https://doi.org/10.31891/csit-2026-1-18

Pavlo PONOMARENKO
Postgraduate of the Department of Cybersecurity and Intelligent Information Technologies, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine
e-mail: p.h.ponomarenko@khai.edu
https://orcid.org/0009-0005-5998-2517
Vyacheslav KHARCHENKO
Cor.-member of National Academy of Science of Ukraine, DrS on Engineering, Professor, Head of the Department of Cybersecurity and Intelligent Information Technologies, National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine,
e-mail: v.kharchenko@csn.khai.edu
https://orcid.org/0000-0001-5352-077X

Received: 11/02/2026
Accepted:20/03/2026
Published:26/03/2026

UDC 004.94:629.7.05:004.9

INTEGRATING ARDUPILOT SITL FOR DRONE BEHAVIOUR SIMULATION IN GAMIFIED TRAINING

New educational technologies, as well as the challenges of the modern world, encourage the search for ways to improve and develop training methods for specialists to operate in dangerous spaces using unmanned swarm systems. Using technologies such as Unity and ArduPilot, it is possible to create a simulated learning environment that, through gamification mechanics, provides a sense of reward and challenge while learning to interact with a swarm of drones, which requires extensive interdisciplinary expertise. The subject of the study is the interaction module of the Unity game engine and the ArduPilot SITL simulation engine. The object of the study: a software system for training specialists to operate in dangerous spaces using drone swarms. The purpose of the article is to suggest ways to integrate the Unity game engine and the ArduPilot SITL simulation engine in context of a gamification of training specialists to operate in dangerous spaces. To achieve this objective, the paper addresses the following tasks: to formulate the concept of an application that simulates missions with drone swarms in dangerous spaces and provides progression and assessment through gamification mechanics; to justify the choice of a technological stack, in particular ArduPilot, SITL, MAVLink, Unity; to describe the architecture of the Unity ↔ SITL interaction, using TCP/UDP communication, MAVLink message processing, as well as the separation of responsibilities between layers and components. As a result of the research, the architecture was substantiated, and an interaction module between Unity and ArduPilot SITL was implemented using the MAVLink protocol, providing two-way data exchange and scalable integration of multiple simulated devices within a single application. To evaluate the scalability of the system, an experiment was conducted to analyze CPU resource usage and model update time depending on the number of simulated drones. The results showed an almost linear relationship between system load and the number of drones, and no bottleneck was observed in Unity's main thread.

Keywords: gamification, UAV, dangerous spaces, hazardous environments, drone swarms, Unity, ArduPilot, MAVLink

Introduction

Modern unmanned technologies allow reducing risk for specialists and accelerate operations in dangerous spaces [1]. However, coordinating a swarm of unmanned systems requires highly qualified specialists in several fields: computer science, mechanical engineering, and domain-specific expertise (e.g., demining, firefighting). The training of such specialists is complex and comprehensive, and if it is necessary to involve a large number of specialists, as in war conditions, it may become shortened and superficial, thus requiring new approaches [2]. Therefore, the creation of an application that simulates the operation of UAV swarms in dangerous spaces, using gamification elements to support learning, focus, progression, and performance assessment, would enhance the training of such specialists.

Literature review

Current research on gamification indicates that it is an effective tool for improving learning outcomes. Gamification is defined as the use of selected categories of game elements (including goals and rules, assessment, challenges, control, environment, immersion, narrative, and social interaction) in non-game educational contexts [3]. Gamification is particularly promising in professional and adult learning, where it is used not as a full-fledged game, but as a means of structuring complex tasks, supporting gradual progression, and fostering a sense of competence [4].

Studies confirm that gamification, combined with virtual and simulation environments, is an effective approach to training specialists whose work involves elevated risk [5]. And while the reviewed simulators do include gamification elements, such as performance assessment tools and operational scenarios for training pilot skills [6], no implementation was found that features a comprehensive, consistent gamified approach to training swarm system operators.

Examples of real-world missions performed by unmanned vehicles were also examined. The work [7] describes tools for detecting explosive objects using various classes of UAVs. The article [8] addresses real-time prediction of wildfire spread and coordination of firefighting operations. In [9], the authors describe the use of unmanned vehicles for measuring air pollution at key locations. Analysis of such sources is necessary for formulating training missions within the gamified application.

The work [10] provides a systematic review of the MAVLink protocol as a communication standard between unmanned vehicles and ground control stations. Works [11], [12], and [13] describe the successful use of game engines and 3D visualization in scientific applications, which confirms the appropriateness of the chosen development stack.

Thus, despite existing technological and methodological practices, there is currently no systematic gamified solution for training specialists to operate in dangerous spaces using unmanned swarm systems. Besides, it's required modern and usable technological solutions for gamification-based training processes, a specially for critical applications.

Purpose of the article

The purpose of the article is to suggest ways to integrate the Unity game engine and the ArduPilot SITL simulation engine in context of a gamification of training specialists to operate in dangerous spaces.

To achieve this goal, the article addresses the following tasks:

– to formulate the concept of an application that models drone swarm missions in dangerous spaces and provides progression and assessment through gamification mechanics;

– to justify the choice of the technology stack, specifically ArduPilot, SITL, MAVLink, and Unity;

– to describe the Unity ↔ SITL interaction architecture, using TCP/UDP communication, MAVLink message handling, and the separation of responsibilities across layers and components.

