keyboard in videogames or experiences. 1 means very unfamiliar, 5 means they use them often.

- A qualitative assessment of the perceived value of the experience as a learning tool was gathered through the question “**After the experience, do you consider yourself more aware of the aspects that a pilot should keep in mind when undertaking a taxi procedure using the TaxiBot**?”

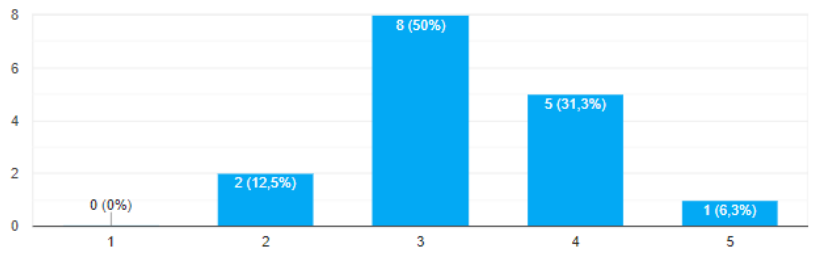


Figure 5.4. Awareness following the experience. 1 means the experience was not very useful, 5 means they are highly aware now.

- As positive feedback, 93.8% of test users stated that they were engaged throughout the experience. The most popular enjoyable aspects were graphical fidelity, physics simulation, and the sense of controlling the TaxiBot and aircraft using the steering wheel and pedals.
- As for constructive feedback, the questionnaire included the question of “**How do you think the experience could be improved or expanded?”**

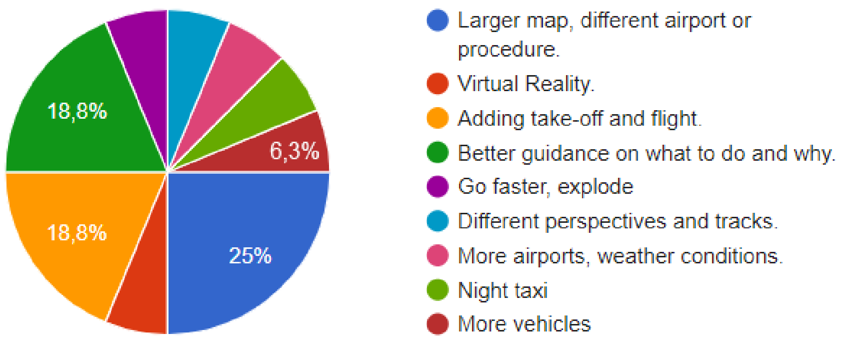


Figure 5.5. Suggestions to expand or enhance the experience.

The most common recommended feature was implementing a different and maybe larger environment. The “More vehicles” suggestion can be grouped together with the “Adding take-off and flight” prompt. It is important to indicate the evaluation session took place before the implementation of the Cessna 172. Better guidance on what to do is a difficult topic because the level of knowledge of the users was widely different.

One of the test users was a pilot who obtained his license (Commercial Pilot License) in 2021. I was very interested in any further insight he could provide and this is an attempt to summarize it.

- He indicated that the dynamics feel well recreated. While he had never used or seen a TaxiBot operation, the acceleration of the aircraft feels very similar to that of a similar weight class aircraft.
- Turning the aircraft is different as the tiller movement is not directly translated to the nose wheel. Instead, the wheels on the TaxiBot turn first and the axle on the docking mechanism follows, which adds some lag when changing directions.
- No additional indications or information were required. Following the automatic controller instructions and making use of the provided chart were sufficient to complete the procedure.
- Night taxi or reduced visibility conditions would be useful additions to the experience.

A thorough evaluation methodology could compare results in two groups of users, one being the experimental group and the other being the control group. The experimental group would test the experience and the control group would receive a theory session with a similar duration covering the same subject. Testing awareness before and after the test would also provide a useful metric to show in the comparison. Such a methodology was not feasible as there were few participants on a single simulator setup. The validation aspect could certainly be expanded upon at a later stage.

CHAPTER 6. CONCLUSIONS AND FUTURE WORK

- A simulator of a TaxiBot and aircraft in a recreated airport has been created and it can be replicated in less than one month time (150 work hours) by a single developer. This estimate considers that most assets like 3D models and textures are pre-made and development is focused on the core simulated experience. The process can be summarized as: (a) identifying the requirements of the simulator (see Sections 3.1 and 3.3); (b) defining the procedures and operations to simulate (see Section 3.4); (c) creating (or importing an existing) the 3D models of the airport, aircraft, TaxiBot, and obstacles (see Section 4.3) considering that they often need to be optimized or corrected; (d) creating a physics system using the tools facilitated by the chosen engine, in this case Unity (see Section 4.4); (e) creating obstacles and dynamic events to complement the experience (see Section 4.5); (f) applying a gamification layer to the training contents (see Section 4.6); (g) data analysis; (h) networking and multiuser support.

