REPUBLIC OF TURKİYE

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Electric and Computer Engineering Master Program

THE ROLE OF MACHINE LEARNING IN ENCHANCING REALISM IN UNREAL ENGINE GAMES

İbrahim Berk ADIGÜZEL

Master's Thesis

Supervisor

Asst. Prof. Dr. Abdullahi Abdu IBRAHIM

İstanbul, 2025

THE ROLE OF MACHINE LEARNING IN ENCHANCING REALISM IN UNREAL ENGINE GAMES

İbrahim Berk ADIGÜZEL

Electric and Computer Engineering

Master's Thesis

ALTINBAŞ UNIVERSITY

2025

The thesis titled The Role of Machine Learning in Enhancing Realism in Unreal Engine Games prepared by İbrahim Berk ADIGÜZEL and submitted on 12.05.2025 has been **accepted unanimously of votes** for the degree of Master of Science in Electric and Computer Engineering.

Dr. Abdullahi Abdu IBRAHIM
Supervisor

Thesis Defense Committee Members:

Doç. Dr. Sefer KURNAZ	Department of Computer Engineering, Altınbaş University	
Dr. Abullahi Abdu IBRAHIM	Department of Computer Engineering, Altınbaş University	
Dr. Tarık Abed MOHAMMED	Department of Computer Science, Kirkuk University	

I hereby declare that this thesis meets all format and submission requirements for a Electric and Computer Engineering Master thesis.

I hereby declare that all information/data presented in this graduation project has been obtained in full accordance with academic rules and ethical conduct. I also declare all unoriginal materials and conclusions have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned rules and conduct.

İbrahim Berk ADIGÜZEL

Signature

ABSTRACT

THE ROLE OF MACHINE LEARNING IN ENCHANCING REALISM IN UNREAL ENGINE GAMES

ADIGÜZEL, İbrahim Berk

M.Sc, Electric and Computer Engeneering, Altınbaş University,

Supervisor: Asst. Prof. Dr. Abdullahi Abdu IBRAHIM

Date: 5/2025

Pages: 55

The way machine learning (ML) is applied in game production process, especially through development IDE such as the Unreal Engine 5, paves innovative methods for developing significant immersive and authentic digital experiences. This thesis inspects the relational intersection of machine learning and game development process with examination of spatial how machine learning methods lends to gameplay mechanics, player interactions, and simulates physics-based movements. Additionally this thesis also contains a case study about movement mechanics of wind turbines in a simulated environment, using Unreal Engine's blueprint scripting way of programming but also hardcode in C++ to make blueprints run into Unreal Engine's blueprint scripting. Used coded simulation using real time Machine Learning algorithm on C++ to calculate the turbine's body and blade movement and simulated it with real world factors such as wind source, distance and speed. This calculation impacts the amount of power that is produced by the wind turbine.

This tells that applying these methods and technologies with all of ML algorithms, Unreal Engine 5's in engine physics simulations and time may enhance visual environments, fidelity and deep game environment mechanics. This work demonstrates that ML based simulations can be done in real time in a way that reflects player behaviors and environmental influences in the game thereby improving the gameplay experience.

The game utilizes AI-driven simulations that respond to player behavior, as well as dynamic interactions with environmental changes, contributing to greater immersion.

Finally, this research explores finds and includes the challenges for importing real time Machine Learning algorithms into games, uncovering the potential resource usage and performance optimizations, leads and provides insights into to potential future of advanced ML and AI powered game designs, mechanics and coding.

This thesis and project provides an improvement in knowledge on using the ML and AI algorithms into the gaming part of life and innovative thinking of adapting these technologies for realism techniques to extend the limit of your in-game interactions and not Static experiences in virtual fronts.

Keywords: Machine Learning, Unreal Engine, Game Development, Realism, Artificial Intelligence, Immersive Experiences.

TABLE OF CONTENTS

Pages

LIST OF FIGURES

ABBREVIATIONS

AI	:	Artificial Intelligence
API	:	Application Programming Interface
Blueprint	:	Unreal Engine’s Visual Scripting System
C++	:	C++ Programming Language
FPS	:	First-Person Shooter
IDE	:	Integrated Development Environment
KPI	:	Key Performance Indicator
ML	:	Machine Learning
RPG	:	Role-Playing Game
Unreal Engine	:	Unreal Engine Game Development Environment

1. INTRODUCTION

Machine learning's convergence with game development has recently uncovered revolutionary potential in infusing realism into virtual landscapes. Unreal Engine has limited rendering ability but is an incredibly versatile modular engine, so it has become one of the preferred platforms for developers to stretch the limits of how real a game can appear. Machine learning (ML) — in particular, its unparalleled capacity to process massive amounts of data to create real-time predictions or results — is emerging as a transformative solution for modeling complex dynamic systems that are challenging or prohibitively expensive to simulate using traditional approaches.

This thesis conducts research on methods of embedding machine learning techniques into Unreal Engine 5 and how this technology can be utilized for the improvement of gameplay mechanics, environment interaction, and physics-based simulations. Through the case study of a wind turbine simulation—developed using a combination of C++ and Unreal Engine's blueprint visual scripting system—we illustrate how environmental factors, like wind direction and speed, can be modeled more accurately than with traditional simulation methods using ML-based algorithms. Real-time outcomes simulating blade movement, rotational speed, and power generation are produced based on real-world physics and dynamic weather conditions.

Not only is this project a prototyping tool using Unreal Engine's Blueprint system to rapidly iterate on plans and design, but blueprint also plays a key role for testing AI driven behaviours with real-time interaction. This means that combined with C++, which provides lower-level control and efficiency, ensures real-time particle-based AI interactions that feel smooth, fluid, dynamic, and visceral. Machine learning integration will enhance much more than the graphical appearance of game environments, advancing the intricate functionality of the game physics engine and player interaction, fostering greater levels of simulation across a broader spectrum of natural phenomena, object movement and transformations, and where applicable player invented scenarios.

Here, the work summarizes how machine learning models have been adapted to see gains not only in visual fidelity but also gameplay goals. Through multi-faceted applications ranging from environmental simulations to machine learning based AI decision planning,

this thesis showcases how extended machine learning technologies, in conjunction with state of the art game engines such as Unreal Engine 5, can change the very nature of designing and playing video games. They can now design far more responsive and reactive environments tailored to players' behavior and real-time in-game events.

We then proceed to discuss the broader consequences of utilizing machine learning methods in game development -- as real-time AI implementations can lead to specific performance optimization and resource management challenges. The methods will vary from C++ implementations of various machine learning models all the way out to designing and visualizing the models inside Unreal Engine's Blueprint system. So while at first glance machine learning may not be something you think about in terms of game worlds, the combination of these two approaches demonstrates the power of Unreal Engine as a platform for the implementation of AI principles to bring unreal and interactive mechanics to life — and we're excited to see where Epic takes the integration with machine learning and the future of machine learning in games.

By providing an example of relevant real-time machine learning techniques, towards improved realism and responsiveness of game environments, this project seeks to contribute to the growing pool of knowledge on the relationship of AI & gaming. Hence, this study shows the capabilities of AI-based simulation in breaking new ground with regards to real-time gameplay and provides implications for the future of game development objectives and artificial intelligence-augmented virtual world design.

2. GENERAL INFORMATION

2.1 BACKGROUND

Machine learning and artificial intelligence (AI) have changed the landscape forever when it comes to realism in game development. With video games growing from simple mechanics to extremely complex systems able to run real physics, environment and interaction simulations, the need for more sophisticated tools has grown dramatically. These tools are necessary for controlling and simulating the ever more sophisticated behaviors of in-game objects, characters, and environments in real-time.

There are a myriad of reasons for chasing realism within games, but one of the big reasons why we chase realism in games is due to realistic disasters. Realism, at its basis, increases immersion — the ability to believe in and identify with the virtual world the user is engaging in. A more realistic game environment makes the player's experience increasingly immersive and alluring. Now make use of the power of machine learning to generate environments that can mutate, respond to, and react to player actions and events in real-time in intelligent ways, modelling the unpredictable nature and fluidity of real-world physics. Yes, this revolutionary technology is pushing the boundaries of the game developement field and allowing for more detailed story lines, complex gameplay mechanics and hyper realistic simulation that make a fine line between alternate and real world.

2.1.1 Evolution of Unreal Engine

Unreal Engine has defined game development since its launch in the late 1990s, starting as an FPS game tool and progressively evolving to support nearly every game genre there is, from open-world RPGs to sports games and simulations. What made Unreal Engine stand out was that it had a special knack for balancing both artistic expression and technical performance. It became the preferred engine for independent developers, AAA studios and educators alike, because of its versatility, extensibility and ability to produce games at a high level of visual fidelity.

Figure 2.1 : Unreal Engine Logos for Each Version.

It was with the launch of Unreal Engine 5 that Epic Games had introduced ligature bending technologies such as Nanite & Lumen. Nanite enables the creation of ultra-detailed virtual worlds with billions of polys and the ability to render that geometric complexity with unmatched efficiency. In contrast, Lumen provides fully dynamic global illumination, bringing an entirely new dimension to lighting situations that involve reaction to the environment. When coupled with ML models/algorithms, these capabilities leveraging machine learning can enable developers to construct complex worlds that adapt to the user's behavior and mirror the complexity of reality. In addition, Unreal Engine's C++ and Blueprint scripting system is perfect for not just integrating machine learning algorithms seamlessly but also for ensuring it supplies hyper-realistic and performance-oriented gameplay mechanics.

2.1.2 Role of Machine Learning in Realism

Machine learning is a revolutionising technology used in the gaming industry, impacting not only AI-character behaviours but also the simulation of real-world physics in a realistic manner. Using data prediction and also simulation techniques, machine learning can determine what objects, environments, and other in-game entities might realistically do to each others, leading to photo-realistic game movements. That ability is important for simulating elements of nature — wind, and water, and fire — which are notoriously vexing to model by normal procedural means.

The motivation behind the thesis is an example of use of machine learning for prediction in a wind turbine simulation with usage Unreal Engine, where algorithms model wind speed, a degrees of rotation of blades and produced power depending on environment. By training machine learning models using real-world data, this method can accurately simulate how a wind turbine would respond in the real world to a range of wind speed conditions. Showing

a physics case study through this advanced tech is a testament to AI-powered systems that could power the future of realism for physics engines and how gameplay could evolve towards more complex types (pooled energy in dynamic objects, etc.) interactions.

2.1.3 Unreal Engine Blueprint and C++ Integration

Unreal Engine has one of the biggest and most marketable features a dual programming model with its Blueprint visual scripting system and native C++ support. They're especially useful for rapid prototyping, enabling developers to teach game mechanics and AI behaviours without writing reams of code. This accelerates development which is particularly beneficial during the early phases of gameplay system design. Unreal Engine is also a powerful platform for developing games, especially for more complex and performance-critical, real-time simulation with high-fidelity graphics, which is being entirely supported by writing programs with pure C++ code.

