МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації, електроніки та телекомунікацій

Кафедра аерокосмічних систем управління

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри АКСУ

_____ Ю. В. Мельник

«_____»_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"БАКАЛАВР"

Тема: «Комп’ютерний тренажер оператора БПЛА»

Виконавець: Гринько Роман Ігорович

Керівник: к.т.н, доцент, Безкоровайний Юрій Миколайович

Нормоконтролер: к.т.н, доцент, Дивнич Микола Полікарпович

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет аеронавігації електроніки та телекомунікацій

Кафедра аерокосмічних систем управління

Спеціальність 151 «Автоматизація та комп’ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ

Завідувач кафедри АКСУ

_____ Ю. В. Мельник

"_____"_____ 2023р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

студента Гринька Романа Ігоровича

1. Тема роботи: «Комп’ютерний тренажер оператора БПЛА» затверджена наказом ректора від 13.04.2023 р. № 507/ст
2. Термін виконання: з 22.05.2023 по 25.06.2023 р.
3. Вихідні дані: дослідити аналітичний огляд літературних джерел та статей з тематики диплому; роглядаються загальні поняття про коп’ютерні тренажери; досліджується архітектура комп’ютерних тренажерів; розробляється власний комп’ютерний тренажер оператора БПЛА на основі вивчених матеріалів.
4. Зміст пояснювальної записки: Теоретичні основи комп’ютерного тренажера.; Проєктування комп’ютерного тренажера.; Розробка комп’ютерного тренажера.

5. Календарний план-графік:

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Підпис керівника
1.	Уточнення постановки задачі	28.03.2023	
2.	Збір інформації	29.03.2023	
3.	Аналіз літературних джерел	08.05.2023	
4.	Розробка детального змісту дипломної роботи	12.05.2023	
5.	Вступ	14.05.2023	
6.	Розділ 1. Теоретичні основи комп'ютерного тренажера.	17.05.2023	
7.	Розділ 2. Проєктування комп'ютерного тренажера.	23.05.2023	
8.	Розділ 3. Розробка комп'ютерного тренажера.	25.05.2023	
9.	Оформлення роботи	31.05.2023	

6. Дата видачі завдання: «__» _____ 2023 р.

Керівник дипломної роботи: _____ Безкоровайний Ю.М.

Завдання прийняв до виконання _____ Гринько Р. І.

РЕФЕРАТ

Текстова частина роботи: 63 стор., 14 рис.

Об’єкт дослідження – підготовка операторів безпілотних літальних апаратів (БПЛА).

Предмет дослідження – Комп’ютерний тренажер оператора безпілотного літального апарату (БПЛА).

Мета роботи – Розробити комп’ютерний симулятора оператора БПЛА для покращення якості підготовки операторів та зниження ризику виникнення непередбачених ситуацій під час експлуатації БПЛА.

Методи дослідження – аналіз наукової літератури, математичне моделювання для створення комп’ютерної моделі тренажера оператора БПЛА; Експеримент (відтворити умови реальної роботи операторів БПЛА та дослідити ефективність використання комп’ютерних симуляторів), рекомендації від експертів у галузі створення тренажерів операторів БПЛА.

У роботі було визначено основні завдання, що були розв’язані для досягнення поставленої мети, а саме:

1. Аналіз літературних джерел та практичних досвідів використання комп’ютерних тренажерів у підготовці операторів БПЛА.
2. Розроблено архітектуру комп’ютерного симулятора оператора БПЛА та визначено його функціональні можливості.
3. Розроблено програмне забезпечення комп’ютерного симулятора оператора БПЛА, яке дозволяє відтворювати різні сценарії використання БПЛА та проводити тренувальні сесії з операторами.
4. Проведено тестування розробленого комп’ютерного симулятора на відповідність вимогам та оцінено його ефективність у підготовці операторів БПЛА.

Отже, розробимо комп’ютерний симулятор оператора БПЛА, який дозволяє підвищити якість підготовки операторів та знизити ризик виникнення непередбачених ситуацій під час експлуатації БПЛА.

КОМП'ЮТЕРНИЙ ТРЕНАЖЕР, КЕРУВАННЯ, БЕЗПІЛОТНИЙ ЛІТАЛЬНИЙ АПАРАТ (БПЛА), МОДЕЛЮВАННЯ, СИМУЛЯЦІЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, VR, AR, GUI.

ЗМІСТ

ВСТУП

У зв’язку зі швидким розвитком беспілотної авіації та збільшенням обсягів її використання, особливо у військових цілях, дедалі більшої актуальності набуває питання підготовки кваліфікованих операторів безпілотних літальних апаратів (БПЛА). Одним із основних етапів підготовки таких операторів є тренування навичок управління БПЛА на комп’ютерному тренажері.

Метою даної дипломної роботи є розробка комп’ютерного тренажера для оператора БПЛА з метою підвищення ефективності та якості тренування. Для досягнення цієї мети передбачено вирішення наступних завдань: аналіз існуючих систем комп’ютерного тренування для операторів БПЛА; розробка програмного забезпечення для тренажера; вивчення ефективності та результативності тренажера в процесі тренування операторів БПЛА.

В результаті дослідження має бути створений комп’ютерний тренажер для оператора БПЛА, який забезпечить якість та ефективність тренування навичок управління БПЛА на рівні вимог, що ставляться до сучасних систем тренування.

Робота включає вивчення сучасних тренажерів, таких як DJI Flight Simulator, аналіз їх можливостей, порівняння їх з власним розробленим тренажером та чи є відповідність сучасним вимогам та стандартів безпеки. Робота також повинна дати поштовх до вивчення та розробку майбутнього інноваційного продукту, який може стати важливим кроком у підготовці операторів БПЛА та забезпечити безпечні та ефективні умови їх роботи.

Окрім цього, робота має на меті зануритися в світ симуляторів і розуміння їх функціонування. Шляхом розробки цього тренажера на основі мови програмування Python, отримаю практичні навички в розробці комп’ютерних програм для симуляції реальних процесів.

Результатами проведеного дослідження, розробки та аналізу проєкту повинні підтвердити можливість успішного створення комп’ютерного тренажера для операторів БПЛА. Отримані навички в процесі розробки цього проєкту виявляють потенціал для їх використання в подальших дослідженнях та розробках симуляторів та тренажерів для різних областей.

Загалом, ця робота сприяє розумінню принципів функціонування та розробки комп’ютерних тренажерів, а також підкреслює важливість тренування операторів БПЛА для забезпечення безпеки та ефективності їх роботи. Цей проєкт є початковим кроком в напрямку розробки більш складних та реалістичних тренажерів, які в подальшому можуть мати велике значення в навчанні та підготовці операторів БПЛА.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ КОМП'ЮТЕРНОГО СИМУЛЯТОРА

1.1 Вступ до комп'ютерних тренажерів для навчання операторів БПЛА.

Історія розвитку комп'ютерних тренажерів для навчання операторів БПЛА починається з появи перших безпілотних літальних апаратів у 1960-х роках. У той час БПЛА використовувалися в основному для військових потреб в якості мішеней для тренування пілотів, а керування здійснювалося оператором по радіоканалу. З розвитком технологій з'явилися більш продвинуті методи віддаленого керування БПЛА з використанням ЕОМ, що вимагало від операторів нових навичок та вмінь.

В тих же роках для майбутніх пілотів почали розробляти і впроваджувати так звані авіасимулятори або комп'ютерні тренажери, які дозволяли імітувати різні ситуації та умови, з якими може зіткнутися пілот повітряного судна під час польоту. На рис. 1 зображенний авіасимулятор для пілотів тих часів.

Рис. 1. Історичний авіасимулятор

Також через якийсь час з’явилися комп’ютерні тренажери та симулятори для операторів БПЛА. Ці симулятори були спеціально розроблені для навчання операторів управлінням безпілотними літальними апаратами.

У 1990-х роках комп’ютерні тренажери для навчання операторів БПЛА стали більш поширеними. Такі тренажери були розроблені для військових потреб, але поступово використання безпілотників розширилося в багатьох галузях авіації, і на сьогоднішній день БПЛА здобувають значне визнання через свої застосування в багатьох сферах.

Віриться, що БПЛА виконують завдання, які є небезпечними, нудними або простими [1]. Крім використання в військових місіях, управління надзвичайними ситуаціями, географічного моніторингу, журналістики, моніторингу сільськогосподарських угідь, прецизійного землеробства, археології та доставки товарів, у майбутньому очікується їх використання у таксі. Використання БПЛА для таких застосувань є відносно недорогим порівняно з пілотованими повітряними засобами. В багатьох сценаріях використовується кілька БПЛА для успішного виконання місій.

У таких застосуваннях однією з ключових особливостей є координація групи БПЛА, а також їхня безпека [2]. Однак робота з групами БПЛА є складним викликом. Останні дослідження показали можливість атак на керування та передачу даних, перешкоджання сигналів управління та перехоплення. Для синхронізування групи БПЛА використовувати традиційну бездротову мережу неможливо, адже є вимоги до потужності, стабільності сигналу, споживанням енергії та протоколами зв’язку.

БПЛА мають різні форми та розміри, і їх аеродинамічні моделі варіюються залежно від типів. Недолік в аеродинамічному профілі БПЛА ставить під загрозу вартість, конфіденційність та успішність місії з їх використанням, тому розробка та функціональність БПЛА є складними та коштовними завданнями. Тому до визначення конструкції або запуску їх в повітря необхідно використовувати тестову платформу або симулятор, що відповідає стандартам життєздатності

БПЛА. Надалі в дипломі я буду використовувати терміни "БПЛА" і "дрони" взаємозамінно. Одна з найважливіших ідей, що підтримує використання дронів, - це концепція Інтернету речей (IoT). Вона відповідає визначенню Міжнародного телекомунікаційного союзу (ITU) про IoT: з’єднання будь-чого, будь-коли, будь-де для будь-кого.

Технологія, яка забезпечує дрони можливостями Інтернету речей, існує сьогодні. Поєднані технології спрямовані на надання аналітики майже в реальному часі на просторово-часові дані, зібрані протягом тривалого часу. Зі зростанням кількості мережевих пристроїв та об’єднанням кількох вже присутніх технологій, зростають і проблеми. Деякі з них включають безпеку, якість обслуговування (QoS) та розробку протоколів для різних рівнів. Для вирішення таких проблем були розроблені деякі симулятори та емулятори. При проектуванні симулятора розробники повинні мати на увазі аеродинаміку, тривимірний польот та мережевий зв’язок дронів. В випадках застосувань БПЛА, віддається перевага групам дронів, а не одному потужному дрону, через кілька можливих причин. По-перше, можливості одного БПЛА для сприйняття та активізації обмежені, тоді як роями БПЛА можна одночасно збирати інформацію з кількох місць та використовувати інформацію, отриману з різних точок, для побудови моделей, які можуть бути використані для прийняття рішень. Крім того, кілька БПЛА можуть одночасно застосовувати сили в різних місцях для виконання дій, які можуть бути дуже складними для одного БПЛА. По-друге, ефективність, вартість і час виконання місій, таких як дослідження та пошук цілей, можуть бути різко скорочені за допомогою роїв БПЛА.