– to conduct an experiment evaluating the scalability of the system by analyzing CPU resource usage and model update time depending on the number of simulated drones.

Gamified application

The idea is to create a gamified simulation environment that will gradually introduce the core concepts of UAV-swarm operator interaction in dangerous space. Within the application, a future operator learns to control drones by starting with the simplest missions — for example, measuring air pollution levels in a defined area using a single sensor on a single drone.

A simple mission allows the trainee becoming familiar with the basics of unmanned vehicle control and provides virtual resources for developing their capabilities (Fig.1). By completing such missions, the future operator unlocks new features: drone types, sensors, objectives, and threat types that hinder mission completion. Over time, the trainee even gains access to a drone constructor, which allows them to assemble a drone from components using virtual currency — independently choosing the frame, motors, propellers, battery, sensor suite, and other electronics, based on analysis of the next mission.

Within this constructor, the player can observe how parameters such as mass, thrust, thrust-to-weight ratio, battery capacity, communication range, flight range, and final drone cost interact and change in response to their choices. This gives the user practical experience in customizing drones and matching their specifications to different task types — from versatile builds that balance a wide range of parameters, cost, and efficiency, to specialized drones with a narrower scope but greater effectiveness in a specific role.

Ultimately, the player enters what can be described as a classic game loop, in which they assemble vehicles for a given mission while adapting to available budgets, mission requirements, and accessible components. Upon mission completion, the player receives immediate feedback, iterates on their previous work, or faces new challenges and unlocks new capabilities.

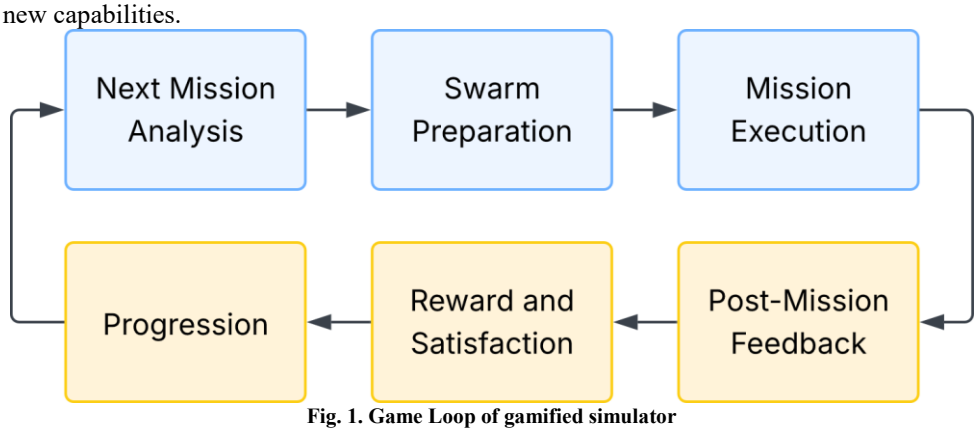

Fig. 1. Game Loop of gamified simulator

Engineering Solution

As the engineering basis, a stable and popular open-source solution is proposed, meaning that the gamification will yield not only a theoretical scientific result but also a practical one — the acquisition of hands-on experience with production-grade drone control technologies.

To that end, ArduPilot [14] will be used as one of the most widely adopted open-source flight control systems. Several well-known Ground Control Station applications are available for ArduPilot — tools used for mission planning and sending commands to drones. The most popular open-source options are Mission Planner and QGroundControl [15]. ArduPilot also includes a built-in SITL (Software in the Loop) component, which enables flight simulation of an unmanned vehicle while accounting for both the onboard hardware configuration and the environmental settings.

Communication between the various components is handled via the MAVLink (Micro Air Vehicle Link) protocol [16]. When SITL is used, it spins up a server and establishes a UDP/TCP transport channel for communication.

Unity was selected as the visualization and user interaction platform, offering rich capabilities for developing game mechanics. But the following ideas can be used in any visualization engine.

From the end user's perspective, all interaction with drones is encapsulated and works identically whether running against a simulated or a real vehicle in terms of protocol. Together, these components allow drone flight to be reproduced in a controlled environment under conditions that closely mirror real-world operations.

Environment Setup

ArduPilot is a large C++ project with a Linux-oriented dependency stack. When working on Windows, it is recommended to use the Windows Subsystem for Linux (WSL). The first step is installing Ubuntu within WSL. This step can be skipped when working directly on Linux.

```
wsl --set-default-version 2
wsl --install -d Ubuntu
```

After running these commands, the console will automatically switch to Linux management mode, where the setup will continue. The next step is to update Linux to the latest version and ensure that git is installed:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y git
```

The next step is downloading the ArduPilot source files, updating the latest versions of all submodules, and launching the build process via the script provided by the developers:

```
cd ~
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
$       git submodule update --init --recursive
Tools/environment_install/install-prereqs-ubuntu.sh -y
```

The script performs installation into a virtual environment (venv), and all subsequent SITL sessions must be launched from within it:

```
source ~/venv-ardupilot/bin/activate
Tools/autotest/sim_vehicle.py -v ArduCopter
```

Running this command starts a SITL session, brings up a MAVLink server, and begins streaming live telemetry from the simulated vehicle to the console. This marks the successful completion of the first stage — the machine now supports running SITL.

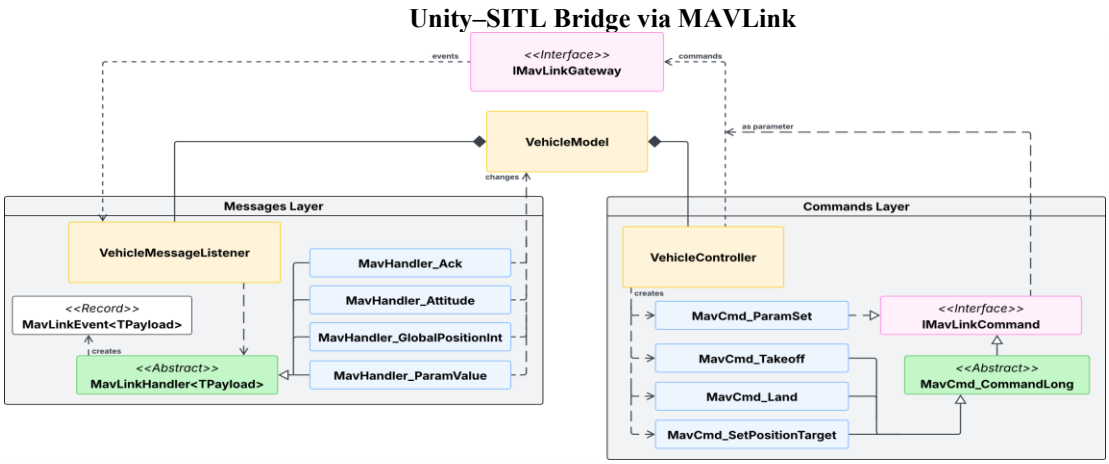

Fig. 2. Architecture scheme of connection module

Architecture scheme is shown on Fig. 2. Its components are described further.

MAVLink is a binary protocol. When a SITL instance is launched, a TCP or UDP connection is opened, and data is written to a stream, from which bytes must be read and assembled into structures according to the protocol's rules. Open libraries exist that implement this byte-reading logic, such as the Mission Planner MAVLink library. In the Unity application, a separate C# TcpClient is created for each drone digital twin. It connects to the host and port specified in the command-line arguments used to launch the SITL instance, and a new Thread is spawned to continuously read from the TcpClient's stream. Each time new data arrives in the stream, the selected MAVLink parser attempts to assemble it into a packet. If a packet is successfully formed, it is added to a ConcurrentQueue, and on the main thread, Unity's Update method periodically checks this queue for newly arrived MAVLink packets.

```
public class MavLinkTcpGateway : MonoBehaviour, IMavLinkGateway
{
   // …
   private readonly ConcurrentQueue<MAVLinkMessage>
      _messageQueue = new();

   private void Update()
   {
      while (_messageQueue.TryDequeue(out var msg))
         onMessage?.Invoke(msg);
   }

   private async void ConnectAsync()
   {
      _tcpClient = new TcpClient();
      await _tcpClient.ConnectAsync(host, port);

      _stream = _tcpClient.GetStream();
      _readThread = new(ReadLoop) { IsBackground = true };
      _readThread.Start();
   }

   private void ReadLoop()
   {
      while (!IsCancellationRequested && _tcpClient.Connected)
      {
         var message = _mavLinkParser.ReadPacket(_stream);

         if (message != null && message.Length != 0)
            _messageQueue.Enqueue(message);
      }
   }
}
```

To keep the codebase organized and maintain a clear separation of concerns, each message type is handled by a dedicated handler, which is also responsible for updating the corresponding drone's model (Fig.3).

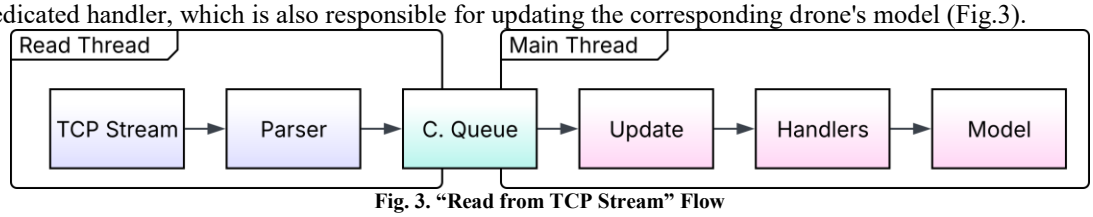

Fig. 3. "Read from TCP Stream" Flow

For the tech demo, command input is implemented using the free TriInspector library, which allows C# attributes to generate Unity Editor controls, such as buttons and read-only fields, for displaying telemetry. These buttons function in both Play Mode and Editor Mode but are only accessible within the editor, meaning they are available exclusively to the developer. While they will not be part of the final product due to their inaccessibility to end users, they serve as a convenient and fast tool for prototyping and technical experimentation. Each VehicleModel has an associated VehicleController that exposes methods marked as buttons. These methods send MAVLink commands to the vehicle and, in doing so, modify its behavior. Fields can be used to pass and live-tune additional arguments.