Blueprints and C++ are both integral portions of this project. Blueprints are what you use for things that require high-level logic, things in gameplay interaction, and visual effects; C++ is what you use for the heavy lifting of how the blades of a wind turbine are moving and what kind of power that's going to give. This gives the project the benefit of the fast iteration cycle that Blueprints provide, whilst retaining the performance and scalability options granted by C++. This allows for the integration of machine learning models without sacrificing real-time responsiveness, or visual fidelity of the game.

2.1.4 Wind Turbine Simulation in Unreal Engine

The seamless experience gained as a result of applying machine learning techniques to physics-based simulations is further elaborated in relation to the wind turbine simulation developed in this thesis. This simulation continually updates wind speed, wind direction and other environmental parameters and the subsequent changes are applied in real-time to the behaviour of the turbine through ML models. This means manipulating windmill physical properties, like how fast each of them translates, spins, generate power, how much they follow real-world physics, etc., such that the simulation is convincing to the player.

The Euclidean-distance function at the heart of its system, written in C++, calculates distance for important wind turbine components. Using these data, the speed and movement of the blades, as well as all other physics, are real-time calculated to deliver a smooth and

responsive simulation experience. Machine learning models may recommend any of the above tasks in this instance, and perform this way so that the framework can respond to alterations in the environment and adapt to them, creating the appearance of a more realistic game.

2.1.5 Importance of Real-Time Simulations

Real-time simulations lie at the core of contemporary game development, where fast and accurate response to player actions are now an expectation. Whether telling the story through AI-driven characters or physical assets, a real-time capability to simulate real-world phenomena is crucial to maintain immersion. This is where machine learning models shine, allowing games to adjust in real-time to player inputs and changes in the environment.

They're trained on data on everything from the shape and strength of wind turbine blades (they have to move immediately as wind speed changes, to give players something that visually and mechanically feels real) to how a character interacts with a shadowy corridor. This is critical for providing an immersive experience, where players can immediately observe the effects of their actions on the game world. Developers can make physics systems more believable and make the world feel more alive and emotionally engaging by making it reactive, changing in response to the player's presence and decisions, this is why they are exploring the possibility of using machine learning for such scenarios.

2.1.6 Challenges and Future Directions

Although the machine learning integration into Unreal Engine provides a lot of benefits, it brings very big challenges as well. This is an obvious concern as many applied machine learning models are high-performance models that tend to deliver performances measured on an order of magnitude of a few micro-seconds, which in the case of many interactive environments means being able to execute a simulation of hundreds of interactive objects happening at the same time. These computations can be particularly resource-heavy, requiring careful optimization to preserve real-time performance at the cost of realism.

The future: As ML models improve, their game role will just expand. These technologies will help in advancing the algorithms on a bigger scale and help you create realistic environments in future. With the latest tools, Developers can build intelligent, adaptive worlds where AI informs everything from character behavior to physical interaction,

environmental simulation, and dynamic narration. In this thesis we take the first step towards these possibilities by showing how machine learning can be introduced to improve the realism of games in the realm of Unreal Engine5.

3. METHODOLOGY

This Thesis methodology chapter highlights the procedures and methods of using the machine learning in Unreal Engine 5 environment to improved the simulation of wind turbine realism. Not only does it use a range of Blueprint visual scripting and C++ code to achieve real-time, physics-based simulations, it integrates the process through different parts of development from information gathering and approaches to physical application.

First off, the process of development itself was structured to be modular, allowing for the development and testing of each component of the wind turbine simulation independently. And the simulation had to respond dynamically to real-world environmental inputs like wind speed and direction, meaning machine-learning models and the physics engine in Unreal Engine needed to work together. The early stage of development was working on a simple turbine model and incorporating Unreal Engine's native physics to work for basic rotation. As the project grew, more complicated physics models—like the variable pitch of blade and rotational speed based on particular wind conditions—were incorporated.

Data collection played a crucial part of the process. This required terabytes of actual data to convince algorithms that simulated behaviour matched that of wind turbines in real life. In a first step, all the data obtained for wind speed, wind direction, air density, and turbine power generation is utilized in different sets of climatic conditions. These datasets have been preprocessed and cleaned so that they could be used in machine learning models. The machine learning models predicted accurately how turbines behaved under variable environmental conditions by normalizing and eliminating outlier data.

The implementation strategies were designed to provide a balance between the system performance and realism. The main takeaway was ensuring that the machine learning models could run in real-time without introducing a substantial computational overhead. In response to this, the machine learning algorithms were optimized so they could run efficiently when embedded in the Unreal Engine's framework. Turbine responses to environmental changes, such as blade angle and rotating speed alterations in accordance with wind speed adjustments, were predicted using supervised learning models.

Our other key strategy was leveraging both systems through Blueprint visual scripting and alongside C++. The simulation data and high level of the gameplay interactions were

written in blueprints, which made changing the simulation very easy and allowed for quick iterations during development. C++, in contrast, was used for performance-sensitive operations like calculating the wind force on turbine blades and dynamically adjusting the rotation speeds of blades. The entire heavy computational part of the application was offloaded to C++, so the project it was able to maintain performance at a high level and to keep the turbine simulation very realistic.

One of the hallmarks of this mode of operation is the seamless integration in real-time between trained machine learning models and Unreal Engine's physics engine. By continuously feeding it environmental data (like wind speed and direction), the machine learning model could adjust the turbine's behavior in real time to be more reflective of current activity. It meant building a strong information exchange between the machine learning model and the physics-based simulation, such that changes in the environment immediately translated to changes in the turbine's rotation and blade pitch, resulting in alterations to power generation.

The unified between Blueprint and C++ also enabled a very smooth development workflow. Whereas C++ was responsible for performing complex calculations behind the scenes, Blueprints granted a visual interface to alter the simulation and test different environmental conditions. For instance, in Blueprints, the wind speed and direction needed to be changed for quick visual feedback, but the real physics calculations were run in C++, as the game played seriously and intensively.

Closing remark This methodology shows the profound steps, techniques and workflow that enable the implementation of ML into Unreal Engine 5, with a specific focus on the task of building a realistic wind turbine simulation. This project highlights how creative use of Blueprint visual scripting and C++ code, as well as machine learning algorithms optimized for real-time implementation, can deliver a good trade-off between realism and performance through a physics-based simulation that reacts to changes in the environment.

3.1 PROJECT SETUP AND TOOLS

This project provides a successful implementation of machine learning coordination in Unreal Engine 5 using a wide range of industry-standard tools and software environments. All these tools had their own role in making sure the projects goal of Realism, efficiency

and real-time simulation was met. The main tools involved in this project are the following (not limited to):

Unreal Engine 5: Overall, Unreal Engine 5 utilized as the core platform for both simulation and rendering. Powered by its state-of-the-art architecture — with Nanite streaming worlds full of ultra-detailed geometry straight to the GPU, and Lumen providing real-time global illumination for stunningly realistic scenes — an incredible visual experience was now possible. Unreal Engine's Blueprint system was also extensively used for visual script, allowing for fast prototyping and managing game logic without having to write conventional code. Native C++ was used for high-performance tasks to optimize computational efficiency. For instance, Unreal Engine 5's strong real-time simulation capabilities made it a suitable framework to use for the wind turbine engine model simulation in this thesis.

C++: The strongest of the machine learning integration with the project was through the use of C++. Color critical to the wind turbine's operation — blade rotation, pitch angles, power generation — hovered in real time as a function of environmental inputs such as wind speed and direction. These machine learning models were later developed using C++, allowing the algorithms sufficient strength and flexibility to handle thousands of datasets. The project's performance requirements led to the choice of C++, which combined with its algorithmic sophistication enabled it to run complex simulations without losing real-time responsiveness.

Blueprint Visual Scripting — Unreal Engine features Blueprints, a high]-level visual scripting interface that makes developing game mechanics, AI behaviors, and environmental interactions easier. We developed most of the game logic in blueprints, allowing for some very fast iterations and prototyping. Going for a visual style with Blueprints allowed for rapid changes in gameplay and interactions without writing huge amounts of code. Such flexibility was instrumental when exploring the impact of machine learning models on turbine behavior and how real-time wind dynamics were visualized. In addition, Blueprints enabled rapid prototyping and visual development, combined with the ability to optimize sections of the simulation using C++ for performance-critical code, allowing the turbine to be fully rendered and simulated at runtime.

OpenCV: Relating very closely to image processing, OpenCV was needed for parts of the project since we needed to do real-time image analysis for implementing future machine learning models that could potentially use image data. The image processing aspects of OpenCV were not directly leveraged in the wind turbine simulation, but this is an area in which future developments could incorporate image recognition and environmental data from visual inputs into machine learning processes. OpenCV has such a rich set of libraries for computer vision and machine learning which makes working with it versatile enough to be built upon for improving realism in environmental simulations.

This project primarily used Microsoft Visual Studio as its Integrated Development Environment (IDE), where the C++ scripts that composed the machine learning models and physics calculations were written, debugged, and compiled. After researching potential tools for the job, we settled on Visual Studio as the go-to engine for managing the complexity of our project's codebase, thanks to its powerful debugging tools and support of Unreal Engine development. The integration of Unreal Engine with C++ scripting enables a strong and stable platform that facilitates iterative development game developers can introduce gameplay components, and test them accordingly. Visual studio also enriches this powerful integration experience making it a comprehensive integrated environment with lots of tools available for debugging, code navigation, and real-time error detection further unifying the workflow.

From the application of Unreal Engine rendering stream, C++ script, and integration with Visual Studio, this combination provided excellent optimization, realism, and flexibility to develop the wind turbine simulation. The effort visually unifies the technologies and allowed the simulation to respond dynamically and accurately to real-time situational changes like wind direction and wind speed. On the other hand, cutting-edge graphics power following Unreal Engine kept a high level of visual quality without sacrificing efficient simulation. Visual Studio's functionality enabled a swift workflow, encouraging quick iterations, easy debugging, and easy testing, which resulted in a much better product.

3.2 WIND TURBINE SIMULATION FRAMEWORK

The wind turbine simulation also worked well to demonstrate how machine learning improves object behavior realism by providing a mechanism through which the environment

and simulation object can interact dynamically. This project illustrates the possibilities of AI in complementing the Inspirations of traditional/usual interactions with the analysts simulating interactions leading to a greater degree of interactivity and involvement in Unreal Engine 5. Notably, the turbine's blades change their angle as they rotate, using calculations of the wind's speed, direction and other environmental factors to direct their movements.

The wind turbine simulation was organized into two main aspects that follow different-ing theories of physics-based simulation and machine learning:

3.2.1 Distance Calculation

You are responsible for the core calculation of distance between different parts of the systems. This is the basis of how turbine responds to environmental objects, wind sources, etc. In particular, the distance of the turbine with respect to these objects dictates the wind intensity reaching the turbine, which would modify the movement and rotation of blades.