Для деяких завдань використання групи недорогих дронів, а не одного БПЛА, також може бути економічно вигідним. З точки зору надійності підхід з використанням груп-БПЛА пропонує резервні рішення, що забезпечують більшу стійкість до помилок та більшу гнучкість, включаючи можливість переконфігурації в разі відмов окремих засобів. Безпека даних та сигналів керування в межах цих мереж має вирішальне значення для конфіденційності та успіху місії. Крім того, динамічний характер дронів ставить додаткові виклики,

наприклад, якщо один вузол виходить з ладу, потрібна самоорганізація, а якщо цього не робити, це може призвести до поганих наслідків. Складнощі самоорганізації роблять цю проблему цікавим дослідницьким напрямом, який може бути додатково досліджений за допомогою симуляції.

Сьогодні комп'ютерні тренажери для навчання операторів БПЛА постійно удосконалюються та оновлюються з використанням новітніх технологій. Вони забезпечують більш повне моделювання процесів, що відбуваються в реальному світі, включаючи моделювання фізики політного процесу, різних типів БПЛА та їх особливостей, впливу зовнішніх факторів, таких як погодні умови, та інших аспектів роботи БПЛА.

Наприклад, використання віртуальної реальності (VR) дозволяє створювати ще більш реалістичні умови для навчання. VR дозволяє створювати іммерсивне середовище, в якому оператор може взаємодіяти з віртуальним БПЛА та його оточенням у віртуальному просторі, що максимально наближається до реальних умов. В цілому, за допомогою віртуальної реальності комп'ютерні тренажери для операторів БПЛА стають більш ефективними та ефективними засобами навчання, дозволяючи операторам відпрацьовувати навички та реагувати на різні ситуації у безпечному та контрольованому середовищі.

1.2 Огляд понять "тренажер" та "симулятор"

Тренажери та симулятори в авіації є важливими засобами навчання та тренування для пілотів та операторів БПЛА. Вони дозволяють зменшити ризики, пов'язані з навчанням на реальних повітряних апаратах, та забезпечують можливість вивчення та практичного вдосконалення навичок у контрольованому середовищі.

Тренажер - це спеціальний пристрій, що використовується для навчання та тренування певних навичок та навчальних завдань. У випадку авіації, тренажер може бути віртуальним пристроєм, що імітує різні ситуації в повітряному просторі, та дозволяє тренувати реакцію пілота на різні випадки.

Симулятор - це комп’ютерна програма, яка імітує роботу реального об’єкта, процесу або системи. У контексті авіації, симулятор може імітувати різні випадки польотів, дозволяючи пілотам вивчити і вдосконалювати свої навички, не ризикуючи життям.

Терміни "тренажер" та "симулятор" часто використовуються як синоніми, проте між ними є певні відмінності. Тренажер, прилад або система, призначена для підготовки людини до виконання конкретної діяльності. Він може бути розроблений для тренування навичок пілотування, водіння, медицинської практики та інших видів професійної діяльності. Тренажери зазвичай забезпечують можливість відтворення реальних умов, з якими стикається людина при виконанні певної діяльності. Симулятори це комп’ютерні програми або пристрої, які імітують деякі реальні умови або події.

Для данної дипломної роботи, визначення "симулятор" підходить краще. Оскільки розробляється комп’ютерна програма для імітації політів безпілотних літальних апаратів, що дозволяє тренувати операторів БПЛА, а також тестувати нові рішення та алгоритми у безпечному середовищі, що імітує реальні умови.

1.3 Види комп’ютерних симуляторів для операторів БПЛА

В загальному професійні симулятори поділяються на три категорії: процедурні, комплексні, тактичні.

Процедурні симулятори - зазвичай використовуються для навчання процедур та протоколів, пов’язаних з експлуатацією та управлінням БПЛА, таких як зльот, посадка, поворот, відстеження цілей, виконання завдань і т.д.

Процедурні симулятори можуть бути більш економічно ефективним рішенням для навчання, оскільки вони не вимагають великої кількості апаратного забезпечення та інфраструктури, що зазвичай потрібні для повноцінних симуляторів. Вони можуть бути використані для навчання та оцінки процедури на різних стадіях навчання операторів БПЛА, що дозволяє досягти більшої ефективності навчання.

Однак, процедурні симулятори мають свої обмеження, оскільки вони не можуть повністю замінити повноцінний симулятор для навчання операторів БПЛА. Вони зазвичай не мають достатньої реалістичності, тому корисні тільки на початкових етапах навчання.

Комплексні симулятори - для операторів БПЛА є найбільш реалістичними тренажерами, які забезпечують найбільш повне моделювання процесів, що відбуваються в реальному світі. Вони включають в себе комп'ютерне моделювання середовища, в якому знаходиться БПЛА, сценарії місій, симуляцію зовнішніх факторів (наприклад, погодних умов) та інші аспекти, які впливають на роботу БПЛА.

Комплексні симулятори можуть бути створені як для окремих типів БПЛА, так і для групи різних моделей. Вони забезпечують можливість моделювання поведінки БПЛА в різних умовах, таких як зміна швидкості вітру, опадів, температури повітря та інших факторів.

Одним з прикладів комплексного симулятора для БПЛА є Global Hawk Simulator, який використовується в американській армії для навчання операторів БПЛА Global Hawk. Цей симулятор забезпечує повне моделювання процесів, що відбуваються в реальному світі, і дозволяє операторам проводити місії в різних умовах, що забезпечує високу якість навчання.

Комплексні симулятори мають свої обмеження, такі як високі вимоги до обладнання, яке використовується для їх роботи, і високі витрати на їх створення та підтримку. Однак вони є найбільш ефективними інструментами для навчання операторів БПЛА і дозволяють підвищувати рівень їх кваліфікації.

Тактичні симулятори - цей тип симуляторів спрямований на координацію взаємодії людей. Такі симулятори надають можливість операторам навчатися вирішувати різноманітні тактичні завдання та співпрацювати з іншими підрозділами.Тактичні симулятори для операторів БПЛА можуть відтворювати різні сценарії, такі як збори розвідувальної інформації, моніторинг підземних структур, патрулювання морських територій та інше. Оператори можуть

використовувати різні типи БПЛА, що дозволяє їм навчатися працювати з різними моделями та конфігураціями.

По суті тактичні симулятори дуже схожі на мультиплеєрні ігри. Але це не просто клієнт-серверний додаток, у зв’язці можуть бути кілька серверів, які відповідають за різні сфери: на одному відбувається лише симуляція фізики, на іншому — погоди, на третій — чогось ще, і всі вони взаємодіють між собою.

Також варто врахувати, що окрім зазначених типів симуляторів, можуть існувати гібридні рішення, які поєднують риси кількох типів. Наприклад, симулятор може поєднувати процедурні функціональності з комплексним моделюванням середовища або тактичними сценаріями для надання більш широкого спектру навчальних можливостей.

Враховуючи різноманіття типів симуляторів та їхніх можливостей, оператори БПЛА можуть використовувати їх для різних цілей, таких як навчання основним процедурам, вдосконалення навичок управління, підвищення рівня кваліфікації та розробка тактичних стратегій.

1.4 Огляд використовуваних технологій у створенні комп’ютерних симуляторів.

Створення комп’ютерних симуляторів потребує використання різноманітних технологій з різних галузей, таких як програмування, графіка, фізика, математика, штучний інтелект та інші. Графічні движки: Графічні движки дозволяють створювати реалістичні візуальні ефекти, моделювати фізичні процеси, такі як взаємодія об’єктів та їхній рух у середовищі. Найпопулярніші графічні движки включають Unity, Unreal Engine, CryEngine та інші.

Фізичні движки: Фізичні движки дозволяють моделювати фізичні процеси, такі як гравітація, сили тертя, взаємодія тіл та інші. Найбільш відомі фізичні движки - PhysX, Havok, Bullet та інші.

Штучний інтелект: Штучний інтелект дозволяє створювати імітацію поведінки людей або інших живих організмів у віртуальному світі. Найбільш

відомі фреймворки для розробки штучного інтелекту - TensorFlow, PyTorch, Caffe, Theano, Keras та інші.

Програмування: Для створення симуляторів необхідно володіти мовами програмування, такими як C++, C#, Java, Python та інші.

Моделювання: Моделювання дозволяє створювати 3D-моделі об’єктів та середовища для віртуальної реальності. Найбільш відомі програми для моделювання - Blender, 3ds Max, Maya, ZBrush та інші.

Віртуальна реальність (VR) і доповнена реальність (AR): Ці технології дозволяють створювати іммерсивне оточення, в якому користувач може взаємодіяти з віртуальними об’єктами або спостерігати взаємодію віртуальних об’єктів з реальними. Для розробки VR- та AR-симуляторів використовуються такі технології, як Unity, Unreal Engine, Vuforia, ARCore та інші.

Системи баз даних: Симулятори можуть використовувати системи баз даних для зберігання, організації та обробки даних, що використовуються в симуляції. Наприклад, для зберігання стану об’єктів, налаштувань і статистики можуть використовуватися реляційні бази даних, такі як MySQL, або NoSQL бази даних, такі як MongoDB.

Комунікаційні протоколи: У симуляторах можуть використовуватися комунікаційні протоколи для обміну даними між різними компонентами системи або для зв’язку з іншими симуляторами або зовнішніми системами. Наприклад, для передачі даних через мережу можуть використовуватися протоколи TCP/IP, UDP або WebSocket.

Інтерфейс користувача: Для взаємодії користувача з симулятором використовуються графічні інтерфейси користувача (GUI) або командний рядок. Інтерфейси можуть містити елементи, такі як кнопки, поля введення, вікна, що дозволяють користувачеві налаштовувати параметри симуляції, спостерігати результати та взаємодіяти з віртуальним середовищем.

Збереження та відновлення стану: Деякі симулятори надають можливість зберігати поточний стан симуляції та відновлювати його в майбутньому. Це

дозволяє користувачам продовжувати симуляцію з певного моменту часу або повторювати певні експерименти.

Можливості паралельного обчислення: Деякі симулятори можуть використовувати паралельні обчислення для прискорення обробки даних та забезпечення високої швидкодії. Для цього використовуються технології, такі як CUDA, OpenMP, MPI та інші.

Зазначені технології представляють лише деякі аспекти, які використовуються у створенні комп'ютерних симуляторів. Вибір конкретних технологій залежить від потреб проєкту, його масштабу, цілей та обмежень. Комбінація різних технологій може забезпечити більш реалістичну та ефективну симуляцію різних процесів у віртуальному середовищі.

Наприклад: X-Plane - це популярний авіасимулятор, що використовує технологію реалістичного моделювання польоту. Він базується на технології OpenGL, що дозволяє отримати високу якість графіки та швидку обробку графічних об'єктів. X-Plane SDK - це набір інструментів та бібліотек для розробки плагінів та додаткових модулів для авіасимулятора X-Plane, включаючи можливість розробки власних літаків.

1.5 Огляд можливостей та обмежень комп'ютерних тренажерів у навчанні операторів БПЛА.

Комп'ютерні тренажери для навчання операторів БПЛА можуть мати різні можливості та обмеження.