```
1  public class VehicleController : MonoBehaviour
2  {
3      // ...
4      [SerializeField]
5      private NedVector pointA = new(20, 0, -10);
6
7      [Button]
8      public void Takeoff() => Send(new MavCmd_Takeoff(address));
9
10     [Button]
11     public void FlyPointA() => Send(
12         new MavCmd_SetPositionTarget(address, position: ned));
13 }
```

An application layer called "Commands" creates a statically-typed facade over the MAVLink protocol (Fig.4). Each command has its own representation in the application with clearly defined parameters. Most MAVLink commands share a single structure with the type MAVLINK_MSG_ID.COMMAND_LONG, a subcommand, and up to seven payload fields with no semantics embedded in the code — as can be seen in the MavCmd_Takeoff example. Semantics that the Commands layer encodes, serving as self-documenting code and reducing cognitive load when working within the business logic layer. Ultimately, each command assembles the packet structure required by the MAVLink protocol and writes it to the TCP stream.

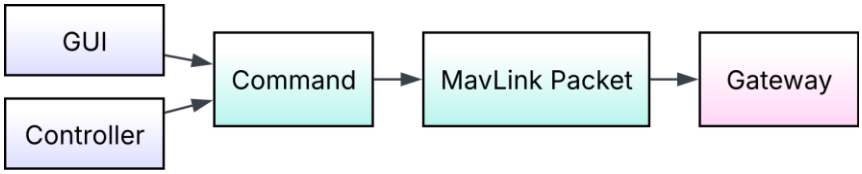

Fig. 4. "Write to TCP Stream" Flow

In this way, a bidirectional communication channel was established between SITL and Unity. On the ArduPilot side, each vehicle is a separate SITL process with its own arguments, which can be obtained from a dedicated service responsible for spawning SITL processes — the VehicleCommandBuilder.

As a result, a vehicle is represented in the application as a single GameObject with the following MonoBehaviour components (Fig.5). To add more drones to the same scene, it is sufficient to duplicate an instance of this prefab:

– MavLinkTcpGateway — the MAVLink communication channel;
– VehicleMessageListener — responsible for receiving messages from MAVLink and configuring their logging verbosity;
– VehicleController — used to issue commands to the drone;
– VehicleModel — stores the drone's current state;
– VehicleCommandBuilder — generates the console command for manual launching a SITL instance.

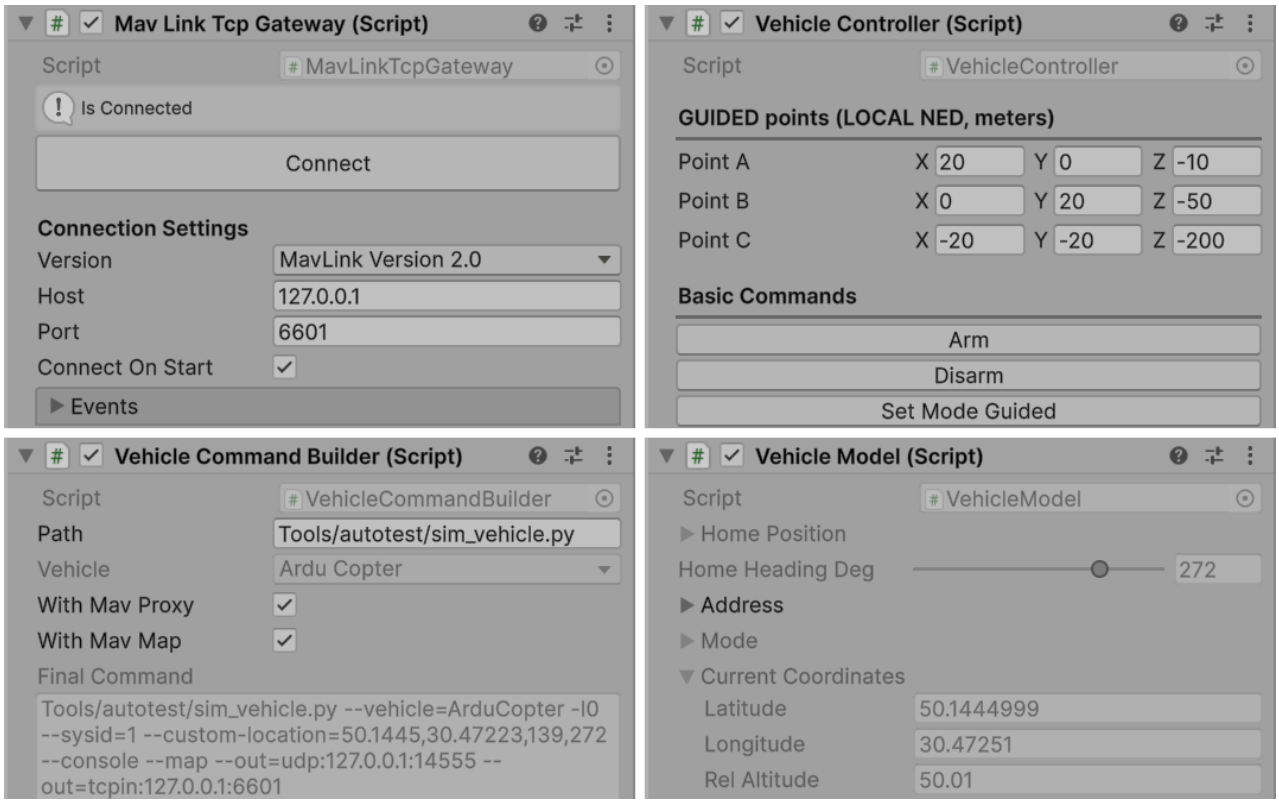

Fig. 5. Screenshots from Unity, demonstrating main components

Since Unity has no access to spawning processes inside the Linux virtual machine, a lightweight Node.js HTTP server is introduced as an auxiliary service to automate SITL instance management. It listens on a single port, accepts start and stop GET requests with a drone identifier as a parameter, and orchestrates SITL processes accordingly — spawning them on demand, signalling when they are ready, and terminating them when no longer needed. This service is launched before the main application starts and remains active for its entire lifetime.

Experiment methodology

Two primary sources of CPU load are of interest in this experiment:

– the vmmemWSL process, which encapsulates the Linux virtual machine and all computation running within it, including all SITL calculations;

– the model update performed on Unity's main thread from the concurrent queue, which at high volumes could become a performance bottleneck.

The goal of the experiment is to evaluate the scalability of the proposed system by analysing how CPU usage and model update time depend on the number of simulated drones.

Before each performance measurement run, the required number of drones is activated as follows. Using the HTTP server running in the WSL context, N SITL processes are launched sequentially. Each process is set to stream GLOBAL_POSITION_INT and ATTITUDE telemetry messages at 10 Hz. The drone is then set to COPTER_MODE.GUIDED, Armed, get a Takeoff command, followed by a SetPositionTarget command targeting a position 1 km away at an altitude of 200 m from the origin. Commands are sent sequentially with a 4-second interval between each. Successful reception of all commands is verified in the console, and the drone's position change is confirmed in the model view.