- The validation sessions and processes were limited in scope, so it is hard to assess to what extent the experience can be utilized as a training tool. With that said, most test users relayed that the experience made them more aware of the processes involved in a TaxiBot procedure and that the experience was engaging, especially with more than one user.

- Compared to other simulator examples like the KLM VR trainings or Microsoft Flight Simulator, the TaxiBot Experience is lacking in terms of scale and features. However, it is currently the only available software that features the TaxiBot. Similarly, other examples of new technology are straightforward to implement as a new profile (see Section 4.7.3).

- While I had worked with Unity previously, this is the largest project I have done so far. In the Operating Systems subject, along with two other teammates, I made a videogame in Unity with networking using a custom messages solution to resolve position changes. In the TaxiBot experience, Mirror was implemented to resolve the networking services instead. While this made development faster, it is not very complete. The current state of Unity networking is not ideal.

- I was satisfied with the visual fidelity of the recreated environment. During my time at university, I have designed runways, taxiways, and aprons in different subjects. Although they might not have been the most exhaustive projects, they drove me to consult the correct sources for requirements, recommended practices. The airport environment was not complete. While the most relevant signs and markings were present, some were not reproduced. An important judgment I can provide is that generating assets takes the largest part of the development process. Even though most 3D models were downloaded for free, they needed to be modified. If I were to start development on this project again, I would attempt to find alternative sources to get a complete pre-made model of an airport.

- The implementation of steering wheel and pedals to approximate the control scheme of using the tiller to control the nose landing gear and pedals to brake was not given much importance during the definition of requirements and design. However, the use of these peripherals resulted in an immediate increase in control over the vehicles when compared to using mouse and keyboard controls. This aspect me encouraged me to follow a more organic approach to development in which requirements and priorities might by influenced by the continuous validation of the application.

On the subject of future lines of research, there are several options to expand the project:

- Porting the experience to different devices, such as smartphones or virtual reality (VR) headsets. Heavy optimizations would need to be made to allow the application to run on smartphone hardware but it is most likely possible. Implementing VR requires a new input reading system to be developed but it could be ported in a short amount of time.

- Having more options and control over training content. For example, operations at night, during a thunderstorm, or with low visibility would not be hard to implement in terms of the systems that need to be developed. There are surely free or paid packages and learning material about simulating night, rain and fog. However, the airport lights would need to be created and positioned correctly, which is a time consuming endeavor.

- Unexpected threats like malfunctions, unauthorized vehicles on the runway or ATC errors offer more training content. Each of these events would not be very hard to implement in Unity. The system that would take the most effort would likely be a control panel from which the user or a supervisor would configure the events (or a system of random incidents).

- At the moment, there are three roles a user can take, pilot controlling a Boeing 787 through a TaxiBot for taxiing, air or ground traffic controller (although it is not physical character and the control tower is not present), and a Cessna 172. These roles can be used to test different scenarios. For example: training a traffic controller to manage a crowded airspace where every aircraft is controlled by a different user; testing methods to increase capacity, reduce delays and other airport optimizations. The networking might need to be optimized for experiments of this style as it was not stress-tested during development. Additionally, more user profiles for other vehicles or crew members can be implemented very easily.

- Improve the data gathering system that runs during the operation to evaluate how the users are learning and whether the contents or assistance needs to be adapted for each user. Other factors of gamification can also be implemented to test their effect on the value of the experience.

REFERENCES

[1] Jardine C. (2005). *Calculating the Environmental Impact of Aviation Emissions. http://www.hwa.uk.com/site/wp-content/uploads/2020/10/CD-16.6-Calculating-the-Environmental-Impact-of-Aviation-Emissions-June-2005-SSE04.pdf*

[2] Xu et al. (2020). *Quantifying aircraft emissions of Shanghai Pudong International Airport with aircraft ground operational data. https://www.researchgate.net/publication/339047513_Quantifying_aircraft_emissions_of_Shanghai_Pudong_International_Airport_with_aircraft_ground_operational_data*

[3] Re, F. (2012). *Viability and state of the art of environmentally friendly aircraft taxiing systems.* https://ieeexplore.ieee.org/abstract/document/6387462

[4] Giangrande P., Klumner C., Nuzzo S. (2018). *State of the Art of Electric Taxiing Systems.* https://www.researchgate.net/publication/328837869_State_of_the_Art_of_Electric_Taxiing_Systems

[5] Jakub, H. (2014). *Electric taxiing – Taxibot system. https://pdfs.semanticscholar.org/9960/6a86de6b3eed1e88a4f6e105bd6534e78b6c.pdf?_ga=2.136422576.1805683604.1630362241-1365292299.1630362241*

[6] Virgin Atlantic. (April 2007). *Virgin Atlantic Makes Europe's Largest Single Order For Fuel-Efficient Boeing 787 Dreamliner* (Press release). Retrieved 13 August 2011.