Можливості:

- Можливість тренуватися в управлінні БПЛА в різних сценаріях, включаючи надзвичайні ситуації, що може бути важко забезпечити в реальних умовах.
- Можливість проводити тренування в зручний для оператора час та місце, що дозволяє економити час та кошти на подорожі та оренду обладнання.
- Можливість моніторити та аналізувати виконання оператора під час тренування, що дозволяє виявляти помилки та недоліки у техніці управління БПЛА та вносити необхідні покращення.

- Можливість інтеграції з іншими симуляторами та системами навчання, що дозволяє створювати комплексні тренажери та розширювати можливості навчання.

Обмеження:

- Відсутність фізичних відчуттів під час тренування може призвести до того, що оператор не зможе повністю передати реальний досвід управління БПЛА.
- Відсутність повної ідентичності віртуального середовища та реального може призвести до того, що оператор може не змогти повністю оцінити ситуацію та зробити правильне рішення.
- Висока вартість створення та підтримки комп'ютерних тренажерів може бути обмеженням для невеликих компаній або індивідуальних користувачів.

У цілому, комп'ютерні тренажери є ефективними засобами для навчання операторів БПЛА, але наприклад, недостатньо точне відтворення фізичних параметрів польоту може призвести до того, що пілот-оператор не зможе повністю осягнути навички управління БПЛА у реальних умовах.

Також варто відзначити, що тренажер може бути досить дорогим для придбання та підтримки, особливо для менших підприємств та навчальних закладів. Більшість тренажерів вимагає певних технічних характеристик комп'ютера та додаткових пристроїв, які можуть збільшувати загальну вартість системи.

Також стоїть відзначити, що комп'ютерні тренажери не можуть повністю замінити реальний політ БПЛА, оскільки навіть найсучасніші тренажери не враховують всіх випадкових факторів, які можуть виникнути в реальних умовах польоту, таких як погодні умови, технічні поломки, або непередбачувана поведінка інших об'єктів у повітрі.

Незважаючи на ці обмеження, комп'ютерні тренажери залишаються незамінними в навчанні операторів БПЛА, оскільки вони дозволяють зменшити ризик для людей та зберегти кошти на тренувальних польотах. Тренажери також

дозволяють операторам БПЛА вдосконалювати свої навички та вчитися новим технікам без ризику для життя та здоров’я людей або руйнування майна.

1.6 Аналіз існуючих комп’ютерних симуляторів для навчання операторів БПЛА.

Проаналізувавши існуючі комп’ютерні симулятори(тренажери) для навчання операторів БПЛА можемо зрозуміти, які можливості та обмеження мають наявні продукти та які їх характеристики мають більший вплив при створенні ефективного тренажеру.

Одним з найпопулярніших симуляторів для навчання операторів БПЛА є DJI Flight Simulator [3]. Цей симулятор розроблений компанією DJI, яка є провідним виробником безпілотних літальних апаратів. На рис. 2 і 3 зображено його інтерфейс та вікно симуляції.

Рис. 2. Зображення головного меню DJI Flight Simulator

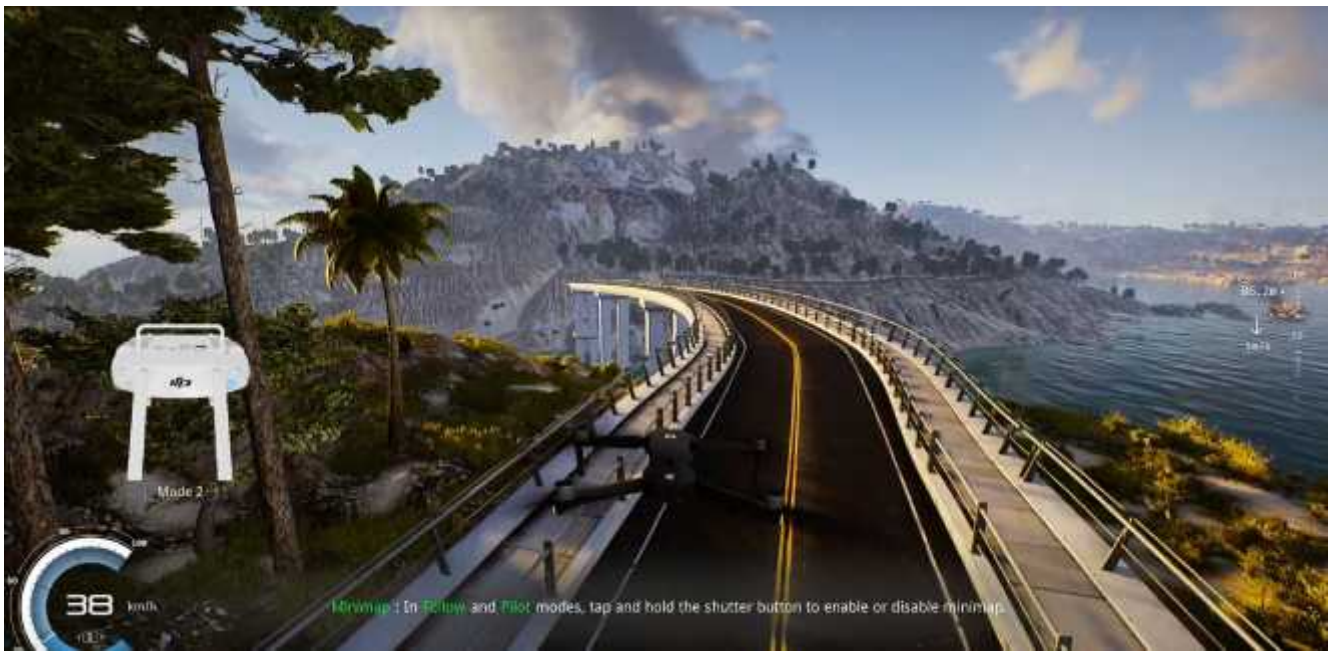

Рис. 3. Зображення вікна симуляції DJI Flight Simulator

Однією з ключових особливостей DJI Flight Simulator є його здатність точно відтворювати умови польоту. Він надає широкий спектр налаштувань та параметрів, які дозволяють користувачу створювати різні сценарії польоту. За допомогою цих налаштувань можна моделювати різні погодні умови, поверхні місцевості та інші фактори, що впливають на політ.

Такий рівень налаштовуваності дозволяє операторам БПЛА отримати більш реалістичний досвід навчання та тренування.

DJI Flight Simulator також має різні режими навчання, які дозволяють операторам вдосконалити свої навички. Наприклад, в ньому є тренування польоту на різних висотах та умовах, що дозволяє користувачам вправлятися в керуванні на різних висотних рівнях. Крім того, симулятор пропонує тренування зйомки фото та відео з повітря, що може бути корисним для операторів, які займаються аерозйомкою.

Окрім моделювання польоту та навчання, DJI Flight Simulator також надає засоби для аналізу та вивчення даних. Оператори можуть переглядати записи своїх польотів, аналізувати додаткові даних, такі як швидкість, висота,

стабілізація та інші параметри. Це допомагає операторам оцінювати свої навички та вдосконалювати їх з кожним тренуванням.

Загалом, DJI Flight Simulator є потужним інструментом для навчання та тренування операторів БПЛА. Він поєднує в собі реалістичну моделювання польоту, широкі налаштування та режими навчання, що дозволяють операторам отримати цінний досвід безпечного та ефективного керування безпілотними літальними апаратами DJI.

Іншим прикладом симулятора для навчання операторів БПЛА є X-Plane [5]. Цей симулятор розроблений для навчання пілотів, але також може бути використаний для навчання операторів БПЛА. X-Plane має велику кількість налаштувань та параметрів, які дозволяють створювати різні сценарії для тренування, такі як польот у різних погодних умовах, різних типах літаків та інші.

Іншим прикладом симулятора для навчання операторів БПЛА є X-Plane. Цей симулятор розроблений компанією Laminar Research і здатний моделювати різні типи повітряних суден та сценарії польоту. Відповідні скріншоти з цього симулятора зображені на рис. 4-5.

Рис. 4. Зображення вікна симуляції X-Plane 12

Рис. 5. Зображення вікна симуляції X-Plane 12

Одна з ключових особливостей X-Plane - це його реалістична фізична модель польоту. Він використовує динамічну модель польоту, яка детально враховує фізичні принципи, такі як аеродинаміка, гравітація, сили тяги та опору. Це дозволяє операторам отримати відчуття реального польоту та відтворити різні сценарії, включаючи політ у різних погодних умовах, висотах та типах повітряних суден.

X-Plane також має широкі можливості налаштування. Він дозволяє користувачам налаштовувати різні параметри польоту, такі як швидкість, тримачі, погодні умови, освітлення та інші. Це дозволяє створювати різноманітні сценарії для тренування, відповідно до потреб операторів БПЛА.

Окрім цього, X-Plane має велику базу даних з аеропортів, місцевості та навігаційних точок по всьому світу. Це дозволяє операторам тренуватися у реальних локаціях та навчатися виконувати точні маневри та процедури.

X-Plane також підтримує різні додаткові плагіни та модифікації, які можуть розширити його можливості. Це дозволяє операторам налаштовувати симулятор

під свої потреби та додавати додаткові функції, які поліпшують навчальний процес.

Загалом, X-Plane є потужним та гнучким симулятором, який може бути використаний для навчання операторів БПЛА. Його реалістична фізична модель, широкі можливості налаштування та реалістична база даних роблять його популярним вибором для навчання польоту та операцій з БПЛА.

Крім того, існують спеціалізовані симулятори для навчання операторів БПЛА, які розроблені під конкретні моделі БПЛА. Наприклад, симулятор для навчання операторів Phantom 4, розроблений компанією DJI.

РОЗДІЛ 2
ПРОЄКТУВАННЯ КОМП'ЮТЕРНОГО ТРЕНАЖЕРА

2.1 Вимоги до комп'ютерного тренажера

У симуляторів БПЛА є три цілі їх використання: оцінка нових технологій, недороге навчання та науково-дослідна робота. Моделювання та симуляція БПЛА - це складне завдання, яке вимагає розрахунку багатьох параметрів. У порівнянні з рухом наземного транспорту, рух БПЛА не обмежений 2D простором. У 3D моделі руху БПЛА розрахунок різних аеродинамічних коефіцієнтів становить додаткове виклик. Симулятори БПЛА використовуються для створення складних математичних моделей, які максимально наближаються до реальної фізичної системи та скорочують час і витрати на експерименти [5]. Є багато спільного між симуляторами БПЛА та традиційними літаковими симуляторами, наприклад, обидва застосовуються в проектуванні літаків та механіки польоту та дозволяють операторам тренуватися у небезпечних ситуаціях. Тому в даній роботі я буду використовувати термін "симулятор БПЛА" та "літаковий симулятор" взаємозамінно.

Хорошому симулятору БПЛА потрібні такі характеристики: легкість використання, можливість запуску на загальнопризначеному обладнанні, менша складність та простий графічний інтерфейс користувача. Для створення ефективного комп'ютерного тренажера для навчання операторів БПЛА, необхідно врахувати наступні вимоги:

Модель динаміки польоту: БПЛА керуються користувачем, або за допомогою автопілота на борту, або за допомогою віддалених комп'ютерів. Розуміння аеродинамічних сил, що виникають під час польоту, має велике значення з багатьох причин.