The vmmemWSL process was measured as follows. Once all conditions are met and all drones are in motion, Windows Performance Recorder is run for 40 seconds. The resulting event log is analysed in Windows Performance Analyzer, filtered by the vmmemWSL process over a 20-second window. The tool reports CPU usage in milliseconds of processor time; dividing by the 20-second interval yields the average load on a single CPU core.

Model update time under the same conditions is measured using the standard C# System.Diagnostics.Stopwatch utility to sum all ElapsedTicks for the Update method of the MavLinkTcpGateway class, including the execution time of all subscribers — which corresponds to a full update of the digital twins of all drones.

Each configuration was run three times and the middle value was taken to avoid outliers. Experiments were conducted on a system equipped with an Intel Core i9-10850K CPU @ 3.60 GHz and 64 GB of RAM, running a licensed installation of Microsoft Windows 11 Pro, version 25H2 (OS build 26200.8037). The simulation environment was implemented using Unity version 6000.3.9f1. The autopilot software used in the experiments was ArduPilot-4.6.0-beta1.

Experiment results

Table 1 presents the results of measuring the amount of resources used by the **vmmemWSL** process according to the described methodology. «vmmemWSL Total» indicates the analytics collection time in milliseconds. «vmmemWSL CPU» shows the number of milliseconds of processor time used by the vmmemWSL process during the analytics collection period. «vmmemWSL CPU %» represents the percentage of CPU usage during the measurement period. «vmmemWSL CPU % per drone» shows the average load on a single CPU core caused by simulating one drone with the vmmemWSL process.

Table 1

Drones Number	0	1	2	4	8	16	32
vmmemWSL Total (MSec)	20000	20000	20000	20000	20000	20000	20000
vmmemWSL CPU (MSec)	7	215	366	665	1385	2998	6450
vmmemWSL CPU %	0,035	1,075	1,83	3,325	6,925	14,99	32,25
vmmemWSL CPU % per drone	0	1,075	0,915	0,831	0,866	0,937	1,008

Figure 6 shows the percentage of CPU core usage (vertical axis) by the **vmmemWSL** process depending on the number of drones (horizontal axis). The resulting graph is close to linear, taking measurement error into account.

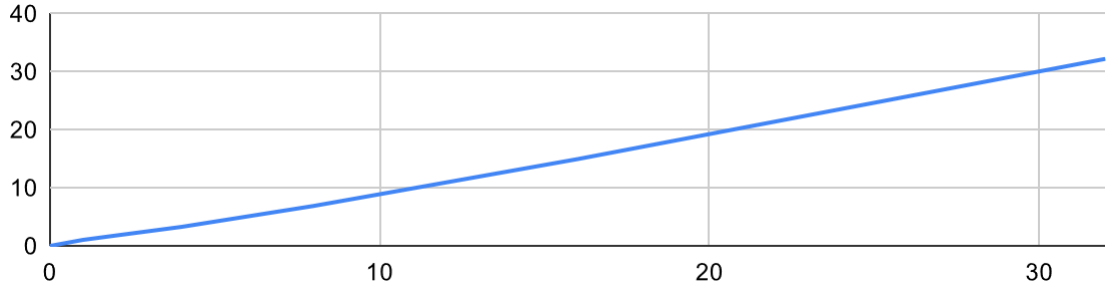

Fig. 6. vmmemWSL CPU core usage total (%)

The graph shown in Figure 7 demonstrates that the SITL process uses on average about 0.8–1% of a CPU core to simulate a single drone depending on the number of drones.

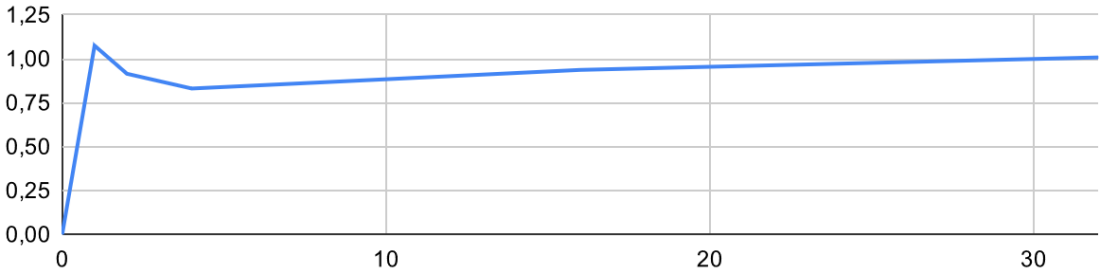

Fig. 7. vmmemWSL CPU core usage per drone (%)

Table 2 presents the results of measuring the time spent by Unity's main thread on updating the model. «Unity Total Time» shows the total duration of one measurement cycle. «Unity Model Time» represents the total time spent updating the model during the measurement period. «Unity % of time» indicates the percentage of the total time used to update the models of all drones. «Unity % of time per drone» shows the average percentage of the total time spent updating the model of a single drone.

Table 2

Drones Number	0	1	2	4	8	16	32
Unity Total Time (MSec)	21107	20469	20633	20282	20333	20447	20332
Unity Model Time (MSec)	0	13	22	42	77	149	319
Unity % of time	0	0,063	0,104	0,207	0,378	0,727	1,568
Unity % of time per drone	0	0,063	0,052	0,052	0,047	0,045	0,049

Figure 8 shows a graph illustrating the linear increase in the time spent updating the models of all drones (vertical axis) depending on the number of drones (horizontal axis).

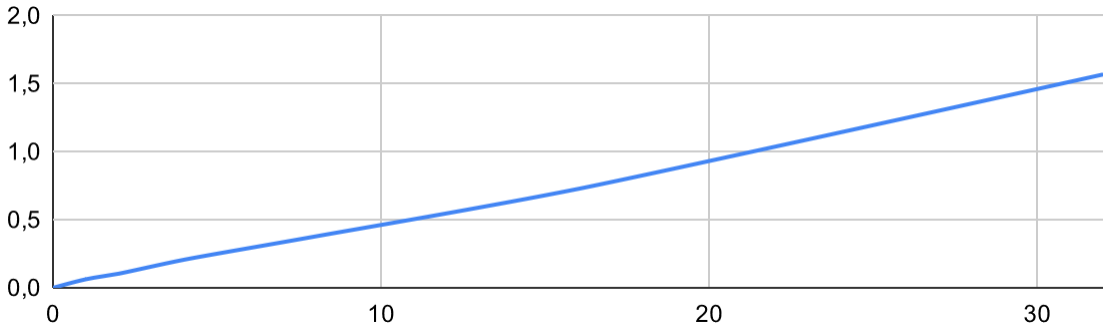

Fig. 8. Unity model update total time (%)