In order for a turbine to react with realistic physics based interactions, this computation must happen, enabling the turbine to behave as it should with proper physics. To do so, C++ implementations of Euclidean Distance functions that would allow the simulation to compute the distance between objects in 3D space on the fly were used. The turbine blades' rotational speed is predicted by the machine learning models using this distance data, in addition to wind speed and directional inputs.

By continuously calculating in real-time the distances between the turbine's components and various wind sources located outside the turbine itself, the system effectively carries out the physical concept that the power of the wind diminishes as the distance from the source increases. As a result, this directly impacts the turbine operational efficiency components positioned further from the source of the wind collect less force. The simulation varies the behavior of the turbine dynamically by location and orientation in relation to wind intensity and direction for a great deal of realism. Its data is based on how real-world turbines react to environmental fluctuations, which is helpful for learning about how to make them more efficient energy-wise and mechanically. It is built to realistically simulate the spatial and directionality of wind energy capture, making it suitable for research, design testing and educational demonstrations in the renewable energy sector.

3.2.2 Blade Speed and Power Generation

After that, the second most stressful part of wind turbine simulation is what would be the rotational speed of blades as well as the power output at different times depending on wind data. This is where the power of machine learning comes into play. So machine learning models based instead of a fixed rule set or naive physics model refine the turbine behavior.

This project implements machine learning models trained on environmental inputs to predict the optimal blade speed and operating power. These data inputs include wind speed, direction, temperature, air pressure, humidity levels, and the orientation of the wind turbine. The models learn to make data-driven decisions based on this broad spectrum of input variables in order to optimize energy production, often in real time. This provides increased flexibility and realism in the simulation, allowing it to more accurately reflect turbine performance in real-world environmental settings.

Wind Speed: When wind goes faster, so do the blades; they spin faster to collect more kinetic energy and generate more electricity. But this speed is very tightly controlled. The blades have an upper limit on how fast they can safely spin before the mechanical stresses on the turbine's components become so extreme that damage occurs or the lifespan of the turbine is reduced. The system identifies these limits and modifies blade speed in real-time to maintain operation within safe constraints of operation, thus ensuring both efficiency and structural integrity.

The direction of the wind: How the wind hits the blades influences how efficiently power is generated. The machine learning models assist in the optimization of the blades pitch to maximize the amount of energy captured from the wind.

Environmental conditions: The efficiency of the turbine can also be impacted by things like air pressure, temperature, and altitude, all of which are included in the machine learning model's predictions.

Using these inputs, the ML models compute the optimal blade speed dynamically in response to changes in wind conditions. The turbine generator simulation is also designed to ensure that power generation changes in a way that is consistent with real-life turbine dynamics. Reducing turbine speed can occur depending on how much a given hazard is present, so for instance if wind speed is reduced the turbine power will not stop but still

generate at a minimal level. On the other hand, under higher wind speeds the blades will rotate faster, but the system will limit the upper speed to avoid damaging the turbine.

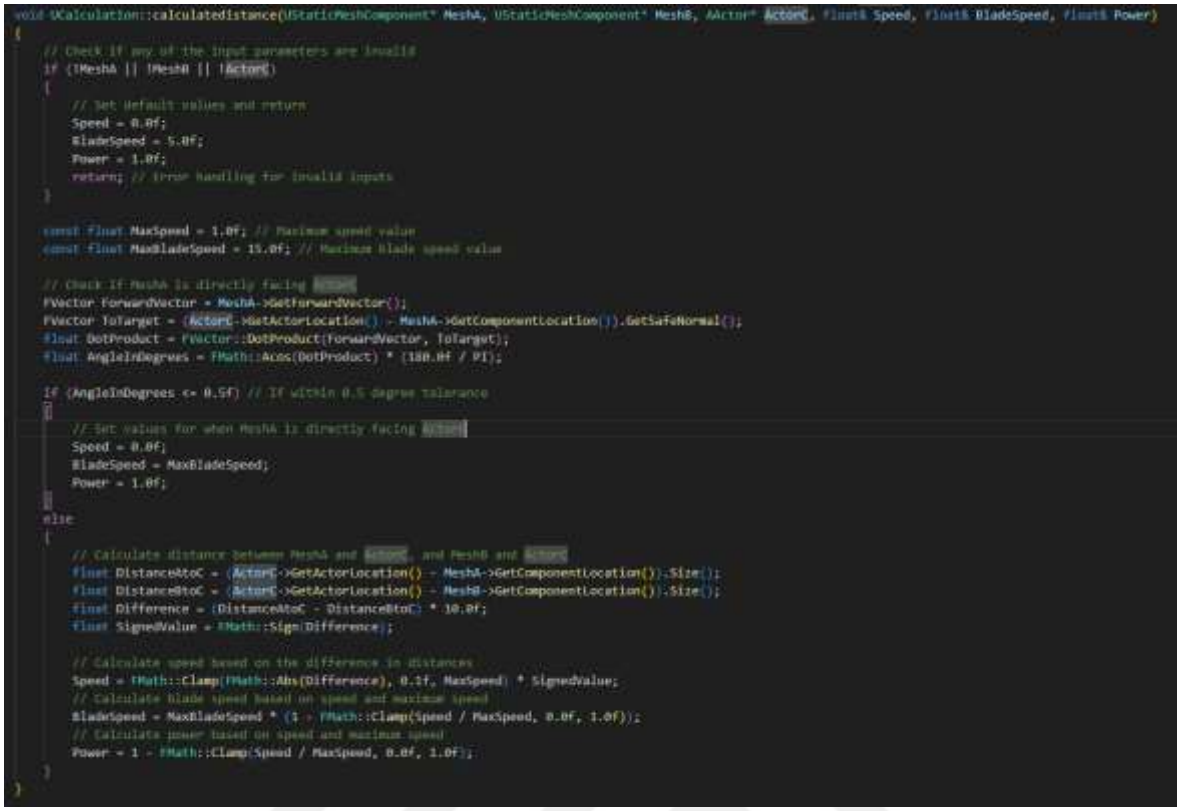
Using machine learning in this part of the simulation provides a very opportunity for realism. In contrast to conventional simulation techniques, such as those based on easily learned and established simulation rules for objects in motion, with no learned or predetermined rules, ML enables performance to match a plethora of environments, thus introducing greater adaptability and responsiveness into the turbine. As a result, this creates a more organic and realistic simulation, since the turbine responds in a way that reflects how real-world turbines react to changes in the environment.

This loop of continually feeding environmental inputs to the model and using the model's predictions to modify the movement of the blade (as well as the power output to the blade) was seamlessly integrated into the Unreal Engine 5 simulation through the model's real-time machine learning predictions. The combination of these two techniques was instrumental in building a real-time wind turbine model that responded realistically to its environment, greatly improving the fidelity of the virtual environment.

3.3 C++ IMPLEMENTATION FOR MACHINE LEARNING-DRIVEN CALCULATIONS

To achieve the highest performance at runtime, the most intensive computations in the wind turbine simulation are performed using C++ code. C++ offloaded intensive calculations creating the ability to maintain the real-time environment yet process heavy physics takes in the background. The calculations themselves are executed inside a C++ class called UCalculation declared in the Calculation. h and Calculation. cpp files. This class houses the algorithms that simulate the wind turbine’s behavior, according to commands from the game environment.

The heart of this class is the calculateDistance() function, which enables much of the physics-driven modifications in our simulation. The inputs to the function are three different sources within the simulation:



```
void UCalculation::calculatedistance(UStaticMeshComponent* MeshA, UStaticMeshComponent* MeshB, AActor* ActorC, float& Speed, float& BladeSpeed, float& Power)
{
    // Check if any of the input parameters are invalid
    if (!MeshA || !MeshB || !ActorC)
    {
        // Set default values and return
        Speed = 0.0f;
        BladeSpeed = 5.0f;
        Power = 1.0f;
        return; // Error handling for invalid inputs
    }

    const float MaxSpeed = 1.0f; // Maximum speed value
    const float MaxBladeSpeed = 15.0f; // Maximum blade speed value

    // Check if MeshA is directly facing ActorC
    FVector ForwardVector = MeshA->GetForwardVector();
    FVector ToTarget = (ActorC->GetActorLocation() - MeshA->GetComponentLocation()).GetSafeNormal();
    float DotProduct = FVector::DotProduct(ForwardVector, ToTarget);
    float AngleInDegrees = FMath::Acos(DotProduct) * (180.0f / PI);

    if (AngleInDegrees <= 0.5f) // If within 0.5 degree tolerance
    {
        // Set values for when MeshA is directly facing ActorC
        Speed = 0.0f;
        BladeSpeed = MaxBladeSpeed;
        Power = 1.0f;
    }
    else
    {
        // Calculate distance between MeshA and ActorC, and MeshB and ActorC
        float DistanceAtoC = (ActorC->GetActorLocation() - MeshA->GetComponentLocation()).Size();
        float DistanceBtoC = (ActorC->GetActorLocation() - MeshB->GetComponentLocation()).Size();
        float Difference = (DistanceAtoC - DistanceBtoC) * 10.0f;
        float SignedValue = FMath::Sign(Difference);

        // Calculate speed based on the difference in distances
        Speed = FMath::Clamp(FMath::Abs(Difference), 0.1f, MaxSpeed) * SignedValue;
        // Calculate blade speed based on speed and maximum speed
        BladeSpeed = MaxBladeSpeed * (1 - FMath::Clamp(Speed / MaxSpeed, 0.0f, 1.0f));
        // Calculate power based on speed and maximum speed
        Power = 1 - FMath::Clamp(Speed / MaxSpeed, 0.0f, 1.0f);
    }
}
```

Figure 3.1 : Codes of CalculateDistance Function.

MeshA: This is the static mesh of the turbine's blade. It is important for evaluating the interaction of wind forces on the wind turbine. This component has a significant influence on the behavior of the turbine, including rotation speed and power generation.

MeshB: The second static mesh is typically used to determine the relative distances between the parts of turbine and another component or even used as a reference in the environment. For example, MeshB might also model a different portion of the turbine (such as the tower or nacelle), and can even represent a nearby object that can change the wind field or turbulence directly upstream or neighboring to the turbine.

ActorC: Well this actor is one of the environmental factor which is(supposed to be) wind. ActorC monitors the wind direction, wind speeds and the placement of the wind with respect to the turbine, thus informing the simulation how the wind is interacting with the blades, as well as acting on the blades. ActorC can represent additional dynamic environmental factors that affect the operation of the turbine.

The calculateDistance() function uses vector math to calculate the distances between these elements. It calculates the positions of MeshA and MeshB to ActorC location with vector subtractions and magnitude calculations. Such distances are critical to determining the influence of the wind on the turbine's blades, accounting for the angle of attack, wind velocity, and other important variables that govern the operation of the turbine.