По-перше, польотна конструкція повинна витримувати діючі сили. По-друге, розуміння аеродинамічних сил сприяє ефективному проектуванню повітряного судна. По-третє, керування повітряним судном здійснюється пілотом, що вимагає розуміння динаміки повітряного судна. На щастя, для розробників літакових симуляторів є багато існуючих рівнянь, пов’язаних з аеродинамікою, які добре вивчені і можуть бути застосовані для моделювання динаміки БПЛА.

Модель системи: Паралельно з цим, достовірність симулятора дрона залежить від точної математичної моделі системи БПЛА. Розроблені моделі базуються або на розумінні фізики загальної системи, або на надійних реальних даних, отриманих з БПЛА. Метою є відтворення поведінки реального БПЛА в симуляторі таким чином, щоб набір вхідно-вихідних зв’язків був ідентичним як для літального апарата, так і для його симульованої моделі, тобто, якщо певний набір вхідних сигналів для дрона викликає певний набір вихідних сигналів, то цей же набір вхідних сигналів викличе ідентичний набір вихідних сигналів в симуляторі.

Графічна модель: Графічні моделі безпілотних літальних апаратів (БПЛА) використовують сучасні технології графічного комп’ютерного моделювання. Вони надають широкий спектр кольорів, динамічні тіні, фотореалістичний рендеринг та текстури високої якості. Дисплеї забезпечують плавну оновлення швидкістю оновлення, більше 30 кадрів на секунду, а для відображення використовуються 3D графічні двигуни. Враховується орієнтація екрану, незалежно від того, чи є вона горизонтальною, чи вертикальною. Сучасні графічні бібліотеки дозволяють користувачам створювати вражаюче графічне програмне забезпечення, використовуючи передові можливості апаратного забезпечення, як показано на рисунку 6.

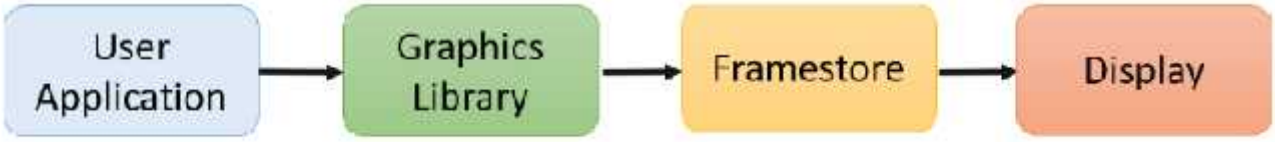

Рис. 6 Графічний конвеєр

Система керування: Літакові симулятори розглядають системи керування як математичні передавальні функції, тому розробник симулятора повинен включити ці передавальні функції у свої алгоритми, щоб забезпечити правильне моделювання відношення вхідного та вихідного сигналу системи. Інженери розглядають перехідну характеристику, амплітудно-частотну характеристику і передавальну функцію як еквівалентні та взаємозамінні [6].

Ідентифікація маршруту польоту: Система навігації у повітряному судні дозволяє користувачам визначати поточне та майбутнє положення. Різні інші підсистеми співпрацюють з системою навігації для забезпечення функцій, таких як ідентифікація маршруту польоту, уникнення зіткнень, планування шляху та визачення повітряного коридору. Однак у літаковому симуляторі відсутні сигнали або датчики, і дизайнерам симулятора доводиться стикатися зі складними викликами. Розробник створює симулятор кожної підсистеми, що відтворює поведінку реальної підсистеми повітряного судна. Розробник переконується, що всі режими роботи повністю відтворені і що немає різниці між симульованим та фактичним компонентом.

Типи та моделі БПЛА: Хороший симулятор також повинен підтримувати різні типи та моделі дронів. Хоча не існує певного стандарту для класифікації БПЛА, деякі експерти категоризують їх за параметрами, такими як розмір, дальність та конфігурація. Їх застосування може варіюватися залежно від розміру, наприклад, більші БПЛА знаходять своє застосування у військових та бойових операціях, тоді як менші БПЛА використовуються для спостереження, академічних досліджень та розваг. Аналогічно, БПЛА з більшою тривалістю польоту застосовуються у місіях спостереження, тоді як менші БПЛА працюють добре для моніторингу транспорту та розваг. За конфігурацією, дрони можна класифікувати як дрони з фіксованим крилом, дрони з обертовими гвинтами та дрони з маховиками. Кожен з них має свій власний аеродинамічний дизайн та функціонал. Дрони з фіксованим крилом зазвичай є більшими і можуть літати на значну відстань. Дрони з обертовими гвинтами є найпоширенішими типами

дронів. Дрони з маховиками натхнені біологічними об’єктами, такими як птахи та комахи, і наразі переважно перебувають у стадії досліджень та розробок.

Специфічні вимоги до застосування: Для симуляторів БПЛА можуть існувати додаткові специфічні вимоги для конкретних застосувань. Хороший симулятор БПЛА може потребувати розширення аспекту комунікації дронів, де кожен БПЛА розглядається як вузол в мережі, а комунікація встановлюється за допомогою специфічних мережевих протоколів. Аналогічно, симулятор, який використовується для дослідження кібербезпеки, може потребувати додаткових модулів для можливих атак. Симуляторам може знадобитися модуль для детального аналізу даних або результатів. Швидкість симулятора повинна бути налаштовуваною, щоб користувачі могли отримати основні деталі. Крім того, симуляції повинні бути виконуваними в різних сценаріях з різними моделями дронів. Крім того, зроблено кілька припущень для спрощення моделювання, таких як жорсткість літака, наземна поверхня як інерціальна система відліку і постійна маса літака протягом всієї симуляції.

Процес розробки симулятора дрона є міждисциплінарним. Він передбачає спільні зусилля з різних галузей, тому він є всеохоплюючою темою, яка потребує комплексного пояснення.

2.2 Архітектура комп’ютерного симулятора БПЛА

Стандартний симулятор є локальною мережею, що складається з декількох комп’ютерів, кожен з яких виконує якусь одну функцію: графічний двигун для імітації зовнішнього середовища; фізичний двигун для моделювання руху та фізичних взаємодій з навколишнім середовищем; інтерфейс інструктора для керування та налаштування симулятора; модулі для підключення зовнішніх пристроїв: джойстиків, кермів керування, рухомих платформ тощо; модулі для обробки та зберігання даних про польоти та результати навчання пілотів.

Кожен комп’ютер у мережі має свої власні завдання та обмеження, але вони всі пов’язані між собою та обмінюються даними для досягнення спільної мети.

Архітектура симулятора БПЛА повинна бути гнучкою та можливою до розширення, оскільки нові пристрої та функціонал можуть бути додані в майбутньому. Також, система повинна бути зручною для використання, має мати інтуїтивний та зрозумілий інтерфейс для керування та налаштування симулятора.

Окремою частиною архітектури є фізична модель БПЛА та його поведінка в різних умовах. Вона повинна бути максимально реалістичною та відображати усі особливості руху та поведінки БПЛА, включаючи фактори, які впливають на політ, такі як атмосферні умови, сила вітру, технічні несправності тощо.

Також, важливим аспектом архітектури є можливість взаємодії з зовнішніми системами та програмними додатками. Наприклад, симулятор може бути підключений до інших симуляторів, що імітують поведінку інших винищувачів чи наземних об’єктів, щоб навчати пілотів реагувати на різноманітні ситуації.

Враховуючи всі ці фактори, архітектура комп’ютерного симулятора БПЛА повинна бути ретельно спроектована та розроблена для забезпечення максимальної ефективності та якості навчання операторів БПЛА.

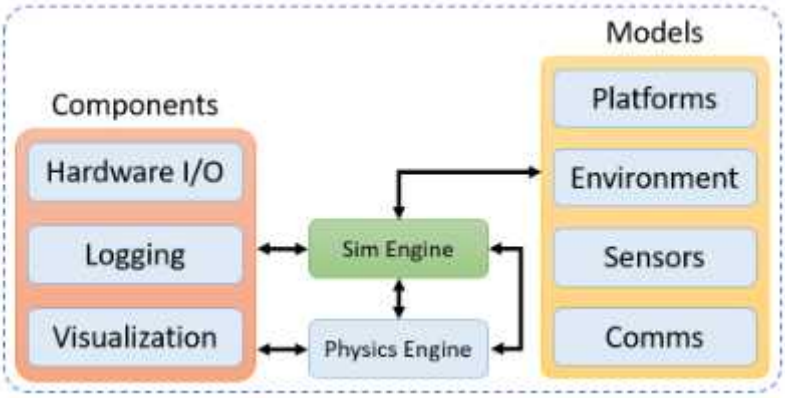

Рис. 7. Архітектура Simbeeotic

Simbeeotic архітектура зображена на рис. 7, вона використовує кілька мережевих комп’ютерів, кожен з яких виконує певну функцію, таку як візуалізація графіки, симуляція фізики, інтерфейс інструктора та інтеграція зовнішніх пристроїв. Цей розподілений підхід забезпечує ефективну та масштабовану продуктивність моделювання. Архітектура Simbeeotic також

наголошує на обробці даних у реальному часі, зборі даних про польоти, а також наданні зворотного зв’язку та аналізу для покращення досвіду навчання [7].

Іншим підходом до симуляторів БПЛА, є AirSim - це кросплатформенний симулятор із відкритим вихідним кодом, розроблений Microsoft для автономних транспортних засобів, зокрема дронів.

Однією з помітних особливостей AirSim є його інтеграція з фреймворками машинного навчання, такими як TensorFlow і PyTorch, що дозволяє дослідникам і розробникам навчати моделі ШІ, використовуючи змодельовані дані. Він також пропонує API та інтерфейси, які забезпечують легку взаємодію з симулятором, що робить його придатним для розробки та тестування алгоритмів керування, систем комп’ютерного зору та інших автономних можливостей. AirSim підтримує цілий ряд транспортних засобів, у тому числі мультиротори та літальні апарати з нерухомим крилом, і надає змодельовані дані датчиків, таких як камери RGB, камери глибини та датчики лідара, щоб імітувати сприйняття реального світу. Це дозволяє розробникам створювати та перевіряти алгоритми для таких завдань, як виявлення об’єктів, планування шляху та уникнення перешкод.