The graph shown in Figure 9 demonstrates that updating the model of a single drone requires approximately 0.05% of the available processing time, depending on the total number of drones.

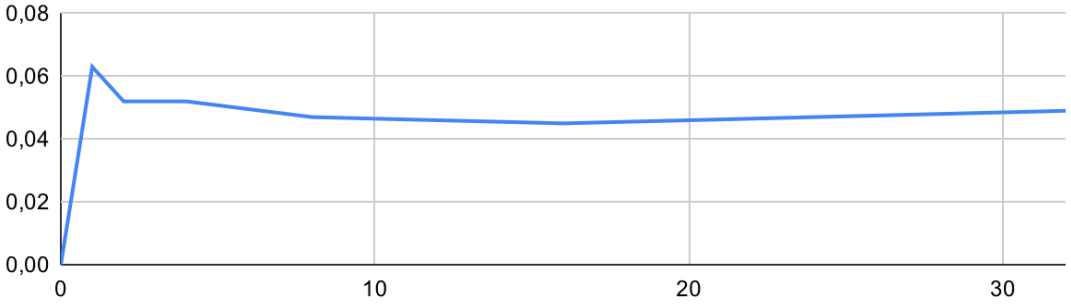

Fig. 9. Unity model update time per drone (%)

The experiment showed that the system's resource usage grows linearly with the number of drones, and that model updates do not become a bottleneck for Unity's main thread, whose resources can therefore be used for visualization.

Discussion

Gamification in this article is addressed only at a conceptual level. The planned game mechanics are mentioned but their interplay and impact on the learning process are not elaborated. The concept has not been validated through a pedagogical experiment, and claims regarding the system's educational effectiveness currently rest solely on the literature review — which is a natural limitation of the present stage of the work. Future studies will examine this direction more thoroughly, including the distinction between the micro game loop (the mission process) and the macro loop, or meta-game loop (the player's overall progression throughout the course of training).

The dangerous-space operation scenarios are referenced as a category — demining, firefighting, environmental tasks — but each requires its own classification, analysis, and dedicated training mission design within the context of gamified UAV swarm operator training.

The environment setup demonstrated in this article is technically demanding. Given that gamification is intended to raise motivation and broaden participation, a incapsulated simulator launch mechanism would be worthwhile. Generally, the authors consider a full ArduPilot deployment on a student's own machine to be a valuable mid-training milestone — one that could be gamified itself.

This article examines only SITL as the simulation backend. Future work may compare it against alternative drone simulation approaches; however, the architecture described and the use of MAVLink make backend substitution largely transparent.

The architectural discussion covers the transport and command layers of the integration. Subsequent work should address the remaining parts of the system, including the implementation of gamification mechanics and the 3D visualisation of drones, terrain, and hazards.

Error handling was deliberately omitted from this article; fault tolerance is a substantial and complex topic that warrants dedicated investigation.

The experiment presented here should also be conducted across different environments in order to establish the upper bound of simulatable drones on a wider range of hardware configurations.

Finally, one of the most significant challenges of this project as a whole will be the integration of all knowledges — requiring deep interdisciplinary expertise spanning education and gamification, computer engineering, and the domain-specific requirements of each mission included in the final product.

Conclusions

The article substantiates the feasibility of a gamified approach to training specialists to operate in dangerous spaces using UAV swarms, and reviews the current scientific literature on gamification, simulation environments, and

the use of UAVs in applied hazardous-areas tasks such as demining, fire monitoring, and environmental control. The analysis revealed the absence of comprehensive solutions that feature a consistent gamified training system.

The technology stack — ArduPilot, SITL, MAVLink, and Unity — is presented, along with the reasoning behind each choice. The Unity ↔ SITL interaction module has been proposed, described, and implemented using a TCP/UDP channel for bidirectional data exchange, with built-in scalability and a unified interface that works equally with both simulated and real unmanned vehicles.