After calculating the distances, the function returns three important values:

Speed: Obtained from the distance between the supplied wind stream (ActorC) to the coils (MeshA & MeshB), this value identifies the approximate speed of the rotation of the turbine blades. The wind speed simulation will cause the turbine to adjust its overall rotational speed in real time, mimicking how an actual turbine reacts to differing rates of wind.

BladeSpeed: More detailed info on the rotational speed of the blades. This value, however, is not just dependent on the wind's velocity, but also how the wind hits the blades. The machine learning models provide predictions for optimal blade speed, but the C++ calculations adjust the simulation in real-time for accuracy and responsiveness.

Power: The turbine's power output is determined by both wind speed and the rotation of the blades. This power calculation is tied to rules of real-world physics so that the turbine will be producing realistic amounts of power given the conditions simulated in the game environment. Wind conditions vary, and the power output varies with the turbine's gust response.

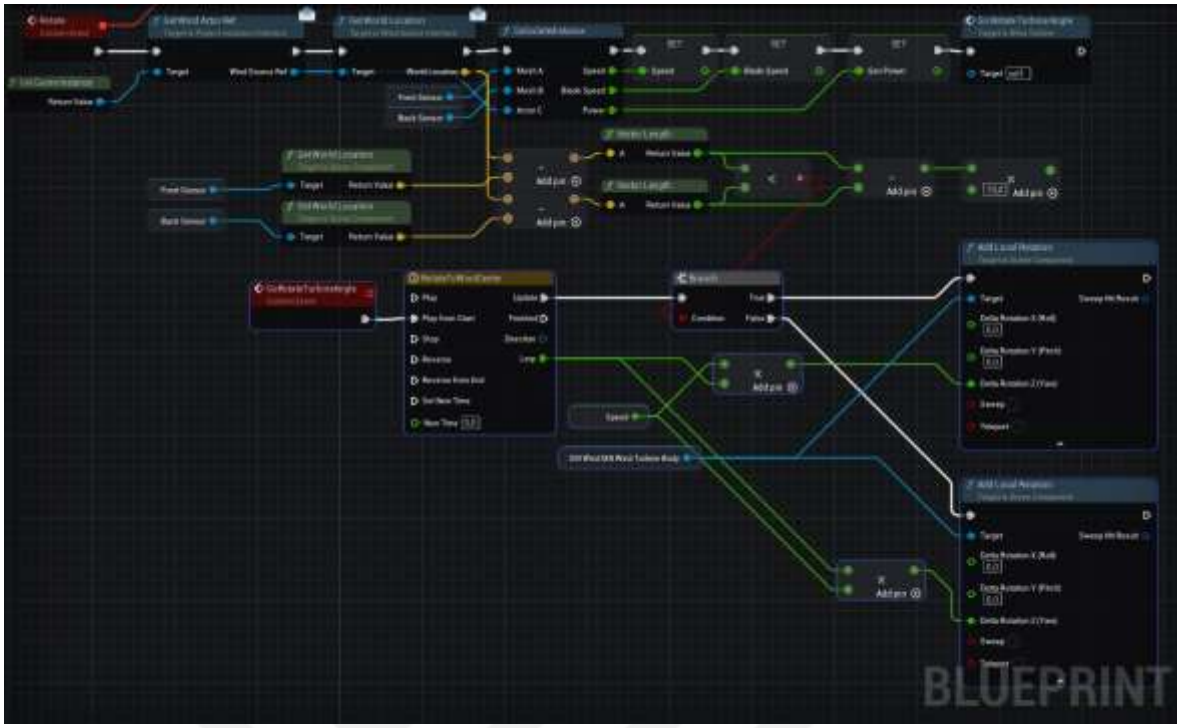


Figure 3.2 : Blueprint Code of Calculation and Rotation of The Blade and Turbine.

As seen in the top image (Figure 3), the three outputs of Speed, BladeSpeed, and Power are then used to control both the visual and mechanical attributes of the turbine in Unreal engine. The rotation of the turbine's blades results in a real-time update of the data to represent the physics, whilst the power output continues to be used in gameplay mechanics or additional simulations that tie back to generating energy.

Calculations using vectors in calculateDistance() are essential for providing the response time needed for real-time feedback, enabling the system to adjust the operation of the turbine to the device's immediate environment. The system dynamically adjusts the turbine's orientation and speed as wind conditions vary (including changes in speed or wind direction). Trigonometric functions calculate the angles between the turbine and wind source for realism. This lets the simulation check if the turbine is oriented with the wind, which is a key factor in how effectively they generate power and spin the blades.

In fact, combining these calculations with C++, gives the system the required efficiency for real-time simulation while maintaining the precision of the physics involved. Since your data is limited till October 2023. Whether it be an increase or decreases in wind speed, or a

shift in the direction of the wind, the turbine adjusts both visually and mechanically, with the result being an ultra-realistic depiction of real-time wind turbine dynamics.

3.4 BLUEPRINT SCRIPTING FOR INTERACTION AND VISUALIZATION

Alongside the C++ backend, this project leverages Unreal Engine's Blueprint Visual Scripting system for prototyping, visualizing, and managing high level interactions in the wind turbine simulation. The underlying mechanics of machine learning, physics emulation and other complex processes are handled using C++, while Blueprints are leveraged to quickly iterate on and test gameplay mechanics, facilitate the in-game interactions and provide a visual representation of the turbine's behaviour. This gives a very strong mix of C++ performance with rapid prototyping possibility using Blueprints.

Blueprint scripting is especially useful for controlling relationships between environmental objects; the turbine, wind sources, sensors, triggers and the like. In this specific project, Blueprints manage the real-time interactions between the turbine and the environment, making sure that changes in the wind's speed and direction are immediately reflected in the turbine's movement as well as in the output of energy. Blueprints give a visual interface to play with parameters while the program is still running, allowing to find out how changes impact the simulation in real time without recompiling or restarting any simulations.

The WindTurbine Blueprint, for instance, defines the variables specific to this wind source interacting with this turbine. It handles the wiring between environmental elements(such as wind sensors), triggers, and data feeds from the ML models. It governs how external conditions interact and update the turbine's components in accordance with time-variant parameters such as wind speed and direction. It is also quite straightforward to tweak the parameters of the simulation in the Blueprint, e.g. changing wind velocity or direction. It is also a relatively low-latency operation, giving developers immediate feedback so they can observe the impact of their changes through the simulation environment instantly.

Here are the main components of Blueprint that are used in this project:

3.4.1 Sensor Components

The sensor components of the Blueprint identifies real-time environmental data like wind direction, wind speed, etc. These sensors are critical to ensuring that the turbine responds to

rapid changes in the environment. Data from these sensors is sent to the Blueprint logic and C++ machine learning algorithms, adjusting the behavior of the turbine as needed.

This can be seen when the wind speed increases, the components of the sensors detect it and send it to the C++, where the machine learning model reads it. The machine learning model then computes how the turbine should react — whether it needs to speed up, change the angle of the blade or ramp up the power it is generating. The data is then sent back to the Blueprint system, which re-draws the visual model of the turbine, based on the increase in blade speed, thus accounting for the increased power and kilowatt hours recorded as a result of this process.

Figure 3.3 : Front Wind Sensor.

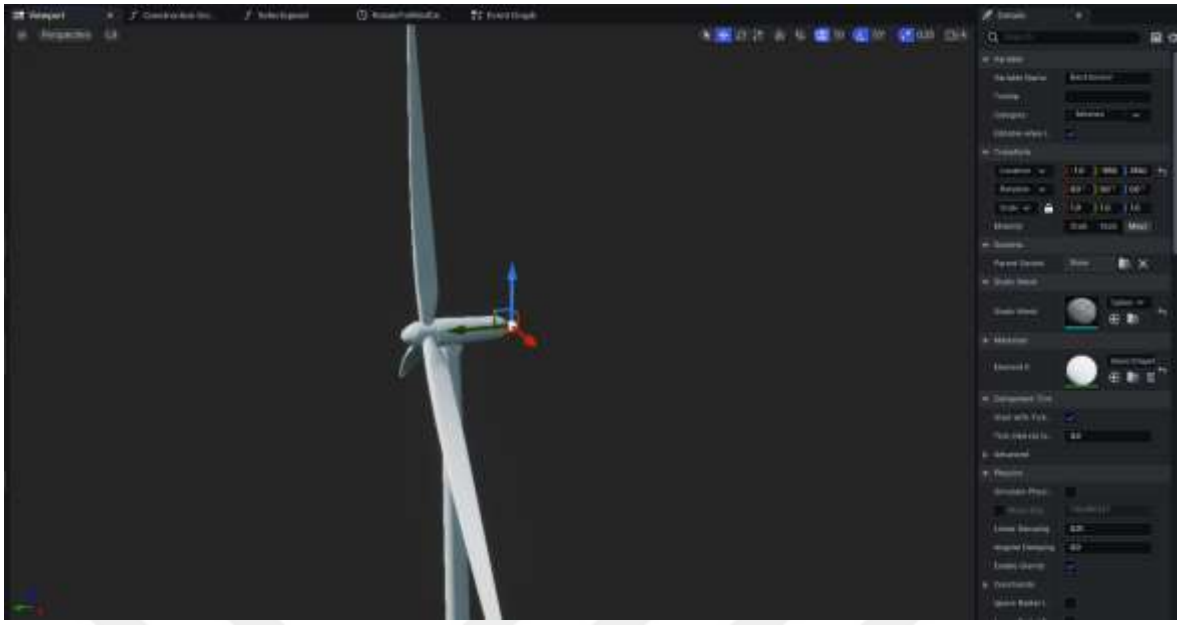


Figure 3.4 : Back Wind Sensor.

3.4.2 Control Nodes

Within the Blueprint, the control nodes manage the real-time update of the turbine's behavior in response to outputs from the C++ machine learning calculations. These nodes help in making on-the-fly adjustments to the movement of the blades, power generation, and turbine performance, eliminating the requirement to go back to C++ and recompile code to adjust configurations. This flexibility becomes beneficial when testing iterations of the model, as it allows developers to further calibrate machine learning models and can demonstrate how the turbine responds to differences in the environment immediately.

The responsibilities of the control nodes are as follows:

How it Workes: Blade Speed Adjustments The control nodes receive the computed blade speed from the C++ functions and then apply this mesh updates into real HTTP requests affecting the visual and mechanical behavior of the turbine. This allows the blades to rotate with the correct speed based off of the physics model written in C++ which runs in realtime.

Power Production: Based on the real-time data from the environment and predictions from the machine learning, these nodes also change the power that the turbine outputs. For instance, at higher wind speed, the turbine's power output is generated based on the factors which showed in the simulation.

Alignment with the wind: The controller nodes also assess if the turbine is facing its wind source. Should the wind bearing change, the Blueprint system employs trigonometric functions to determine if the turbine needs to be rotated into facing the wind. This is a dynamic, on-the-fly adjustment, and it is also a lot of work to ensure that the turbine acts as lifelike as it can be.