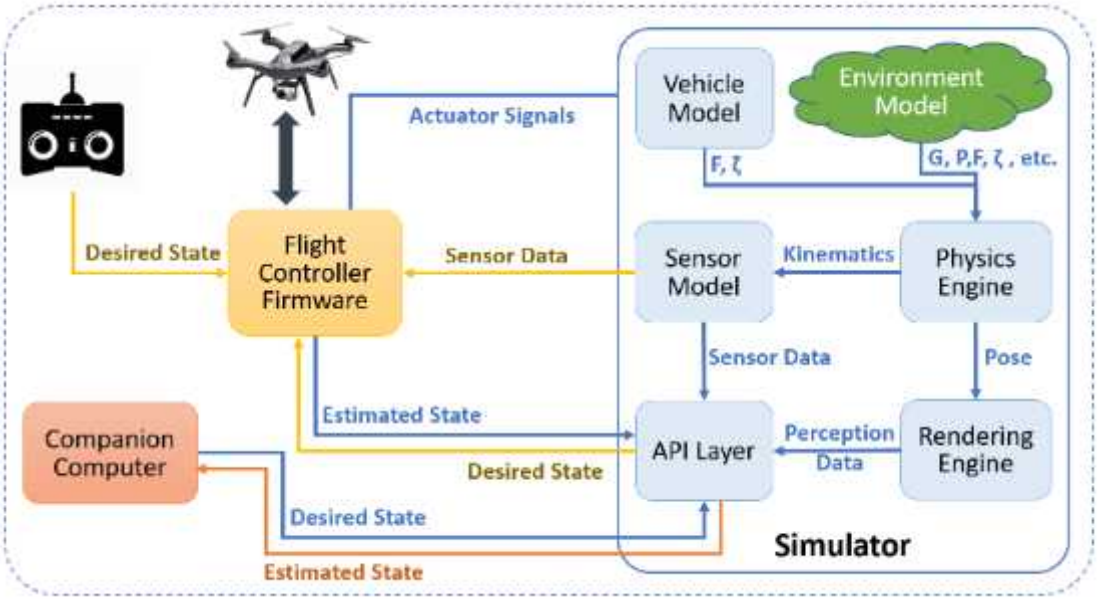

Рис. 8. Архітектура AirSim

Основними компонентами AirSim є модель середовища, модель транспортного засобу, фізичний двигун, моделі сенсорів, рівень API, інтерфейс рендерингу та інтерфейсний рівень, як показано на рис. 8. Загалом, безпілотники містять прошивку контролера польоту, таку як PX4, ROSFlight та Hackflight. Контролер польоту приймає бажаний стан та дані сенсорів як вхід, визначає загальну оцінку поточного стану та генерує сигнали керування приводами для досягнення бажаного стану. Наприклад, у квадрокоптерів бажаними станами є кут крену, тангажу та рискання. Контролер польоту оцінює поточні кути за допомогою даних з акселерометра та гіроскопа і обчислює сигнали мотора для досягнення бажаних кутів. Симуляція надає дані сенсорів контролеру польоту, який подає сигнали приводів до моделі транспортного засобу симулятора. Модель транспортного засобу призначена для обчислення сил та моментів, що виникають від симульованих приводів. Наприклад, у випадку квадрокоптерів тяга та моменти генеруються пропелерами за умови, що напруга на моторах обчислена. Аналогічно, можуть бути відтворені інші сили, такі як тертя, опір повітря та гравітація [8].

2.3 Вибір технологій та засобів розробки

Для розробки комп’ютерного тренажера я обрав мову програмування Python та бібліотеку Pygame. Python є високорівневою мовою програмування, яка володіє простим і зрозумілим синтаксисом. Це дозволяє швидко та зручно реалізовувати різноманітну функціональність у комп’ютерному тренажері. Python має велику спільноту розробників, що сприяє доступності різної документації, бібліотек та підтримки спільнотою.

Pygame, з свого боку, є бібліотекою для розробки ігор та графічних додатків у Python. Вона надає широкий спектр функцій для створення графічного інтерфейсу, обробки введення користувача та роботи з графікою. За допомогою Pygame ви зможете створити вікна, робити малювання на екрані, обробляти події миші та клавіатури, анімувати об’єкти та взаємодіяти з ними.

У контексті комп’ютерного тренажера, Pygame можна використовувати для створення графічного інтерфейсу, що відображатиме візуалізацію польоту БПЛА, зображення сценаріїв та інформаційні панелі. Також можливо обробляти введення команд користувачем, яке включатиме управління БПЛА та взаємодію з компонентами тренажера. Крім того, Pygame дозволяє реалізувати фізику польоту БПЛА, обробку колізій та взаємодію з оточуючим середовищем [9].

Вибір Python та Pygame для розробки комп’ютерного тренажера забезпечить гнучкість та простоту у реалізації різноманітної функціональності, а також доступ до багатого набору інструментів та ресурсів, що сприятимуть успішній розробці тренажера [10].

Також, Python має багато корисних модулів, які допоможуть нам у розробці тренажера, таких як:

1. argparse: Модуль argparse дозволяє зручно обробляти аргументи командного рядка. Він надає можливість визначати та керувати параметрами, що передаються програмі з командного рядка. Завдяки argparse, ви можете легко встановлювати опції, обов’язкові аргументи, створювати документацію та багато іншого.
2. datetime: Модуль datetime надає функціональність для роботи з датами та часом. Він дозволяє створювати, змінювати та форматувати дати та час, а також виконувати різноманітні операції з ними, такі як обчислення різниці між датами, перетворення до різних форматів і т.д.
3. io: Модуль io надає базові класи та функції для роботи зі введенням-виведенням. Він дозволяє зчитувати та записувати дані з різних джерел та призначення, таких як файли, рядки, буфери пам’яті і т.д.
4. json: Модуль json надає інструменти для роботи з форматом JSON (JavaScript Object Notation). Він дозволяє кодувати дані у формат JSON та декодувати дані з формату JSON у Python-об’єкти. Це корисний модуль для обміну даними між різними системами та зберігання конфігураційних файлів.

5. logging: Модуль logging дозволяє здійснювати логування подій та повідомлень в процесі виконання програми. Він надає можливість контролювати рівень логування, записувати повідомлення до різних джерел, таких як консоль або файл, та визначати формат виведення логів.
6. math: Модуль math містить математичні функції та константи. Він надає широкий спектр математичних операцій, таких як обчислення тригонометричних функцій, логарифмів, степеней, констант, арифметичних операцій і багато іншого.
7. os: Модуль os надає функції для взаємодії з операційною системою. Він дозволяє отримувати доступ до системних ресурсів, таких як файли, каталоги, змінні середовища, виконувати команди оболонки і багато іншого.
8. sys: Модуль sys надає функції та змінні, пов’язані з інтерпретатором Python. Він дозволяє отримувати доступ до аргументів командного рядка, взаємодіяти з потоками введення-виведення, керувати виконанням програми та іншими системними ресурсами.
9. time: Модуль time надає функції для роботи з часом. Він дозволяє отримувати поточний час, вимірювати час виконання операцій, затримувати виконання програми та багато іншого.
10. zipfile: Модуль zipfile надає функціональність для роботи з архівами ZIP. Він дозволяє створювати, зчитувати та модифікувати ZIP-архіви, витягувати файли з архіву, додавати файли до архіву та інше.

Кожен з цих модулів має свою унікальну функціональність та може бути використаний для забезпечення потрібного функціоналу та взаємодії з різними аспектами симулятору.

РОЗДІЛ 3

РОЗРОБКА КОМП'ЮТЕРНОГО ТРЕНАЖЕРА

3.1 Розробка моделі БПЛА комп'ютерного тренажера

Почнемо розробку моделі БПЛА (безпілотного літального апарата) комп'ютерного тренажера з класу Airplane. Клас Airplane можна розглядати як основу для моделі літака в тренажері. Він буде містити важливі атрибути та методи, які дозволять нам відтворити поведінку та основні властивості.

```
class Airplane(pygame.sprite.Sprite):
    """ Клас для спрайта БПЛА.
        Усі одиниці вимірювання зберігаються у системі SI """
    NEXT_ID = 0
    MAX_SPEED = 500
    TERMINAL_VELOCITY = MAX_SPEED / 5 # Why not?
    LABELS = "ID:\tX:\tY:\tALT:\tSPD:\tACCEL:\tVSPD:\t\
HDG:\tROLL:\tPITCH:\tPTS:\tDMG:\t"
    def __init__(self, x=(0, 0, 0, 0, 0), z=None, width=None,
                 height=None, altitude=None, player_id=None):
        """Ініціалізування екземпляру класа."""
        super(Airplane, self).__init__()
        if z is None:
            x, z, width, height, altitude = x
        elif width is None:
            altitude = z
            x, z, width, height = x
        elif height is None:
            altitude = width
            width, height = z
            x, z = x
        if player_id is None: # Отримати ID літака
            self._id = Airplane.NEXT_ID
            Airplane.NEXT_ID += 1
        else: self._id = player_id
        # Ініціалізувати приватні змінні
        self._pos = [x, z]
        self._size = [width, height]
        self._altitude = altitude
        self._heading = 0
        self._pitch = 0
        self._speed = 0
```

```
        self._acceleration = 0
        self._gravity = 0
        self._throttle = 0
        self._roll_level = 0
        self._vertical_roll_level = 0
        self._autopilot_info = {
            'enabled': False,
            'conditions': {
                'roll-centered': True,
                'vertical-roll-centered': True,
                'throttle-centered': True
            }
        }
        self._within_objective_range = False
        self._points = 0
        self._exit_code = 0
        self._health = 100
        self._time = time.time()
```

Даний код визначає клас Airplane, який успадковує клас pygame.sprite.Sprite (базовий клас) і представляє спрайт літака в тренажері. Він надає початковий каркас для моделі літака в тренажері.

Основні атрибути класу Airplane включають:

- NEXT_ID: змінна класу, яка визначає наступний доступний ідентифікатор для літака.
- MAX_SPEED: максимальна швидкість літака.
- TERMINAL_VELOCITY: термінальна швидкість, яка обчислюється як п’ята частина максимальної швидкості.

Крім того, клас містить ряд приватних змінних, таких як позиція, розмір, висота, кут, швидкість, прискорення, гравітація тощо, які використовуються для відстеження стану літака. Також в класі присутня змінна LABELS, яка містить мітки для виведення важливої статистики про літак.

У конструкторі __init__ класу Airplane ініціалізуються всі атрибути літака на основі переданих параметрів, або призначаються значення за замовчуванням. Клас також має різні методи, але їх реалізація буде описана далі. Загальною метою методів класа Airplane є симуляція поведінки літака, таких як зліт, посадка, рух у заданому напрямку, зміна висоти, поворот і т. д.

```
def __repr__(self, show_labels=True):
    """Виведення важливої статистики про літак."""
    msg = ("%i\t%i\t%i\t%i\t%.1f\t%.1f\t%.1f\t%.1f\t%.1f\t%.1f\t\
            %i\t%.1f\t" % (self.id_, self.x, self.z,
           self.altitude, self.speed, self.acceleration,
           self.vertical_velocity, self.heading, self.roll,
           self.pitch, self.points, 100 - self.health))
    if show_labels:
        return "%s\n%s" % (Airplane.LABELS, msg)
    else:
        return msg
```

Метод __repr__ класу Airplane генерує рядок, який містить важливу статистику про літак. У рядку виводяться значення ідентифікатора (id_), позиції по осях X та Z (x та z), висоти (altitude), швидкості (speed), прискорення (acceleration), вертикальної швидкості (vertical_velocity), напрямку (heading), крену (roll), тангажу (pitch), набраних балів (points) та відсотку життя (health).

Якщо параметр show_labels встановлений на True, метод повертає рядок, який містить мітки для кожного значення, розділені табуляцією, разом з рядком статистики. У протилежному випадку, метод повертає лише рядок статистики без міток. Цей метод дозволяє отримати зручне представлення стану літака, що може бути використано для виведення на екран або збереження в лог-файлі.