The practical outcome of the work is a technical implementation of the module with a clear separation of responsibilities across the following layers: transport layer, commands layer, messages layer, and vehicle layer — reducing cognitive complexity and improving code maintainability.

The scalability experiment demonstrated that CPU usage grows approximately linearly with the number of simulated vehicles. Specifically, simulating a single drone in the SITL environment consumes roughly 0.8–1% of a single CPU core, while updating one vehicle's model on Unity's main thread requires approximately 0.05% of the available execution time. These results indicate the absence of critical bottlenecks in the proposed architecture and confirm its suitability for scaling.

The proposed solution establishes a technical basis for further development of a full-featured drone swarm simulator with gamification mechanics, including progression, immediate feedback, assessment, and engaging mission-scenario context, to enhance motivation in the training of specialists working in dangerous spaces.

ADDITIONAL INFORMATION

AUTHOR CONTRIBUTIONS

The authors' contributions are as follows: Pavlo Ponomarenko — methodology, architecture, algorithms, experiment, gamification proposal; Vyacheslav Kharchenko — supervising, validating and editing study.

DECLARATION ON THE USE OF GENERATIVE ARTIFICIAL INTELLIGENCE TOOLS

In the preparation of this work, the author used ChatGPT, Claude and Grammarly for grammar and spelling checks, paraphrasing, and rephrasing of individual sentences. After using these tools/services, the authors reviewed and edited the content and take full responsibility for the content of this publication.

REFERENCES

1. Zatserkovnyi V., Nikoliuk I. Using UAVs to detect explosives and create maps in humanitarian demining. Technical sciences and technologies. 2025. № 1 (39). С. 328–345. https://doi.org/10.25140/2411-5363-2025-1(39)-328-345.

2. Romanyshyna L. M., Zdanevich L. V. Innovative domestic and foreign experience in training future mine clearance experts (demining) in the context of digital transformations in society. Scientific Notes LSULS. Pedagogy and Psychology. 2025. № 2 (6). С. 7–12. https://doi.org/10.32782/3041-1297/2025-2-1.

3. Landers R. N., Armstrong M. B., Collmus A. B. How to Use Game Elements to Enhance Learning: Applications of the Theory of Gamified Learning // Serious Games and Edutainment Applications. 2017. P. 457–483. https://doi.org/10.1007/978-3-319-51645-5_21.

4. Triantafyllou S. A., Georgiadis C. K., Sapounidis T. Gamification in education and training: A literature review. International Review of Education. 2025. Vol. 71, № 1. https://doi.org/10.1007/s11159-024-10111-8.

5. Cardona-Reyes H., Trujillo-Espinoza C., Arévalo-Mercado C. A., Muñoz-Arteaga J. Training of Drone Pilots through Virtual Reality Environments under the Gamification Approach in a University Context. Interaction Design and Architecture(s). 2021. № 49. P. 64–83. https://doi.org/10.55612/s-5002-049-004.

6. Nikolaiev M., Novotarskyi M. Comparative review of drone simulators. Information, Computing and Intelligent Systems. 2024. № 4. P. 79–99. https://doi.org/10.20535/2786-8729.4.2024.300614.

7. Milokhin M. Multi-sensor technologies for the detection of explosive objects in mining using UAV swarms. Measuring and computing devices in technological processes. 2024. № 78. С. 348–356. https://doi.org/10.31891/2219-9365-2024-78-40.

8. Korniienko D., Kharchenko V. Intelligent UAV-UGV-SN-based system for monitoring and preventing forest fires // Intelligent Systems Workshop at CoLInS 2025. 2025. https://doi.org/10.31110/COLINS/2025-2/010.