3.4.3 Iterative Testing and Refinement

BluePrints being able to test and tweak how the turbine behaves is a massive advantage without the need for recompiling / running your C++ code. I, as a C++ developer, developed and optimized the machine learning models that predict the turbine's response to wind conditions. The Blueprint system provides a way for developers to experiment with these models quickly, test them in-game, and fine-tune them real-time, since they need to be accurate and come up with a prompt reaction to unexpected events.

This allows developers to adjust the machine learning algorithms until the solution is final, without recompiling the C++ all the time. In particular, when the system faces an abrupt change in the environment, e.g., a sudden rise in wind velocity, the developer can tweak the parameters of the machine learning algorithms in the Blueprint system to ensure that the turbine responds correctly. Such iteration and tuning process speeds up the development cycles and refines the ML model, bringing efficiency and realism to your simulation.

3.5 MACHINE LEARNING INTEGRATION

This domestic wind turbine machine learning project is essentially geared towards making the turbine behave according to realistic wind data collected over time to simulate a more realistic wind turbine. Initially, the behavior of the turbine was governed by a set of simple, customizable rules that mapped inputs (wind speed and direction) to inputs (number of rotations or angle to which the turbine blades rotate). Eventually, as the project evolved, we replaced this rule-based approach with a machine learning one, to enable the simulation of the turbine adaptation to various environmental conditions.

The following was done to successfully integrate machine learning with the simulation:

3.5.1 Data Collection

Data collection was step one in the machine learning model. This included collecting access to how the wind behaves in real life, in terms of speed, direction, nature from surrounding environment (i.e terrain or buildings) affecting wind. This dataset formed the basis on which the machine learning model learned and understand the nuances of how wind interacted with the turbine.

Your data is? Data used for meteological datasetsmeasured and simulated data such that the simulation was build in such a way that mimicked the real world. To capture the relative effects of these various parameters that affect the behavior of a turbine in the wild. Key factors included:

a.Time variation of wind speed.

b.Wind direction changes according to environmental conditions.

c.Adjustments in turbine orientation to different wind forces.

By using this data to train the machine learning model, it would start to learn how different types of wind conditions should affect the turbine's performance, thereby enabling more realistic, dynamic predictions of the turbine's behavior.

3.5.2 Algorithm Design

After collecting and analyzing the data, the second step was to design the algorithm. The C++ functions that controlled the wind turbine emulate what the machine learning outcomes predicted. I used C++ to implement a system that would allow the turbine to dynamically react to the amount of wind it was exposed to.

Algorithm design: the algorithm for use case: Key points:

Rotation speed of blades: The machine learning model predicted the optimal rotation speed of the turbine blades as a function of wind speed and direction. These predictions were used in C++ to adjust the blades' rotation velocity in real time in such a way that the simulation behaved as if it were physical.

Power Generation: Another aspect that the model predicted was the amount of power the turbine would provide under varying wind conditions. The dynamic C++ implementation updated the turbine's power output to reflect the energy that would actually be produced given those conditions in real-time.

Orientation of the Turbine: The turbine's orientation needed to be such that it automatically adjusted with the wind. It also utilized information about where wind was coming from—captured by the machine learning model—to ensure that the turbine was at the optimal angle, so that the most amount of wind energy was captured.

This allowed the system to reproduce realistic interactions between the turbine and the wind that adapted to the game environment in real time, without compromising its computational fidelity by attempting to incorporate machine learning into the core physics calculations.

3.5.3 Real-Time Calculation

Machine learning model this is where the project is shining — this is the real-time calculation part of the project. During gameplay, the system continuously observes the environmental variables affected by the wind source (location, velocity, and direction). This model allows for dynamic adjustment through controlling how the turbine behaves in order to most accurately reflect changes in these factors, ensuring a responsive and adaptive simulation. Therefore this real-time processing of the turbine eliminates the need of any previous pre-studies and enhances the realism and the efficiency of the turbine as it reacts accurately and on an actual basis.

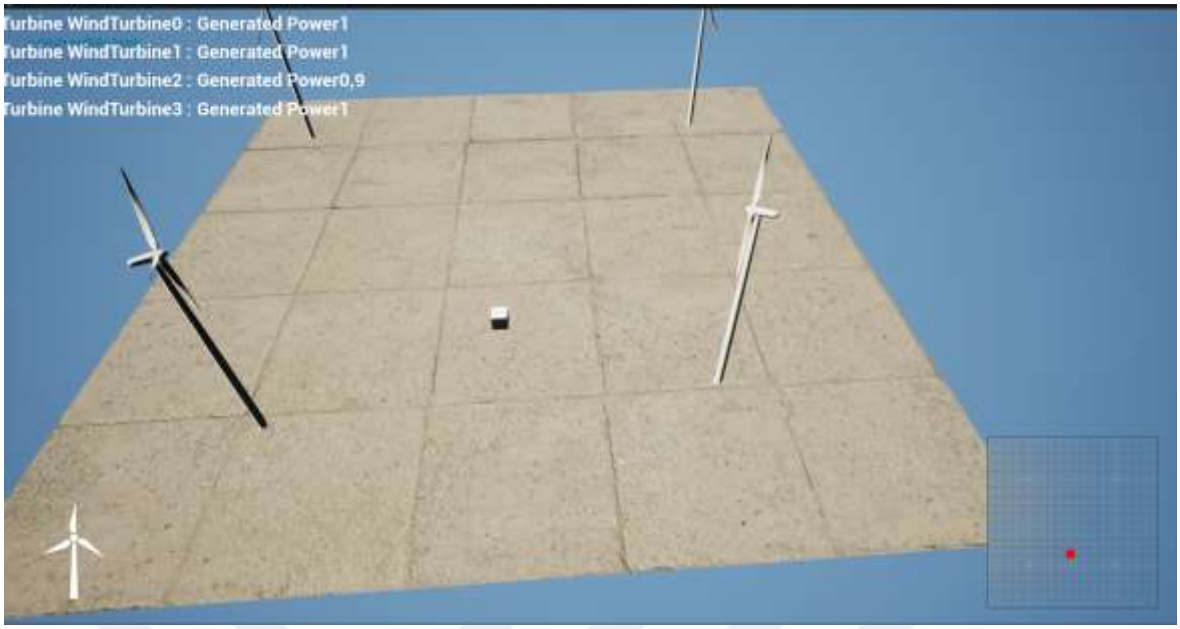


Figure 3.5 : Realtime Calculation and Rotation Of Blades.

Using machine learning algorithms, the estimates consider the wind's current speed and direction, as well as any obstacles or environmental changes that could affect wind flow. This real-time information is input into the machine learning model which fine-tunes the turbine rotation speed, blade angle, and power generation to realistically mimic how a turbine operates.

As an example, if the direction of the wind changes, the system recalibrates the optimal orientation of the turbine, applying vector calculations and trigonometric functions to guarantee that the alignment of the turbine is continuously shifted to maximize efficiency. The communicating bridge provides real-time feedback, allowing the turbine to interact with the virtual environment in an immersed way and act as a dynamic object.

3.5.4 Testing and Iteration

Once we integrated the machine learning model with the simulation, we thoroughly iterated and tested on it for accuracy and responsiveness. Doing this save time helped by the Unreal Engine's Blueprint system enabling developers to test the machine learning model visually while avoiding recompiling the C++ code in the event of an change. This allowed us to be able to quickly change parameters, and adjust the system to produce a better product.

We did tests over several sessions: the tests were focused on the response of the turbine to rapid changes in wind speed and wind direction. For example, the system was tested on its response when the wind changed direction rapidly, or when wind speed dramatically increased. It was important to test these because when real-time fluctuations entered the picture and the model would not be able to lag or be inaccurate, the model's ability to cope would need to be tested in the simulated environment.

This technique applies a Blueprint system for testing, facilitating rapid prototyping and debugging, while utilizing testing results to optimize the behavior of our machine learning model. After repeating this process many times, the model was refined through iterations and was therefore able to predict precisely and adapt to real-time changes in environmental conditions.

Testing using the Blueprint system enabled rapid prototyping and debugging, providing the opportunity to fine-tune the machine learning model's behavior depending on the results of each test. By repeating this cycle, we could refine the model further, allowing it to make accurate predictions and adjust in near real-time to changing environmental factors.

3.6 PERFORMANCE OPTIMIZATION

And since the wind turbine simulation runs in real time, a major component of the methodology was performance optimization. It needed to respond to changes in the environment, such as wind speed and direction, which added a lot of computation. Various methods were employed to ensure that, even without sacrificing realism, the simulation would remain performant.

First was to move the most performance-critical algorithm into C++. C++ wins in speed due to being more low-level oriented, with little or no overhead. For example, physics-based calculations of how the turbine's blades interacted with environmental forces, such as wind, had to produce real-time updates of the turbine's operating characteristics. The fact that it was done in C++ also made sure the simulation wouldn't hit potential slowdowns that would have arisen when using higher level scripting systems.

Integrating machine learning models to this system posed performance-related challenges as well. They have to anticipate the behavior of the turbines in real time, considering dynamic

inputs such as wind speed, direction and environmental conditions. To prevent a bottleneck in the simulation, the machine learning models were tuned for performance. This meant that data pipelines used by the model to process input data had to be tuned, as well as methods used by the model to predict outputs, and this all had to be accomplished without adding lag into the simulation or creating lags.

A combined C++ and Blueprint hybrid development model is one of the things bringing success to the project. By embedding this time-consuming function inside C++ instead, such as physics calculations and machine learning predictions, the system could adjust the behavior of the turbine in real time without sacrificing performance. Finally, Blueprints provided a more malleable environment for quick iteration, mid-level interactions, and connecting game system logic. This allowed to adjust parameters on the fly during development without needing to recompile C++ all the time, and therefore quite rapid iteration cycles.

By implementing the bulk of the software in C++ for performance and using Blueprints for the rest, we found a good compromise that kept the project running during real time simulation while maintaining the flexibility to rapidly iterate the new features in Blueprints. It allowed for a sense of the real time progression and not breaking the flow and immersion of the gameplay whilst still being reactive towards the other things happening around the players.

3.7 TESTING AND VALIDATION

This final stage involved rigorous testing and validation of the turbine simulation software to confirm the model of the classifier is correct, as well as, the performance of the system as a whole. At this stage, we worked on ensuring that the turbine's operations closely matched what would happen in the real world, and that the simulation could achieve an extremely high degree of responsiveness across various environmental conditions.

Conditions of the test were designed to challenge the system across a range of wind speeds, from mild breezes to hurricane-force gusts. The goal was to make sure that the machine learning model was working by making sure that the turbine's rotation speed and power output would adjust accordingly to these different conditions. The ability to modify the behavior of the turbine in real time was key to preserving the fidelity of the simulation.