Надалі в коді реалізувано для кожного приватного атрибуту класа Airplane властивість getter і setter за допомогою @property і @.setter. Ось приклад коду:

```
# змінні
@property
def id_(self):
    """Отримати ID літака."""
    return self._id
@property
def pos(self):
    """Отримати координату (x, z) в метрах."""
    return self._pos
@pos.setter
def pos(self, new_value):
    """Встановити координату (x, z) в метрах."""
    if not isinstance(new_value, (list, tuple)):
        raise TypeError("Позиція має бути списком або кортежем.")
    if len(new_value) != 2:
```

```
            raise ValueError("Позиція повинна складатися з двох значень.")
        if not isinstance(new_value[0], (int, float)):
            raise ValueError("X повинно бути числом")
        if not isinstance(new_value[1], (int, float)):
            raise ValueError("Z повинно бути числом")
        self._pos = new_value
```

Цей код визначає три властивості (properties) для доступу до змінних id_ та pos об’єкта класу Airplane. Властивість id_:

- @property означає, що це властивість доступу для змінної id_.
- Метод id_ без параметрів є геттером для отримання значення змінної id_.
- При використанні airplane.id_ (де airplane - об’єкт класу Airplane), ця властивість поверне значення _id.

Властивість pos:

- @property означає, що це властивість доступу для змінної pos.
- Метод pos без параметрів є геттером для отримання значення змінної pos.
- При використанні airplane.pos (де airplane - об’єкт класу Airplane), ця властивість поверне значення _pos.

Властивість pos (setter):

- @pos.setter означає, що це властивість доступу для встановлення значення змінної pos.
- Метод pos з параметром new_value є сеттером для встановлення значення змінної pos.
- Перед встановленням нового значення, перевіряється, чи new_value є списком або кортежем, чи має він довжину 2 та чи перші два елементи є числами.
- Якщо всі перевірки успішні, то значення _pos оновлюється на new_value.

Отже, використовуючи ці властивості, можна отримувати та встановлювати значення змінних id_ та pos об’єкта Airplane, забезпечуючи при цьому перевірку вхідних даних.

За фізичну модель літака відповідає метод update(), оновлює його стан і параметри на кожному кроці. В цьому методі виконуються різні обчислення, що відображають фізичні властивості літака, такі як прискорення, швидкість,

положення, кут повороту, висота та інші. Крім того, метод також враховує різні фактори, такі як гравітація, пошкодження від перевищеної швидкості та взаємодію з автопілотом.

```
def update(self):
    """Оновлення бпла"""
    tick_duration = time.time() - self._time
    self._time = time.time()

    # пошкодження
    damage = 0
    # розрахунок урону від пошкоджень
    self.health -= damage
```

В даній частині коду визначається тривалість одного кроку симуляції шляхом розрахунку різниці в часі між поточним і попереднім кроком, а також ініціалізується змінна damage зі значенням 0, яка представляє сумарне пошкодження, яке буде застосоване до літака.

Потім, якщо швидкість літака перевищує 75% від максимальної швидкості (self.MAX_SPEED * 0.75), обчислюється пошкодження від перевищення швидкості.

Далі відбувається обробка фізичних ефектів заносу та впливу гравітації на бпла.

```
    # обробка фізичних ефектів заносу та впливу гравітації
    if self.speed <= (self.MAX_SPEED / 5):
        max_vert_roll = max((self.speed-(self.MAX_SPEED / 10))
                            / (self.MAX_SPEED / 40), 0)
    else: max_vert_roll = 4
    self.gravity += (((self.MAX_SPEED / 10 - self.speed)
                      / self.MAX_SPEED * self.TERMINAL_VELOCITY)
                     - (self.gravity ** 2
                        / (self.TERMINAL_VELOCITY ** 2 / 10)))
    if self.gravity < 0:
        self.gravity = 0
    if self.altitude <= 0.1:
        self.gravity = 0
```

Умова if self.speed <= (self.MAX_SPEED / 5) перевіряє, чи швидкість літака менша або рівна одній п’ятій від максимальної швидкості (MAX_SPEED). Це використовується для визначення максимального рівня "заносу" (slip) літака, коли

він рухається з низькою швидкістю. Занос - це стан, коли літак втрачає контроль і починає ковзати або обертатися на боку.

Змінна max_vert_roll визначає максимальний рівень "заносу" в залежності від швидкості літака. Змінна gravity відповідає за гравітаційний вплив на літак. Вона оновлюється на основі розрахунків, що враховують швидкість літака. Формула для оновлення gravity включає в себе наступні елементи:

- (((MAX_SPEED / 10 - speed) / MAX_SPEED * TERMINAL_VELOCITY) – ця формула відповідає за гравітаційну силу, що залежить від різниці між швидкістю літака та MAX_SPEED / 10. Чим ближче швидкість до MAX_SPEED / 10, тим більше гравітаційна сила.
- (gravity ** 2 / (TERMINAL_VELOCITY ** 2 / 10)) - ця формула відповідає за аеродинамічну силу опору повітря, яка залежить від квадрата значення gravity. Він зменшується пропорційно до квадрату gravity. Це допомагає обмежити швидкість зниження літака при високих значеннях gravity.

Умови if self.gravity < 0 та if self.altitude <= 0.1 встановлюють значення gravity на 0, якщо воно стає від’ємним або висота літака менша або рівна 0.1. Це означає, що літак знаходиться на землі або поза межами повітряного простору, і гравітаційний вплив не застосовується.

У цьому фрагменті коду відбувається оновлення напрямку та крену (roll) літака.

```
# напрямок та крен бпла
self.heading += (self.roll * tick_duration)
if self.vertical_roll_level > max_vert_roll:
    self.vertical_roll_level = max_vert_roll
self.pitch_degrees = self.vertical_roll_level * 10
```

self.heading відповідає за напрямок літака. Він оновлюється шляхом додавання self.roll * tick_duration до поточного значення self.heading. self.roll визначає крен літака, тобто нахил вліво або вправо. Чим більше значення self.roll, тим більше зміна в напрямку літака.

Умова if self.vertical_roll_level > max_vert_roll перевіряє, чи значення self.vertical_roll_level (рівень крену по вертикалі) перевищує max_vert_roll,

обмеження, яке ми обрахували раніше. Якщо це так, значення self.vertical_roll_level встановлюється на max_vert_roll, щоб обмежити крен літака.

self.pitch_degrees визначає кут нахилу літака (pitch). Він обчислюється, помноживши self.vertical_roll_level на 10. Значення self.vertical_roll_level відображається у градусах, а множення на 10 дає нам відповідний кут нахилу літака.

У цьому фрагменті коду відбувається обчислення та оновлення прискорення та швидкості літака.

```
# прискорення
    self.acceleration = ((self.throttle ** 2 / 250)
                         - (self.speed ** 2 * 40 / self.MAX_SPEED ** 2)
                         * (1 - self.speed / self.MAX_SPEED))
    self.speed += (self.acceleration * tick_duration)
```

) self.throttle відповідає за потужність двигуна літака. Вираз self.throttle**2 / 250 обчислює прискорення на основі квадрата значення self.throttle. Це означає, що більша потужність двигуна призводить до більшого прискорення.

) self.speed відображає поточну швидкість літака. За допомогою виразу self.acceleration * tick_duration, прискорення помножується на часовий крок tick_duration та додається до поточної швидкості self.speed. Це дає нам нове значення швидкості літака після оновлення.

Оновлення позиції літака відповідно до його швидкості та висоти.

```
    # рух бпла
    hspeed = self.horizontal_speed * tick_duration
    vspeed = self.total_vertical_velocity * tick_duration
    self.x += math.sin(self.heading) * hspeed
    self.z -= math.cos(self.heading) * hspeed
    self.altitude += vspeed
    if self.altitude < 0.1:
        self.altitude = 0
```

hspeed відображає горизонтальну швидкість літака, обчислену як добуток self.horizontal_speed (горизонтальна швидкість) та tick_duration (часовий крок).

vspeed відображає вертикальну швидкість літака, обчислену як добуток self.total_vertical_velocity (загальна вертикальна швидкість) та tick_duration. Він використовується для обчислення зміни висоти літака.

self.x та self.z відповідають координатам літака на площині XZ (горизонтальна площина). За допомогою тригонометричних функцій math.sin та math.cos, обчислюється зміна горизонтальних координат літака на основі кута self.heading (напрямок літака) та hspeed.

self.altitude відображає висоту літака. Вона оновлюється шляхом додавання vspeed до поточної висоти. Умова if self.altitude < 0.1 перевіряє, чи висота літака менша за 0.1. Якщо це так, висота встановлюється на значення 0.

Обробка пошкоджень, які виникають внаслідок перевищення швидкості літака та позиції дроселя (throttle).

```
# пошкодження від перевищення швидкості
if self.speed > self.MAX_SPEED * 0.75:
    damage += ((self.speed - self.MAX_SPEED*0.75) ** 2
               / (self.MAX_SPEED**2*10) * tick_duration)
if self._throttle > 75:
    damage += (self._throttle - 75) ** 2 / 1000 * tick_duration
```

Якщо швидкість літака (self.speed) перевищує 75% максимальної швидкості (self.MAX_SPEED * 0.75), обчислюється пошкодження. Воно залежить від квадрата різниці між поточною швидкістю літака та 75% максимальної швидкості. Це значення потім ділиться на квадрат максимальної швидкості помножене на 10. Отримане значення помножується на tick_duration для врахування тривалості часового кроку. Загальне пошкодження додається до змінної damage.

Якщо позиція дроселя (self._throttle) перевищує 75, також обчислюється пошкодження. Воно залежить від квадрата різниці між позицією дроселя та значенням 75. Отримане значення ділиться на 1000, а потім помножується на tick_duration. Результат додається до змінної damage.

Також важливою особливістю будь-якого бпла є автопілот, ось частина коду яка це реалізує:

```
    # автопілот
    if self.autopilot_enabled:
        self.roll_level *= (0.5 ** tick_duration)
        self.vertical_roll_level *= (0.5 ** tick_duration)
        self._throttle = 50 + (self.throttle-50) * (
            0.5 ** tick_duration)
```

Якщо автопілот (self.autopilot_enabled) увімкнено, відбувається зменшення рівня крену (self.roll_level) та вертикального крена (self.vertical_roll_level) з часом. Значення кожного рівня помножується на (0.5 ** tick_duration), де tick_duration відображає тривалість часового кроку. Це зменшення рівнів крену та вертикального крена забезпечує поступове згладжування руху літака при використанні автопілоту.

Значення дроселя (self._throttle) також змінюється в режимі автопілоту. Воно обчислюється на основі різниці між поточним значенням дроселя та значенням 50, помноженого на (0.5 ** tick_duration). Таке зменшення адаптує позицію дроселя до середнього значення, забезпечуючи стабільніший режим політів при використанні автопілоту.