9. Gu Q., Michanowicz D. R., Jia C. Developing a Modular Unmanned Aerial Vehicle (UAV) Platform for Air Pollution Profiling. Sensors. 2018. Vol. 18, № 12. Article 4363. https://doi.org/10.3390/s18124363.

10. Koubaa A., Allouch A., Alajlan M., Javed Y. Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey. IEEE Access. 2019. Vol. 7. P. 87658–87680. https://doi.org/10.1109/ACCESS.2019.2924410.

11. Mitter M., Landa J., Pisařovic I., Procházka D., Varga M., Dařena F. 3D geospatial data

visualization in VR. European Journal of Business Science and Technology. 2025. Vol. 11, № 1. P. 129–140. https://doi.org/10.11118/ejobsat.2025.005.

12. Rantanen T., Julin A., Virtanen J.-P., Hyyppä H., Vaaja M. T. Open geospatial data integration in game engine for urban digital twin applications. ISPRS International Journal of Geo-Information. 2020. Vol. 8, № 10. Article 310. https://doi.org/10.3390/ijgi12080310.

13. Kyropoulou D., Karalis P., Dotsika E., Rizou F., Raptis I., Drosou A., Tzovaras D., Mazarakis Ainian A., Kolofotia E. A 3D geospatial platform and AI tour application for visualizing archaeological, analytical, and conservation data: The case of the Vryokastro archaeological ensemble, Kythnos Island. Journal of Archaeological Science: Reports. 2025. https://doi.org/10.1016/j.jasrep.2025.105480.

14. ArduPilot development site. ArduPilot Development Team. URL: https://ardupilot.org/dev/

15. QGroundControl guide. QGroundControl Development Team. URL: https://docs.qgroundcontrol.com/Stable_V5.0/en/qgc-user-guide/

16. MAVLink developer guide. MAVLink Community. URL: https://mavlink.io/en/

Павло ПОНОМАРЕНКО, Вячеслав ХАРЧЕНКО
Національний аерокосмічний університет «Харківський авіаційний інститут»

ІНТЕГРАЦІЯ ARDUPILOT SITL ЯК ВІРТУАЛЬНОГО СИМУЛЯТОРА ДРОНА В ГЕЙМІФІКОВАНІЙ ПІДГОТОВЦІ

Нові освітні технології, а також виклики сучасного світу спонукають до пошуку шляхів покращення і розвитку методів підготовки фахівців з подолання небезпечних просторів за допомогою безпілотних ройових систем. За допомогою таких технологій як Unity та ArduPilot можна створити симульоване навчальне середовище, яке завдяки гейміфікаційним механікам надає відчуття винагороди та виклику при навчанні взаємодії з роєм дронів, який вимагає глибоких знань в багатьох дисциплінах. Предметом дослідження є модуль взаємодії ігрового рушія Unity та симуляційного рушія ArduPilot SITL. Об'єкт дослідження: програмна система для підготовки фахівців з подолання небезпечних просторів. Метою статті є запропонувати шлях інтеграції ігрового рушія Unity та симуляційного рушія ArduPilot SITL в контексті гейміфікованої підготовки фахівців з роботи з небезпечними просторами. Для реалізації поставленої мети у статті вирішено наступні завдання: сформулювати концепцію застосунку, що моделює місії з роями дронів у небезпечних просторах та забезпечує прогресію й оцінювання через гейміфікаційні механіки; обґрунтувати вибір технологічного стеку, зокрема ArduPilot, SITL, MAVLink, Unity; описати архітектуру взаємодії Unity ↔ SITL, а саме TCP/UDP-комунікацію, обробку повідомлень MAVLink, а також розподіл відповідальності між шарами та компонентами. Результати дослідження: обґрунтовано архітектуру та реалізовано модуль взаємодії між Unity та ArduPilot SITL на основі протоколу MAVLink, що забезпечує двосторонній обмін даними та масштабовану інтеграцію кількох симульованих апаратів у межах єдиного застосунку; для оцінки масштабованості системи проведено експеримент з аналізу використання процесорних ресурсів та часу оновлення моделі залежно від кількості симульованих дронів. Отримані результати показали майже лінійну залежність навантаження від кількості апаратів та відсутніть вузького місця в основному потоці Юніті.

Ключові слова: гейміфікація, БПЛА, небезпечні простори, рої дронів, Unity, ArduPilot, MAVLink