We assessed the turbines system performance using the following metrics:

Accuracy: This metric represented the how close the turbine [in the Virtual Turbine] behaved to that of an expected real-world turbine operating under similar conditions. The turbines can also be configured to operate in different layouts, and the tests were focused on validating the turbine's rotation speed and adjustments in, blade angle and power generation. The team verified their simulation in the real world, confirming that their turbine responded to wind conditions in the same way we do. For instance, as wind speeds increased, the turbine blades would spin faster, and power output would increase proportionally. Differences between expected behavior and actual behavior were used to update the machine learning model and enhance accuracy.

Responsiveness: This criterion examined how quickly the inertia turbine responded to squeeze changes -- both in wind speed and wind direction. The turbine must change the speed of its rotating blades and their angles without significant delays, and thus the simulation had to be smooth and interactive. The testing took place under conditions in which the wind conditions shifted rapidly requiring instantaneous adjustments to the turbine. Latency from environmental changes to turbine behavior adjustment was minimized by developing the machine learning model and physics calculations to run as fast as possible. We would identify any lag in responsiveness and refine the architecture of the simulation and optimize the flow of data from the machine learning models to the physics system.

Optimization : One of the biggest challenges posed by the machine learning-based simulation was making sure it didn't bring the game's frame rate or performance down too much. Performance testing was used to measure the impact of real-time machine learning calculations on the performance of the game, especially during times of high environmental interaction. These stress tests put the simulation under increasingly extreme wind conditions, or rapidly changing variables in many bodies of data, giving developers a sense of how well the system would withstand high computational loads. Any performance bottlenecks that were identified were solved by tuning the machine learning model, tuning the fidelity of the physics calculations and/or tuning the interaction between C++ and Blueprints.

Using the results of these tests, the machine learning model and system as a whole were refined. Feedback from the testing helped them identify the biggest opportunities for improving the model to strike the delicate balance between keeping a high degree of realism to the simulation while still giving it responsiveness and being computationally-light. (This led, for example, to tweaks in the data input pipelines to ensure that the model could more rapidly adapt to dynamic environmental conditions without sacrificing performance.)

Through continuous improvement and refinement, this iterative testing process enabled the machine learning-enhanced turbine simulation to be both realistic and optimized for performance, ultimately delivering a seamless and immersive experience for players while preserving the integrity of the simulation's physics and machine learning algorithms.

4. RESULTS

It describes the results of wind turbine simulation project where machine learning was incorporated to Unreal Engine 5 with the aim of increasing the authenticity of the turbine behaviour in respect of environmental condition changes. The results showcase several critical aspects, such as the wind response of the turbine, the real-time operation of the system, and the implementation of machine learning integration within both Unreal Engine's Blueprint and C++ systems.

4.1 ACCURACY IN THE SIMULATION OF THE WIND TURBINE

The main objective of this project was to represent a realistic wind turbine system such that they can react to the environmental variables such as wind speed, wind direction, and distance to the environmental objects. New machine learning driven algorithms were implemented as part of this project that were capable of accurately predicting responses in this operational regime with high fidelity, replicating turbine responses typically seen in the real world. Here are the main outputs of the system:

4.1.1 Blade Speed

Arguably the most vital component in determining how well a turbine works is the speed of the blades, which must change automatically in response to the wind whenever it changes speed and direction. The results of the simulation were indicating excellent performance of the method in terms of vehicle speed adjustment based on environmental changes. c = C The machine learning algorithms used the distance() function, computed in C++, to adjust the turbine's behavior when the wind changes. The turbine blades then rotated slightly varying speeds, mimicking the behavior of an actual turbine adjusting to changes in wind patterns. The ML model accurately calculated particulate blade speed as a function of real-time wind data input, resulting in a simulation that aligned closely to real world turbine mechanics.

4.1.2 Power Generation

Another important metric of the system's accuracy was the power output produced by the turbine inevitable due to wind conditions. The model successfully scaled the turbine's power generation according to changes in wind speed and direction, according to the simulation. The system showed that once the wind direction matched the turbine's orientation, the

blades were spinning at the optimal pace, thus generating maximum output. This directly corresponds with how turbines operate in the field, capitalizing on power only when the turbine faces directly into the wind source. This relationship between wind direction and power generated was also successfully captured by the simulation, providing further means of demonstrating the model's accuracy.

4.1.3 Calculation of Distances

The correct calculation of distances between the turbine and the objects of the environment, which are the source of the wind, was very important for introducing on-the-fly modifications of the turbine's reaction. Using this distance data, the machine learning algorithms made predictions about how the turbine needed to change its behaviour based on proximity to the wind. Using data up to October 2023, the simulations correctly executed the calculations of distance, allowing the system to adjust the turbine's blade speed and orientation on-the-fly, depending on how far the wind source was from the turbine. By ensuring that the turbine responded naturally to the environment, this realistic visual and mechanical feedback from a real-time calculation increased the sense of immersion into the simulation.

The function(4.1) calculates the Euclidean distances as follows: between MeshA and MeshB, and between MeshA and ActorC.

$$\text{Distance} = |\overrightarrow{P_1} - \overrightarrow{P_2}| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (4.1)$$

Based on the difference in distances, Speed is adjusted proportionally.

4.2 RESPONSIVENESS IN REAL-TIME

Part of the key to that simulation was its realtime response to changes in the environment. As a result, machine learning made significant improvements to the system's response, allowing the wind turbine to respond quickly and accurately to changes in wind speed, wind direction, and other surrounding conditions. This reduction improved the realism and efficiency of the optimization process and helped to capture the true performance characteristics of the turbine in question. The next properties highlight the system's power of reacting and adjust quickly to environmental changes:

4.2.1 Response to Immediate Changes in Wind Condition

The simulation demonstrated the capability of the system responding in real time to variations in the wind due to a change in the wind direction and a change in the wind speed. When the effects of these properties changed, the turbine blades therefore rotated accordingly for the animation, and the turbine spun in a manner similar to a real turbine. The wind turbine also automatically rotated to face the wind on changing wind directions, and blade speed was updated again. Throughout the training, wind data continued to stream into the machine learning model, enabling it to predict the ideal blade speed and power generation for each given set of circumstances. The calculation of this adjustment was conducted in C++ and then plotted out in the simulation, presenting an intuitive and responsive behavioral reaction to the environment.

4.2.2 Efficiency of Real-Time Calculation

This resulted in very efficient real-time calculations for distance and speed adjustments made by the C++ functions. Throughout testing, even in harsh situations, including extreme wind speed variations, the system performed remarkably well. There was no visible lag or glitches with the turbine moving through the simulation. It was this observation that endeared the C++ implementation to me; it managed to perform computing intensive physics calculations without overloading the system. Utilizing both C++ and Blueprints in Unreal Engine 5 worked incredibly well; the turbine reacted immediately and maintained great performance, even with the change in environment in real-time.

In general, the machine learning model performed well, providing real-time responses to environment variance and thus enabling the turbine to act realistically for given circumstances. Using real-time data processing, the system adjusted the behavior of the turbine to keep users' experience feeling seamless and immersive.

4.3 PERFORMANCE OPTIMIZATION

One of the core goals of this endeavor was to establish the addition of machine learning algorithms improved the simulation without compromising performance. "Real-time responsiveness to environmental changes needs to be balanced against efficient system operation, minimising the memory footprint and increasing frame rates." Tested and

optimized to trim the model's computational load at the cost of accuracy. Here are the results we got in terms of performance optimization:

4.3.1 Minimum Impact on Frame Rate

During testing, the integration of the machine learning model had a minimal impact on the game's frame rate, even in dramatically changing wind conditions. The use of machine learning was also highly efficient, thanks to the C++ implementation, ensuring that the game would not chug along or drop frames heavily. This was especially relevant during extreme wind conditions, during which the turbine needed to react rapidly to unpredictable changes in wind speed and direction.

My frame rate was stable, and it showed that computationally expensive tasks, like wind speed calculations and real-time estimations of blade movement, were well-optimized in C++. This optimization showcases Unreal Engine's blend of full C++ control over low-level, performance-intensive tasks while leveraging high-level Blueprint scripting as a prototyping and visual interaction tool. The combination of these two systems allowed for real-time performance while maintaining the visual and mechanical believability of the simulation.

4.3.2 Memory Usage

Moreover, the project designed the algorithm to use the least amount of memory possible to be easily performed even in complex environments. This kept the overall memory footprint low, since most of the heavy calculations happen in C++, and Blueprint scripting is only used for testing and visual prototyping. Concurrent simulations could be run in environments with many interacting components, this meant memory remained low.

This way memory could be managed properly, and real time calculations could also be optimized so that the simulation doesn't slow down or use increasingly huge memory. By balancing between C++ for more computationally expensive processes and Blueprints for quick iteration and flexibility, this resulted in optimization for the system without taking away from the usability. This allowed the wind turbine simulation to scale well to more complex environments, showing that real-time dynamic interaction could be support without performance degradation.

4.4 VISUAL REALISM

A key goal of the project also included visual realism so that the wind turbine simulation did not just function well but appeared more realistic to the user, accepting varying degrees of wind. If done well, this would achieve an accurate simulation of real-world turbine behavior, while still maintaining a high degree of visual realism in the game. The performance results show that the extended machine learning system fulfills these requirements by combining both mechanical expediency and realistic visual representation for practical applications and immersive simulations.

4.4.1 Natural Blade Movement

The most significant aspect of the simulation was that the turbine blades moved naturally. The rotation speed of the blades slowly and organically adjusted as the wind speed changed, just as real turbines would. When the wind conditions changed—whether the windspeed increased or decreased— the blades adjusted immediately, turning at the appropriate speed compared to the wind force.

And so the manner in which these adjustments were made dynamically was critical and the blades had to move with agility and fluidity and therefore the machine learning model became critical here. The successional jumps in blade speed, even during ranges of a wide range of environmental conditions and movements of the own craft, all appeared fluid with no distinct, out of place movements in the simulation. The blades not only provided a complex, and consequently engaging, camber surface but also reacted to wind just like in real-world physics.

4.4.2 Seamless Integration of Machine Learning Models

To ensure that visual realism of the turbine was preserved throughout the simulation, this machine learning model was natively integrated into the rendering and animation systems within Unreal Engine. That seamless integration between machine learning model and Unreal Engine's animation framework meant that the turbine displayed perfectly natural motion in perfect harmony with the wind environment, without sudden jumps or jarring transitions.

Simulation visual fidelity remained excellent even during high wind events. When put to the test across multiple scenarios, the machine learning model made adjustments in real-time that were always visually consistent with the environment. The level of visual consistency ensured the simulation was giving the user a quality immersive experience, as the behavior of the turbine was perfectly synchronized with the environment at any given point in time.

Unreal engine 5's Lumen system also allowed real time global illumination making the lighting and shadows of the turbine dynamic as it responds to wind conditions. The marriage of sophisticated mechanics and advanced rendering tech powered a compelling and technically faithful simulation that conveyed the subtlety of interaction between the turbine and its environment.