Для отримання статусу автопілота літака використовується метод @property autopilot_enabled:

```
@property
def autopilot_enabled(self):
    """Отримати статус автопілота бпла."""
    if not self._autopilot_info['enabled']:
        return False
    else: # Перевірити, чи може бути автопілот вимкненим
        if abs(self.roll_level) < 0.1:
            self.roll_level = 0
            self._autopilot_info['conditions'][
                'roll-centered'] = True
        if abs(self.vertical_roll_level) < 0.1:
            self.vertical_roll_level = 0
            self._autopilot_info['conditions'][
                'vertical-roll-centered'] = True
        if abs(50 - self.throttle) < 1:
            self.throttle = 50
```

```
            self._autopilot_info['conditions'][
                'throttle-centered'] = True
        if all(self._autopilot_info['conditions'].values()):
            self._autopilot_info['enabled'] = False
        return self._autopilot_info['enabled']
```

Якщо self._autopilot_info[’enabled’] дорівнює False, автопілот вимкнений і повертається значення False. У протилежному випадку, проводяться перевірки для визначення, чи може автопілот бути вимкненим.

Якщо всі умови в self._autopilot_info[’conditions’] виконуються (тобто всі значення рівні True), тоді self._autopilot_info[’enabled’] встановлюється на False, означаючи, що автопілот може бути вимкнений. На кінці методу повертається значення self._autopilot_info[’enabled’], що вказує на поточний статус автопілота.

Метод enable_autopilot використовується для увімкнення автопілота літака:

```
def enable_autopilot(self):
    """Enable the autopilot."""
    self._autopilot_info['enabled'] = True
    for condition in self._autopilot_info['conditions']:
        self._autopilot_info['conditions'][condition] = False
```

У тілі функції спочатку встановлюється self._autopilot_info[’enabled’] на True, що означає, що автопілот активований. Потім за допомогою циклу for проходимося по всіх умовах в self._autopilot_info[’conditions’] і встановлюємо кожну умову на False. Це означає, що автопілот не центрується ні по розвороту (’roll-centered’), ні по вертикальному розвороту (’vertical-roll-centered’), ні по швидкості (’throttle-centered’). Таким чином, автопілот запускається в початковому стані без центрування.

За відображення бпла на екрані графічного інтерфейсу відповідає функція draw().

```
def draw(self, client, airspace):
    """Відображення бпла"""
    image = pygame.transform.rotate(
        client.scaled_images['navmarker'], -self.heading_degrees)
    image.set_colorkey((255, 255, 255)) #робимо білий колір прозорим
    image = pygame.transform.scale(image,
(image.get_width()//8,image.get_width()//8 ))
    draw_rect = image.get_rect()
```

```
draw_rect.center = (
    self.x / airspace.width * client.airspace_rect.width
    + client.airspace_rect.left,
    self.z / airspace.height * client.airspace_rect.height
    + client.airspace_rect.top
)
client.screen.blit(image, draw_rect)
```

Зображення літака client.scaled_images[’navmarker’] обертається на протилежний кут відносно градусної величини self.heading_degrees. Це дозволяє візуалізувати орієнтацію літака на екрані. Зображення літака масштабується до нового розміру, за допомогою pygame.transform.scale(). Це забезпечує відповідний розмір зображення літака на екрані. Центр зображення літака позиціонується відповідно до положення літака (self.x, self.z) у повітряному просторі. Зображення літака (image) відображається на екрані у визначеному положенні (draw_rect) за допомогою функції blit(). В якості спрайту для нашого бпла я буду використовувати наступний рисунок бпла:

Рис. 9. Зображення бпла в симуляторі

3.2 Розробка моделі повітряного простору

За управління повітряним простором у симуляторі, відображення, оновлення бпла та об’єктів відповідає клас Airspace. Модель повітряного простору, реалізована у класі Airspace, має багато функціональностей та можливостей, вона пропонує широкий набір функцій, які можуть бути використані для створення складних повітряних сценаріїв. Декілька ключових особливостей цієї моделі включають:

1. Розмір повітряного простору: Повітряний простір представляється у вигляді прямокутника з фіксованою шириною та висотою. Цей розмір може бути налаштований, і він використовується для обмеження руху літаків та розташування об’єктивів.
2. Керування літаками: Модель повітряного простору містить групу літаків, представлених у вигляді об’єктів класу Airplane. Ця модель дозволяє додавати та видаляти літаки з повітряного простору, а також оновлювати їх стан.
3. Генерація об’єктів: Модель також має функціональність для генерації об’єктів, представлених у вигляді екземплярів класу Objective. Генерування об’єктів відбувається з урахуванням обмежень їх розташування та уникнення колізій з літаками.
4. Оновлення та відображення: Методи update() та draw() відповідають за оновлення та відображення повітряного простору та його елементів, таких як літаки та об’єкти. Це дозволяє забезпечити реалістичний та динамічний вигляд симулятора.

Цей код визначає клас Airspace:

```
class Airspace(pygame.rect.Rect):
    """The class for an airspace."""
    AIRSPACE_DIM = 100000
    MIN_OBJ_ALT = 7500
    MAX_ALTITUDE = 15000
    ALTITUDE_TOLERANCE = 1400
    ALTITUDE_WITHIN = 1000
    POINTS_REQUIRED = 5
    def __init__(self, x=(0, 0, 0, 0), y=None, w=None, h=None):
        """Ініціалізація екземпляру класу"""
```

```
        if y is None:
            x, y, w, h = x # Вхід: 1
        elif w is None and h is None:
            (x, y), (w, h) = x, y # Вхід: 2

        super(Airspace, self).__init__(
            x, y, Airspace.AIRSPACE_DIM, Airspace.AIRSPACE_DIM)
        self.planes = AdvancedSpriteGroup()
        self.objectives = AdvancedSpriteGroup()
```

Основні атрибути класу Airspace включають:

- AIRSPACE_DIM: Розмір повітряного простору (ширина і висота), встановлений на 100000 одиниць.
- MIN_OBJ_ALT: Мінімальна висота об’єктів, встановлена на 7500 одиниць.
- MAX_ALTITUDE: Максимальна висота, до якої можуть піднятися літаки, встановлена на 15000 одиниць.
- ALTITUDE_TOLERANCE: Допустиме відхилення висоти між літаком і об’єктивом для визначення колізії, встановлене на 1000 одиниць.
- ALTITUDE_WITHIN: Максимальна висота між об’єктивом і літаком, при якій вони вважаються "в межах" тобто можуть взаємодіяти , встановлено на 2000 одиниць.
- POINTS_REQUIRED: Кількість точок, необхідних для досягнення мети, встановлена на 5.

Конструктор __init__ ініціалізує об’єкт Airspace. Він приймає аргументи x, y, w та h, які використовуються для визначення розташування та розміру повітряного простору. Він створює об’єкти planes і objectives, які є екземплярами класу AdvancedSpriteGroup.

Метод __repr__ в класі Airspace відповідає за представлення об’єкта повітряного простору у вигляді рядка. В цьому випадку, метод повертає рядок, який містить важливу інформацію про повітряний простір.

```
def __repr__(self):
    """Відображення важливої інформації про повітряний простір."""
    return "{}x{} AIRSPACE\nPLANES:{}\nOBJECTIVES:{}".format(
        self.width, self.height,
        ''.join(["\n%s" % repr(plane) for plane in self.planes]),
        ''.join(["\n%s" % repr(obj) for obj in self.objectives]))
```

Розміри повітряного простору: ширина і висота (наприклад, "100x100 AIRSPACE"). Інформація про бпла та об’єкти: перераховані усі літаючі об’єкти у повітряному просторі. Отриманий рядок використовується для представлення інформації про повітряний простір при використанні функції print або при виведенні об’єкта Airspace у консоль.

Метод draw() в класі Airspace відповідає за відображення повітряного простору та всього, що знаходиться в ньому.

```
def draw(self, client):
    """відображення повітряного простіру і всього, що в ньому
    знаходиться."""
    client.screen.blit(
        client.images['navcircle'], client.airspace_rect)
    for plane in self.planes: # Відобразити бпла
        plane.draw(client, self)
    for obj in self.objectives: # Відобразити об’єкти
        obj.draw(client, self)
```

У цьому методі використовуються графічні можливості бібліотеки pygame для відображення елементів повітряного простору на екрані. Використовуючи функцію blit(), зображення "navcircle" розміщується на екрані за допомогою client.screen.blit. Це візуальний елемент, який представляє повітряний простір («зображено на рис. 10).

Рис. 10. Зображення повітряного простору в симуляторі

Для кожного БПЛА і об’єкту у списку planes і objectives викликається метод draw об’єкта літака (plane.draw(client, self)) та об’єкту (obj.draw(client, self)). Це дозволяє відобразити БПЛА та об’єкти на екрані повітряного простору.

Метод update в класі Airspace відповідає за оновлення повітряного простору.

```
def update(self):
    """Оновлення повітряного простіру."""
    self.planes.update()

    for plane in self.planes: # Перевірка колізії об'єктів з бпла
        collisions = pygame.sprite.spritecollide(
            plane, self.objectives, True, self.collided)
        for collision in collisions:
            plane.points += 1
            self.generate_objective()
```

Працює наступним чином:

Викликається метод update для кожного бпла в списку planes, що оновлює стан кожного. Для кожного бпла в списку planes перевіряється, чи відбувається колізія з об’єктами. Після перевірки колізій генерується новий об’єкт за допомогою

методу generate_objective(). Це забезпечує постійне створення нових точок після їх проходженням бпла.

Метод add_plane в класі Airspace відповідає за додавання літака до повітряного простору.

```
def add_plane(self, plane=None, player_id=None):
 """Додає літак до повітряного простору.
    Створений літак знаходиться в центрі повітряного простору."""
    if plane is None:
        plane = Airplane(
            self.width/2, self.height/2,
            self.width*0.07, self.height*0.07, 0,
            player_id=player_id)
    if isinstance(plane, Airplane):
        self.planes.add(plane)
        return plane
    raise TypeError("plane must be an Airplane or None.")
```

Метод add_plane надає можливість додавати літаки до повітряного простору. Якщо літак не передано, створюється новий літак за замовчуванням і розміщується у центрі повітряного простору. Після додавання літака до списку planes, метод повертає цей новододаний літак.

Також в класі Airspace присутні інші методи, коротко розповім про кожного з них:

1. generate_objective() створює новий об’єкт Objective у межах повітряного простору.

2. collided() є статичним (метод класу), що використовується для перевірки, чи сталася колізія між літаком (airplane) і об’єктом (objective) в повітряному просторі.

3. in_bounds() метод перевіряє, чи знаходиться об’єкт всередині меж повітряного простору. Він потрібний для правильної генерації об’єктів.

3.3 Розробка інтерфейсу користувача

Розділ "Розробка інтерфейсу користувача" відповідає за створення елементів інтерфейсу, які дозволяють користувачу взаємодіяти з програмою(тренажером). Цей розділ фокусується на створенні зручного та ефективного інтерфейсу, який дозволяє користувачеві виконувати налаштовування параметрів та отримувати необхідну інформацію. Через громіздкість коду, не будуть наведені повні приклади. Замість цього, будуть короткі уривки коду або фрагменти, які ілюструють концепцію або основну ідею, щоб дати загальне уявлення про те, як використовувати певний елемент коду. Щоб побачити повний код, можна переглянути вихідний код проєкту, доступний в відповідних ресурсах.