[7] Peffers et al. (2006). *The design science research process: A model for producing and presenting information systems research.* https://www.researchgate.net/publication/228650671_The_design_science_research_process_A_model_for_producing_and_presenting_information_systems_research

[8] Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee S. (2007). *A Design Science Research Methodology for Information Systems Research.* https://www.researchgate.net/publication/228650671_The_design_science_research_process_A_model_for_producing_and_presenting_information_systems_research

[9] Walls, J., Widmeyer, G, El Sawy, O. (1992). *Building an Information System Design Theory for Vigilant EIS.* https://pubsonline.informs.org/doi/abs/10.1287/isre.3.1.36

[10] Hevner, A., March, S., Park, J., Ram, S. (2004). *Design Science in Information Systems Research. Vol. 28, No. 1.* https://www.jstor.org/stable/25148625

[11] Bernatzky, T., Baumann, S., Klingauf, U. (2016). *An Interface Concept to Support the Cockpit Crew during Trajectory-Based Dispatch Towing. https://www.icas.org/ICAS_ARCHIVE/ICAS2016/data/papers/2016_0080_paper.pdf*

[12] Sinha S. (May 2021). *Going green: Delhi Airport becomes first globally to have 1000 TaxiBot movements. https://timesofindia.indiatimes.com/business/india-business/going-green-delhi-airport-becomes-first-globally-to-have-1000-taxibot-movements/articleshow/82430183.cms*

[13] Rolfe, J., Caro, P. (1982). *Determining the training effectiveness of flight simulators: Some basic issues and practical developments.* https://www.sciencedirect.com/science/article/abs/pii/0003687082900631

[14] *EASA approves the first Virtual Reality (VR) based Flight Simulation Training Device* https://www.easa.europa.eu/newsroom-and-events/press-releases/easa-approves-first-virtual-reality-vr-based-flight-simulation. Accessed on May 2021.

[15] Tecknotrove System (I) Pvt Ltd. (2017). *Pushback operator first person view*. https://www.youtube.com/watch?v=-TzDYp65zo4

[16] AIRFRANCE KLM-XR Center of Excellence. (2021). Pushback training. https://www.youtube.com/watch?v=C6EiRQjvZjo

[17] KLM News*. (2020). KLM Cityhopper introduces Virtual Reality training for pilots.* https://news.klm.com/klm-cityhopper-introduces-virtual-reality-training-for-pilots/

[18] 320 Sim Pilot. (2020). *Real Airbus Pilot Taxi Tutorial.* https://www.youtube.com/watch?v=fLQgAN79YqE

[19] Kiili, K. (2004). *Digital game-based learning: Towards an experiential gaming model.* http://www.savie.ca/SAGE/Articles/940_300027-KIILI-2005.pdf

[20] Electromobility at Frankfurt Airport. (2015). *Hybrid motor silences jets: the TaxiBot.* http://www.e-port-an.com/project/our-electric-vehicles/taxibot.html

[21] The Little Big Tug Company. (2007). *T80 General Specifications.* http://www.littlebigtugcompany.co.uk/specs.htm

[22] International Civil Aviation Organization (ICAO). (2014). *Annex 14 to the Convention on International Civil Aviation.* https://www.pilot18.com/wp-content/uploads/2017/10/Pilot18.com-ICAO-Annex-14-Volume-1-7th-Edition-2016.pdf

[23] Federal Aviation Administration. (2020). *Standards for Airport Markings Advisory Circular.* https://www.faa.gov/documentLibrary/media/Advisory_Circular/150-5340-1M-Chg-1-Airport-Markings.pdf

[24] Mazareanu E. (2021). *Volume of passenger traffic at the Valencia airport from 2005 to 2019.* https://www.statista.com/statistics/775775/traffic-from-passengers-in-he-airport-from-valencia/

[25] AIP España. Plano de Aeródromo para Movimientos en Tierra-OACI. https://aip.enaire.es/AIP/contenido_AIP/AD/AD2/LEVC/LE_AD_2_LEVC_GMC_1_en.pdf

[26] The Boeing Company. (2005). *Boeing 737 Flight Crew Training Manual.* https://aviation-is.better-than.tv/B737NG_FCTM_(31-10-05).pdf

[27] International Civil Aviation Organization (ICAO). (2005). *Annex 2 to the Convention on International Civil Aviation. Rules of the Air.* https://www.icao.int/Meetings/anconf12/Document%20Archive/an02_cons%5B1%5D.pdf