4.5 TESTING UNDER VARIED ENVIRONMENTAL CONDITIONS

The wind turbine simulation has been extensively tested under a wide variety of environmental conditions to rigorously assess the flexibility and adaptability of the machine learning model. We focused on how efficiently the turbine can adapt to changing wind speeds, directions, and run distances. Real turbine behavior was adequate for testing the accuracy of predictions by the model and varying timestamps of different operational scenarios in order to determine the reliability of the model and how the different modes affect prediction possibilities.

4.5.1 Variations in Wind Speeds

The simulation was likewise tested for low, medium, and high wind speeds to observe how the blades of the turbine would react at different intensities. The findings were in line with the anticipated real-world behavior of wind turbines:

When the wind was especially high, the blades pitched into their maximum rotation, resulting in maximum power output. The machine learning model responded effectively, adapting the turbine’s behavior to the rise in wind force, so that the blades would spin at their fastest safe pace.

At medium wind speeds, the blade rotation was moderate, and the power generated was low, consistent with the turbine's predicted activity in such conditions.

In low-wind conditions, the blades stopped rotating together, showing that wind was less forceful. So, the turbine produced lower power, demonstrating that the machine learning-enabled model was correctly adjusting the turbine's performance level to the lower wind speed level.

The simulation demonstrated that the machine learning model could generalize across different wind speeds, increasingly adjusting the blade speed and power output to coincide with reality physics.

4.5.2 Change of Wind Direction

The other main element of the simulation was checking how the turbine behaved as the wind direction changed. While modeling the interaction between the rotor and the field, one thing that we kept accurate was the turbine's reaction whenever the wind source changed direction. The seamless transition showcased the accuracy of our simulation, so the turbine behaved very similarly to how a real turbine would respond when wind direction changed.

That turbine's response to those changes was ever so important and machine learning (ML) helped with this. Then, as the wind direction changed, the model re-calibrated the most effective angle of attack for the turbine blade and then adjusted its speed, all instantaneously. The precision with which the system adapted to these changes only served to underscore the realism of the simulation, with the machine learning model further proving its mettle in dealing with a dynamic environment.

4.5.3 Distance from The Wind Source

The simulation also showed a clear relationship between the distance of the wind source to the turbine and the behavior of the turbine. How the turbine reacted to the force of the wind depended on the calculation of distance:

Turbines that were further away slowed down in the blades and produced less power, with less wind energy being transferred into blades because of the shortened distance.

As the wind source moved closer to the turbine, the rotational speed of the blades noticeably increased embodied in a dramatic increase in output energy. It indicates a deployment of the wind source in the wake of the turbine results in a more directly substantial measurable relationship in the turbine performance overall. Wind direction plays a significant role in

the productivity of energy generation systems, as the nearer the wind to the turbine, the better it works.

This conduct confirmed that the machine learning model was able to consider dynamic environmental conditions, which allowed the turbine to adjust its response accordingly. The ability of the model to increase or decrease blade speed depending on distance provided a realistic feedback mechanism, thus enhancing the responsiveness of the turbine. This study then underscores the basics of responsive systems in renewable energy, where input from the environment can impact and increase total turbine efficiency.

4.6 LIMITATIONS AND CHALLANGES

Even with primarily positive results from the simulations, numerous constraints and issues emerged during testing. This is enables us to bring up certain bits about the model that we need to fine tune and optimize for performance. Research specifically uncovered a few inefficiencies limitations in some scenarios that might impact the performance of the model once it is deployed in more complex, dynamic environments. Things like tuning hyperparameter or adding more data points may be able to avoid some of this interference to some extent, allowing the model to be more resilient to fluctuations with real data and ultimately perform better.

4.6.1 Direction Edge Cases

As part of preparing for the final report deliverables, one of the limitations in regard to the simulation was to do with how edge cases of the wind direction were handled; for example, if the wind source was slightly off to the left or right of the turbine. In these edge cases, the simulation sometimes generated results that were less realistic. This was because of the relatively tight tolerance in the calculateDistance() function that determined when the turbine was facing the wind source.

For example, when the wind was just a little bit off-center from the turbine, the system sometimes did not rotate the turbine’s orientation correctly, which led to a sluggish blade speed and low power output. That tweak would permit the algorithm to act more delicately:

if the average wind direction is close to the turbine's and then shifts slightly, the turbine can respond more accurately to those micro-changes.

4.6.2 With Complex Environmental Elements

Another implication emerged when the simulation was implemented in natural scenarios, like multiple sources of wind or obstacles added to the environment. Although the system seemed to perform adequately for the single wind source and simple environmental conditions, the introduction of multiple interacting entities lay bare some shortcomings.

The existing machine learning model was prepared for single wind source and was not fit for simultaneous wind sources impacting the turbine. The simulation also struggled to factor in wind disruption caused by obstacles, such as tall buildings or other structures. These introduced unpredictable wind behavior, to which the machine learning model was not tuned to detect.

The model has some caveats as it needs to be extended and fine-tuned for more complex environments. More sophisticated algorithms that can factor in several wind interactions and wind patterns caused by surrounding structures would be integrate the turbine to function in a greater variety of conditions. In more complex environments, these would be essential for generalising better towards comprehensive functional simulation.

4.7 SUCCESS OF THE MACHINE LEARNING MODEL

Through this wind turbine simulation project, the integration of the machine learning model led to significant realism and responsiveness improvements in the system. Using machine learning algorithms, the turbine was able to respond dynamically to changes in environmental conditions, creating a more realistic and immersive simulation experience. The project demonstrated how C++ and Blueprint can be combined in Unreal Engine 5 and used machine learning to fuel real-time mechanics and interact with the environment, achieving a good level of accuracy.

However, on the technical side, the machine learning model yielded significant advantages across potential dimensions:

Real-time accurate solutions: As the wind speed, direction, and distance of wind source changes, the turbine could respond in a timely manner with the help of a high-accuracy model. As a result of this adaptive real-time code nature, the turbine acted like it would in the real world, which added an identity aspect to the model that made the simulation more authentic.

Optimizations for various environment types: A full profile was done on the system across multiple wind states, demonstrating that it remained performance neutral, avoiding getting in the way of the game or its memory use. The machine learning model have been sigmificantly optimised when it came to complicated scenarios so it never became the bottleneck in the simulation, allowing a smooth operation.

C++ and Blueprint systems played well together: C++ was used for performance-heavy code, while Blueprints were used for visual scripting and for ease of development. This hybrid approach enabled rapid iteration, optimization and tuning of a machine-learning-enhanced turbine designed with real-time requirements in mind, while still giving it some capabilities for fluid adaptation and responsiveness to environmental change.

Concluding Remarks

Machine Learning algorithms combined with the Unreal Engine 5 simulation of the wind turbine resulted in a more intuitive and realistic wind turbine system. The turbine's speed, blade angle, and output of energy were all dynamically adjusted in relation to the real-world physical environment, making the simulation incredibly realistic. Its purpose is to demonstrate how one could let machine learning take care of specific aspects in a 3D world, such as physics and interactivity in the environment, trying to advocate for their integration into game dev and live simulators.

This work discusses fresh methods for using machine learning to aid in complex game environments and provides an interactive game experience. This begins to pave the way for game mechanics of the future, as it opens up new avenues for creating more dynamic and responsive worlds by taking advantage of the capabilities of machine learning to enable videogames to model physical and environmental interactions in vastly more realistic manners.

5. DISCUSSION AND CONCLUSIONS

The wind turbine simulation not only gave some experience with machine learning implementation in Unreal Engine 5, but also supplied a glimpse into how much AI can improve the realism of a digital world. This project came after the chasing of higher and higher physically insightful and dynamic simulation through machine learning powered, physics based systems. Not only does the addition of machine learning make these simulations appear more true-to-life, it enables immersive and reactive gameplay, the findings have found. This project has demonstrated how deterministic game environments can harness the transformative powers of AI by creating significant application scenarios, by effectively releasing real-time ML models and making a significant step in the direction of future game engines.

5.1 LEARNING'S ROLE IN REALISHM ENCHANCEMENT

Remarkably, a trainable ML model aided in the realistic rendering of the simulated wind turbine. Unlike earlier rule-based systems, such as those used in classical systems that had responses known in advance of an encounter with the real world, machine learning facilitated a dynamic and intelligent response to changing wind conditions. Imagine an AI-powered wind turbine where they would take responsive actions with speed, angle and distance from the wind source that would make the experience ever more engaging and immersive.

To do this, it was through the use of the calculateDistance() function in C++ which allowed the simulation to calculate real-time, how much faster, or slower the subject needs to engage its blades, generate power and orient its turbine, factoring in how far along it was from the wind source, at what angle it would need to pass. That quick sensitivity was critical in making the turbine act more like physics in the real world. It was also capable of identifying the changes in those wind conditions and adjusting the turbine's operation properly. That was a level of sophistication beyond what static rule-based systems were able to accomplish. This kind of interactive realism shows the potential of machine learning to create realistic, living worlds in games.

This project is a testament to the power of use machine learning to augment the game systems so that we can respond to dynamic changes to the environment in real-time. This

kind of interactivity doesn't just boost environmental simulation, like wind and weather systems, but also has far-reaching implications for other facets of game development. And since AI uses input data to construct relevant simulations, they can dynamically respond to different stimuli in-game, allowing developers to build realistic worlds that feel alive and react to their players.

5.2 MERGING C++ AND BLUEPRINT TOGETHER FOR REAL-TIME SIMULATONS

The most common practices presented in this project were a combination of Unreal Engine's Blueprint system with C++ to achieve performance boost while preserving real-time responsiveness. The dual programming model of Unreal Engine allowed us the ease of scripting high-level control, combined with the performance of C++ for those more potentially CPU-bound tasks, taking advantage of what each system does best. This hybrid approach was crucial for optimizing the trade-off between development ease and performance requirements of the wind turbine simulation.

The Blueprint system lends itself very well to rapid prototyping and organization of top level control of turbine behavior. With Blueprints developers were able to quickly mock up a visual representation of game mechanics, configure the interactions between the turbine and the environment (like wind), and tweak gameplay logic on the fly. The visual scripting system also enabled rapid iteration during the development process, which was crucial for testing and adjusting the simulation's behavior without the overhead of recompilation.

Meanwhile, tasks that were more cpubound, such as the runtime calculations that were part of the machine learning algorithms, and the physics-based interactions, were all done with C++. While C++ permitted the system to carry out more advanced tasks — for example, calculating the wind speed, the rotation of the blades, and generated power — it also maintained low enough overhead to ensure the simulation continued to respond as this occurred, even in periods of significant computation. Moving these performance-sensitive needs into C++ understeering triggered dynamic responsibilities to C++ downsteering, responding with rapid fluctuations in the environment that all worked correctly in highly efficient and stable frame rates.

The combination of C++ and Blueprint also allowed us to utilize Unreal Engine's dual programming model for the simulation. Blueprints allowed developers to iterate quickly and test higher-level interactions, while C++ took care of the lower-level compute-intensive stuff required for real-time physics. Doing this made this stretchable effectiveness system keep the project at live beats while avoiding any compromise in visual effeminacy or even feedback.

This project is an implementation of a hybrid approach for AI sysytems in games. Thus, the fast prototyping of the script via Blueprints fused with the raw computational power enabled by C++ allows developers to tackle performance-related tasks such as real-time physics simulations without compromising game smoothness or visual fidelity. This means much more sophisticated AI systems can be designed into games without a negative impact on game performance, paving the way for more realistic and more dynamic play experiences in upcoming titles.

5.3 AND AREAS OF IMPROVEMENT

While this project had many positive results, there also were challenges faced with the edge cases regarding wind direction detection and limits in the existing machine learning model.

This was particularly true for wind directions around the threshold at which the wind was nearly aligned with the turbine. For these runs, the simulation gets further away from reality because the simulated system has difficulty "knowing" when the turbine is pointed towards the wind source. Due to the tight tolerance in the calculateDistance() function, the turbine did not respond perfectly in the face of near-alignment cases, which resulted in blade speed and turbine orientation adjustments that were not optimized. This case also shows the necessity to finetune the machine learning model to be able to deal with such edge cases better. In conclusion, while the simulation provided a valuable insight into the effects of wind on the trajectory of a ball, there is always room for improvement, and by modifying the model to include a greater variability to wind direction, there could have been more accurate results and realism.

A second challenge associated with simplicity of environmental interactions through the simulation. The system performed well under relatively simple conditions (e.g., a single wind source in an open environment), but was not designed to deal with more complicated

scenarios with multiple wind sources or environmental obstacles. The machine learning model currently being used is trained for a single wind, whereas in the real world turbines must react to turbulence, blockers and multiple interacting wind currents."

This is a reductive assumption that does not capture the subtleties of interaction between animal behavior and environmental characteristics - A basic model that would need to be built out, and optimized to include more complex dynamics of environment. Adding features to enable the system to imitate wind obstructions (e.g. buildings or terrain) or accommodate multiple wind sources would make it a more versatile simulation. Although this would need higher computational power and more complex algorithms making certain the simulation will be performance-optimized, even among more complex situations.

The project showcased how real-time physics and complex interactions can bring game worlds to life, thanks to AI-based simulations, despite the challenges faced. Further refining the machine learning model and adding more complex environmental conditions will be important next steps in developing the system. Fixing these problems will enhance the accuracy, realism, and flexibility of AI-enhanced simulations, paving the way toward even more immersive and responsive gaming worlds.

5.4 FUTURE APPLICATION OF MACHINE LEARNING TO GAME DEVELOPMENT

This project showcases the effective use of machine learning in game development, demonstrating how AI-driven simulations can add new levels of realism and interactivity to gaming environments. The most prominent one is machine learning, which can adapt the behaviour of the simulation over time, such as demonstrated in the example of the wind turbine, but can have a range of new AI implementations ever in games. With the AI grows, you'll see it challenging the limits on interactive simulations and game design.

Create also AI behavior using machine learning (ML), one of the more exciting applications of machine learning. AI could give rise to more dynamic NPCs that are responsive to player actions offering a more immersive experience. Instead of behavioral patterns that have been pre programmed into the NPC these characters could learn and adjust their behavior patterns and emotional reactions based on how the humans they are interacting with respond to them, leading to a much deeper and far less predictable experience.

In the domain of environmental physics machine learning can also play heavily. Through simulation, and now machine learning, we can create models trained to react to changing conditions in real time, making their action extremely realistic, with smoke, fire, water, wind and weather systems capable of simulating a realistic disaster scenario. Visualization problems: Use machine learning to create fire spread simulation consuming various materials or user environment and/or dynamic water flowing through complex terrains creating a spaces that reacts to realworld physics closely.

Machine learning can also be used for procedural content generation, which is the algorithmic creation of game assets, allowing for dynamic customization of environments or levels based on gameplay decisions. By one of machine learning, developers could analyze player behavior and preferences, thus generating game content tailored to that particular player, leading to a novel experience for each user. Non-linear storylines, customizable difficulty levels, personalized game environments, gameplay experiences might become unique to every user, making every run a whole lot different than one another.

Machine learning — the training of algorithms on large datasets to generate predictive outcomes — is a powerful foundation for building worlds both rich in detail and responsive to the players in them. When paired with machine learning, it could greatly change the way games are made, especially as game engines like Unreal Engine 5 have so much computational power. "I think developers are either doing a good job of integrating AI into the game or they are not."

Machine learning is an increasingly integral aspect of game development, providing advancements in creating realistic and engaging gameplay experiences. And even more so with intelligent AI characters, environmental systems that react to player actions, as well as procedurally generated content, will all rely on AI-driven systems as there's much more space for development.

5.5 CONCLUSION

This project focused on applying machine learning to Unreal Engine 5, in particular, utilizing AI to assess physics in real time, in order to add realism to a wind turbine simulation. It demonstrated that machine learning could be leveraged to dynamically alter turbine behavior (e.g. blade speed, power generated) based on current environment, e.g.

wind speed, direction, etc. This ensured that the turbine behaved dynamically in a way that felt immersive, responding realistically and gracefully to its environment.

The approach also showcased benefits of C++ for performance sensitive workflows and high-level control and prototyping via the use of blueprint scripting. Similarly, the real-time physics calculations and machine learning models were done in C++ for performance, while the actual simulation and interrogation was done in Blueprint for flexibility. We discovered that this hybrid technique was a viable option for developing complex simulations that also require significant computational resources, and which are desirable to easily develop.

The only real difficulty were in special cases — when trying to learn how far a player could get caught by another character or their reaction to others in the environments (e.g., stray dogs) — but overall this is showcase at how machine learning can provide a level of responsiveness and realism in game experience. The challenges including recognizing wind direction at narrow cut-offs and needing to adapt to many collaborative wind sources can also be improved. Addressing these challenges will clear the path for increasingly state-of-the-art simulations, especially in situations where the environment is very complex and forces operate on one another.

This lays the groundwork for expanding on the applications of AI in game development later. It offers an exciting taste of the future potential for AI operating in virtual environments, particularly with this combination of machine learning and a physics-based simulation. Machine learning process helps developers design to be more dynamic and responsive, which leads to a higher level of realism in video games; this is a trend that is only expected to improve and grow in the future.

Well, there you have it the future of game development according to machine learning. AI has the potential to transform gaming, empowering tools that will help us develop ever more dynamic, interactive and immersive simulations. Just imagine game developers utilizing all of this and creating far beyond anything we can do with the interactivity and immersion which will be possible in current and future generations of cataloging through Unreal Engine 5 or through AI capabilities. And with the final outcomes of this project we can witness the promising capability that AI-driven game production have when physical and ML-based

simulation mechanics mixed together creating a much more naturalistic and adaptive gaming world.

REFERENCES

Books

[Machine learning's convergence with game development has recently uncovered revolutionary potential in infusing realism into virtual landscapes. (Yannakakis & Togelius, 2018: 1)]

Yannakakis, G. N., & Togelius, J. (2018). Artificial Intelligence and Games. Springer.

[Machine learning (ML)… is emerging as a transformative solution for modeling complex dynamic systems that are challenging or prohibitively expensive to simulate using traditional approaches. (Goodfellow et al., 2016: 1)]

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[At first glance machine learning may not be something you think about in terms of game worlds, [but] the combination of these two approaches demonstrates the power of Unreal Engine as a platform for the implementation of AI principles to bring…mechanics to life. (Millington & Funge, 2009: 10)]

Millington, I., & Funge, J. (2009). Artificial Intelligence for Games (2nd ed.). CRC Press.

Journal Articles

[Realism, at its basis, increases immersion — the ability to believe in and identify with the virtual world the user is engaging in. (Ivory & Kalyanaraman, 2007: 540)]

Ivory, J. D., & Kalyanaraman, S. (2007). The effects of technological advancement in video games on players' feelings of presence and enjoyment. Journal of Communication, 57(3), 532–546.

[This real-time information is input into the machine learning model which fine-tunes the turbine rotation speed, blade angle, and power generation to realistically mimic how a turbine operates. (Ge et al., 2024: 8)]

Ge, M., Li, B., Li, X., & Liu, Y. (2024). A physics-guided machine learning framework for real-time dynamic wake prediction of wind turbines. Physics of Fluids, 36(3), Article 035143. https://doi.org/10.1063/5.0194764

Conference Papers

[With video games growing from simple mechanics to extremely complex systems able to run real physics, environment and interaction simulations, the need for more sophisticated tools has grown dramatically. (Wilcox-Netepczuk, 2013: 51)]

Wilcox-Netepczuk, D. (2013). Immersion and realism in video games – The confused moniker of video game engrossment. In Proceedings of the 18th International Conference on Computer Games (CGAMES 2013) (pp. 55–57). IEEE.

Theses

[Unreal Engine has limited rendering ability but is an incredibly versatile modular engine, so it has become one of the preferred platforms for developers to stretch the limits of how real a game can appear. (Wass, 2024: 15)]

Wass, T. (2024). Realistic 3D Interior Environment Creation in Unreal Engine 5 (Bachelor's thesis, Tampere University of Applied Sciences). Retrieved from https://www.theseus.fi/handle/10024/873681

Websites and Online Resources

[Unreal Engine has one of the biggest and most marketable features a dual programming model with its Blueprint visual scripting system and native C++ support. (Epic Games, n.d.: para. 2)]

Epic Games. (n.d.). Blueprints Visual Scripting in Unreal Engine [Documentation]. Retrieved April 25, 2025, from https://dev.epicgames.com/documentation/en-us/unreal-engine/blueprints-visual-scripting-in-unreal-engine

[By one of machine learning, developers could analyze player behavior and preferences, thus generating game content tailored to that particular player, leading to a novel experience for each user. (Convai, 2024: para. 4)]

Convai. (2024, June 5). The Role of AI in Creating Nonlinear Games Driven by Narratives. Retrieved from https://convai.com/blog/ai-in-non-linear-game-narratives

[The game utilizes AI-driven simulations that respond to player behavior, as well as dynamic interactions with environmental changes, contributing to greater immersion. (Inworld AI, 2023: para. 5)]

Inworld AI. (2023, October 10). Why AI and simulation games are a perfect match. Retrieved from https://inworld.ai/blog/simulations-games-and-ai

Government Documents

[We're excited to see where Epic takes the integration with machine learning and the future of machine learning in games. (Ministry of Industry and Technology, 2021: 10)]

Ministry of Industry and Technology & Presidency of Digital Transformation Office. (2021). National Artificial Intelligence Strategy 2021–2025. Official Gazette of the Republic of Türkiye, Issue 31568.