[28] Milo1002 (2017). *Komatsu WZ4000 TL206.* https://3dwarehouse.sketchup.com/model/59580aea-835f-458b-bd1f-f0334c9ab538/The-bales-tractor%E3%83%88%E3%83%BC%E3%83%90%E3%83%BC%E3%83%AC%E3%82%B9%E3%83%88%E3%83%A9%E3%82%AF%E3%82%BF%E3%83%BC%EF%BC%89

[29] Rocha, M. (2016). *Air France Boeing 787-9 DreamLiner [F-HRBA].* https://3dwarehouse.sketchup.com/model/4cb927d5-d250-422c-aa2f-26aca255db2a/Air-France-Boeing-787-9-DreamLiner-F-HRBA-ver-2016

[30] Comugi. (2019). *BOEING787 Cockpit 787.* https://3dwarehouse.sketchup.com/model/66993df7-3703-4095-aebb-21ff9d472bab/BOEING787-Cockpit-%EF%BD%9E%E3%83%9C%E3%83%BC%E3%82%A4%E3%83%B3%E3%82%B0787%E3%81%AE%E3%82%B3%E3%82%AF%E3%83%94%E3%83%83%E3%83%88%EF%BD%9E

[31] Photo by Grid9. (2016). *Boeing 787 Dreamliner-Cab.* https://www.360cities.net/image/boeing-787-dreamliner-cab

[32] R. Greg. (2017). *3D Airport with marking and signs.* https://3dwarehouse.sketchup.com/model/a7240881-6a5b-4d30-abfb-1f4437d9e2f2/3D-Airport-with-marking-and-signs

[33] Gegge. (2015). *Aeropuerto Manises.* https://3dwarehouse.sketchup.com/model/u63d05123-54c1-4f05-b759-984a410b6e76/Aerpuerto-Manises

[34] Lufthansa LEOS. *Taxiing without engines running.* http://www.lufthansa-leos.com/documents/438370/444077/2013_10_TaxiBot.pdf/40fc4f7e-c10a-4ca6-aa0a-9337d2f922f8

[35] The Boeing Company. *787 Airplane Characteristics for Airport Planning. https://www.boeing.com/assets/pdf/commercial/airports/acaps/787.pdf*

[36] Budriz, (2014). Airport Fire Engine. https://3dwarehouse.sketchup.com/model/bf6cce269070af48ba929c0372e76921/Airport-Fire-Engine

[37] International Civil Aviation Organization (ICAO). (2004). *Advanced Surface Movement Guidance and Control Systems.* https://www.icao.int/Meetings/anconf12/Document%20Archive/9830_cons_en%5B1%5D.pdf

[38] Yang, L., Yin, S., Han, H., Haddad, J., Hu, M. (2017). *Fundamental Diagrams of Airport Surface Traffic: Models and Applications.* https://www.researchgate.net/publication/320939098_Fundamental_diagrams_of_airport_surface_traffic_Models_and_applications

[39] H. (2014). KLM Cityhopper Embraer 190. https://3dwarehouse.sketchup.com/model/bae04b58d06edd55324c65994a923d3f/KLM-Cityhopper-Embraer-190

[40] Szilárd, K. (2014). Cessna 172. https://3dwarehouse.sketchup.com/model/bb0795effdb85b13e87d121b5fbf21f0/cessna-172

[41] *Hernandez, B. (2019). Mouse Flight v1.3.* https://github.com/brihernandez/MouseFlight

[42] Logitech 963293-0403 - *Driving Force Pro Wheel Installation Manual.* (2004). https://www.manualslib.com/manual/442879/Logitech-963293-0403-Driving-Force-Pro-Wheel.html#manual

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

APPENDIX

Title: Simulation of a TaxiBot Operation for First Person Training

Author: Gabriel Sebastián Font Rojas

Director: Jordi Pons Prats

Codirector: Francisco Javier Mora Serrano

Date: July 2021

A. TaxiBot Experience Survey

1- What was the time you required to finish the experience? (Please answer "not finished" if that is the case).

1a- (Optional) If you could not complete the procedure, what was the main difficult aspect?

a) Understanding ATC instructions.
b) Relating aircraft position and direction to the provided chart.
c) Not knowing signs and markings.
d) Other (short answer)

2- After the experience, do you consider yourself more aware of the aspects that a pilot should keep in mind when undertaking a taxi procedure using the TaxiBot? (1 to 5)

3- Were you engaged throughout the experience? Any particular enjoyable aspect?

4- How do you think the experience could be improved or expanded?

a) Larger map, different airport or procedure.
b) Virtual Reality.
c) Adding take-off and flight.
d) Better guidance on what to do and why.
e) Other (short answer)

5- How would you rate your knowledge on the subject of airport ground movement prior to the experience? (1 to 5)

6- Are you familiar with videogames or other software that make use of mouse and keyboard control schemes to control a vehicle or character? (1 to 5)