Клас Client відповідає за реалізацію інтерфейсу користувача.

```
class Client(pygame.rect.Rect):
#...
def __init__(self, window_size=DEFAULT_SIZE, player_id=None):
# ініціалізація класу
#...
```

Він забезпечує наступні можливості:

1. Налаштування тренажеру: клас містить властивості, які дозволяють користувачеві налаштувати параметри гри, такі як музика, звук, одиниці виміру, максимальна кадрова частота тощо.
2. Обробка командного рядка: клас має методи для обробки командного рядка, які дозволяють користувачу передавати певні параметри або виконувати певні дії за допомогою команд.
3. Відображення графічного інтерфейсу: клас включає методи для відображення графічного інтерфейсу, який включає елементи керування, вікна, відображення стану тренажеру тощо.
4. Управління: клас надає можливості для керування ходом симуляції, зміни етапів, обробки введення команд користувача, перехід до наступного етапу тощо.

Цей розділ є важливим у розробці програми, оскільки він забезпечує зручну та інтуїтивно зрозумілу взаємодію користувача з програмою.

Правильно розроблений інтерфейс користувача може поліпшити враження користувача від програми та забезпечити зручну роботу з нею.

Основні атрибути(змінні) які були створенні для цього класу.

- Клас Client успадковується від pygame.rect.Rect, що означає, що він має всі властивості та методи базового класу Rect з бібліотеки Pygame.
- Константи GAME_STAGES та GAME_LOOPS є словниками, які використовуються для визначення методів та циклів тренажеру для кожного етапу.
- Змінна PATH містить шлях до поточного каталогу, а LOG_PATH - шлях до папки з журналами.
- DEFAULT_SIZE містить розмір вікна гри за замовчуванням.
- NEXT_ID використовується для надання ідентифікатора.
- EXIT_TITLES та EXIT_REASONS містять опис та причини виходу з тренажеру.
- DEFAULT_OPTIONS містить налаштування за замовчуванням, такі як звук, одиниці вимірювання тощо.
- CONTROLS містить список усіх можливих керувань.
- DEFAULT_CONTROLS містить значення за замовчуванням для керування.
- CONTROL_NAMES містить назви керувань.
- FPS_OPTIONS містить доступні варіанти кадрової частоти.
- UNITS містить набір одиниць для вимірювання швидкості та положення.
- Метод __init__ виконує початкову ініціалізацію класу Client, включаючи обробку командного рядка та завантаження налаштувань з файлу.
- Властивість id_ повертає ідентифікатор користувача.
- Властивість stage використовується для отримання та встановлення поточного етапу симуляції.
- Властивість exit_code повертає код виходу з гри на основі різних умов, таких як здоров’я літака, набраних балів, висоти, позиції тощо.

Це лише атрибути та методи доступів до атрибутів класу Client, решта коду буде містити методи для логування, завантаження ресурсів, відображення інтерфесу і тд.

Ось приклад коду.

Метод mainloop(), є основним циклом тренажеру:

```
def mainloop(self, airspace):
#...
```

Налаштування Pygame: Ініціалізація Pygame, створення вікна тренажера та завантаження ресурсів. Налаштування простору повітря: Створення простору повітря, включаючи налаштування його розмірів та розташування на екрані.

```
#... повітряний простір
self.airspace = airspace
self.airspace_rect = pygame.rect.Rect(
    self.size[0]*7/16, self.size[1]/24,
    self.size[0]*35/64, self.size[1]*35/48)
```

Створення літака та об’єктивів: Створення гравця-бпла та генерація об’єктів у просторі повітря. Обробка вхідних подій: Зчитування подій в меню тренажера, таких як натискання клавіш, зміна розміру вікна тощо.

```
# створення бпла та об’єктів
self.plane = self.airspace.add_plane(player_id=self.id_)
self.airspace.generate_objective()
```

Логіка тренажеру: Виконання логіки гри, такої як перевірка стану бпла, визначення найближчого об’єкта, перевірка умов для завершення завдання.

```
for obj in self.airspace.objectives: # визначення найближчого об'єкта
    self.closest_objective = obj
```

Оновлення відображення: Оновлення екрану гри згідно зі змінами, які відбулися під час обробки подій та логіки. Цей етап включає виклик методів для відображення різних елементів на екрані. Наприклад:

startup_screen(): екран запуску

instructions_screen(): екран інструкцій

settings_screen(): екран налаштувань

main_screen(): основний екран

end_screen(): екран завершення

Завершення роботи програми: Перевірка умов завершення гри, таких як вихід з програми або досягнення кінцевого стану гри. Якщо гра закінчена, виконується необхідна обробка і завершення програми.

Метод draw(), відповідає за відображення графічних елементів на екрані.

```
def draw(self):
#...
```

Основний код методу виконує наступні дії:

1. Визначає найближчий об’єкт до літака на основі обчислення відстані між літаком і об’єктами в просторі. Знаходиться об’єкт з найменшою відстанню та зберігається у змінній closest_objective.
2. Обчислюється положення та перетворення для графічного елементу attitude_tape та його оверлею attitude_tape_overlay. Використовуються математичні операції для обчислення положення та обертання.
3. Рисуються панелі, прямокутники та зображення на екрані за допомогою методів pygame.draw.rect та pygame.Surface.blit.
4. Відновлюється фоновий колір на певних областях екрану за допомогою pygame.draw.rect.
5. Викликається метод draw об’єкта airspace для відображення повітряного простору.
6. Виводяться тексти, які відображають інформацію про локацію літака, об’єктив та різні параметри. Це виконується за допомогою методу draw_text, який вже описаний раніше.
7. Малюються різні графічні елементи, такі як панель дроселя, рівень швидкості, стан автопілота та попередження.
8. Якщо графічний інтерфейс паузи активовано, малюються кнопки та відображається текст "Settings".

Метод draw_text(), відповідає за відображення тексту на екрані гри.

```
def draw_text(self, text, x, y=None, mode="center",
              color_id=(0, 0, 0), font_id='default', antialias=1,
              bg_color=None):
```

```
#...
```

Основні параметри методу:

- text: Текст, який потрібно відобразити.
- x: Позиція по осі X, де текст буде відображений.
- y: Позиція по осі Y, де текст буде відображений.
- mode: Режим позиціонування тексту. За замовчуванням використовується режим "center", що означає, що позиція тексту визначається його центром.
- color_id: Ідентифікатор кольору тексту. Може бути заданий як кортеж (R, G, B) або (R, G, B, A), або як ідентифікатор кольору зі словника colors.
- font_id: Ідентифікатор шрифту, що використовується для відображення тексту.
- antialias: Параметр згладжування тексту. За замовчуванням встановлений на 1, що означає використання згладжування.
- bg_color: Колір фону тексту. Якщо заданий, то навколо тексту буде відображений прямокутник з цим кольором фону.

Метод control_plane відповідає за управління літаком на основі введення користувача.

```
def control_plane(self):
#...
```

Загалом, цей метод дозволяє оператору керувати нахилом літака, вертикальним нахилом та швидкістю за допомогою клавіатурного введення. Він також обробляє певні ключові події для спеціальних дій, таких як встановлення швидкості на певні відсотки або увімкнення автопілоту.

Пояснення кожної частини цього методу:

1. pygame.key.get_pressed(): Ця функція отримує стан усіх клавіш на клавіатурі. Вона повертає список булевих значень, які показують, чи натиснута кожна клавіша в даний момент.
2. Цикл for проходиться по списку keys, щоб перевірити стан кожної клавіші і оновити словник keys_held відповідно. Словник keys_held відстежує, як довго утримується кожна клавіша.

3. Рівень нахилу літака оновлюється на основі клавіш ліворуч і праворуч. Значення roll_level зменшується, коли утримується клавіша ліворуч, і збільшується, коли утримується клавіша праворуч
4. Рівень вертикального нахилу літака оновлюється на основі клавіш вгору і вниз. Значення vertical_roll_level зменшується, коли утримується клавіша вгору, і збільшується, коли утримується клавіша вниз.
5. Швидкість літака оновлюється на основі клавіш управління швидкістю. Значення throttle зменшується, коли утримується клавіша зменшення швидкості, і збільшується, коли утримується клавіша збільшення швидкості.
6. Метод також обробляє конкретні події натискання клавіш. Якщо натиснута клавіша відповідає одній з попередньо визначених клавіш управління, виконуються певні дії. Наприклад, якщо натиснута клавіша зменшення швидкості, швидкість літака встановлюється на 0%. Якщо натиснута клавіша автопілота, увімкнюється автопілот літака.

Також в симуляторі реалізована функція звукового супровіду, але вона не отримала реалізації у зв’язку з відсутністю звукових доріжок, для тих чи інших аспектів симуляції, наприклад звуки двигунів, повітряного потоку або інших звуків, що створюють іммерсивну атмосферу та допомагають оператору краще сприймати ситуацію під час польоту.

```
def play_sounds(self):
    """відтворити звукові ефекти та сигнали."""
...
```

Варто відзначити, що функціональні частини гри можуть взаємодіяти між собою та викликати один одного для досягнення бажаного результату. Наприклад, метод control_plane викликається для обробки керування літаком, але цей метод може викликати інші методи для зміни стану літака або виконання інших операцій, пов’язаних з рухом літака.

Цей розділ не включає опис методів які використовуються для логування та обробки помилок при польоті, вони будуть описані в наступному розділі про тестування.

3.4 Тестування комп’ютерного тренажера

Методи які допомагають збирати інформацію під час польоту бпла та записувати їх у журнал для подальшого аналізу та налагодження представленні нище:

Метод log(self): Цей метод використовується для записування повідомлень у журнал програми.

Метод get_tick_values(self): Цей метод викликається кожен раз перед записом значень у журнал. Він оновлює значення tick (часовий маркер), previous_time (попередній час) та time (поточний час) за допомогою модуля time. Цей метод викликається в кожній ітерації циклу тренажеру, щоб оновити ці значення для подальшого запису у журнал (Рис. 11). Журнал має наступний вигляд:

```
DEBUG:root:TICK:    ID: X:  Y:  ALT:    SPD:    ACCEL:  VSPD:
DEBUG:root:0    0   50000   50000   0   0.0 0.0 0.0
DEBUG:root:231  0   50000   50000   0   0.0 0.0 0.0
DEBUG:root:471  0   50000   49860   0   21.9    2.5 0.0
DEBUG:root:771  0   50000   49719   0   34.4    2.5 0.0
DEBUG:root:1074 0   50000   49516   0   46.9    2.5 0.0
DEBUG:root:1380 0   50000   49250   0   59.4    2.5 0.0
DEBUG:root:1675 0   50000   48923   0   71.8    2.5 0.0
DEBUG:root:1966 0   50000   48533   0   84.3    2.5 0.0
```

Рис. 11 Журнал симулятора

В ньому представлені основні показники для відображення в консолі: id літака, X координата, Y координата, ALT- висота, SPD - швидкість, ACCEL - потужність, VSPD – вертикальна швидкість).

Наступним кроком є запуск симулятора, щоб проілюструвати його роботу.

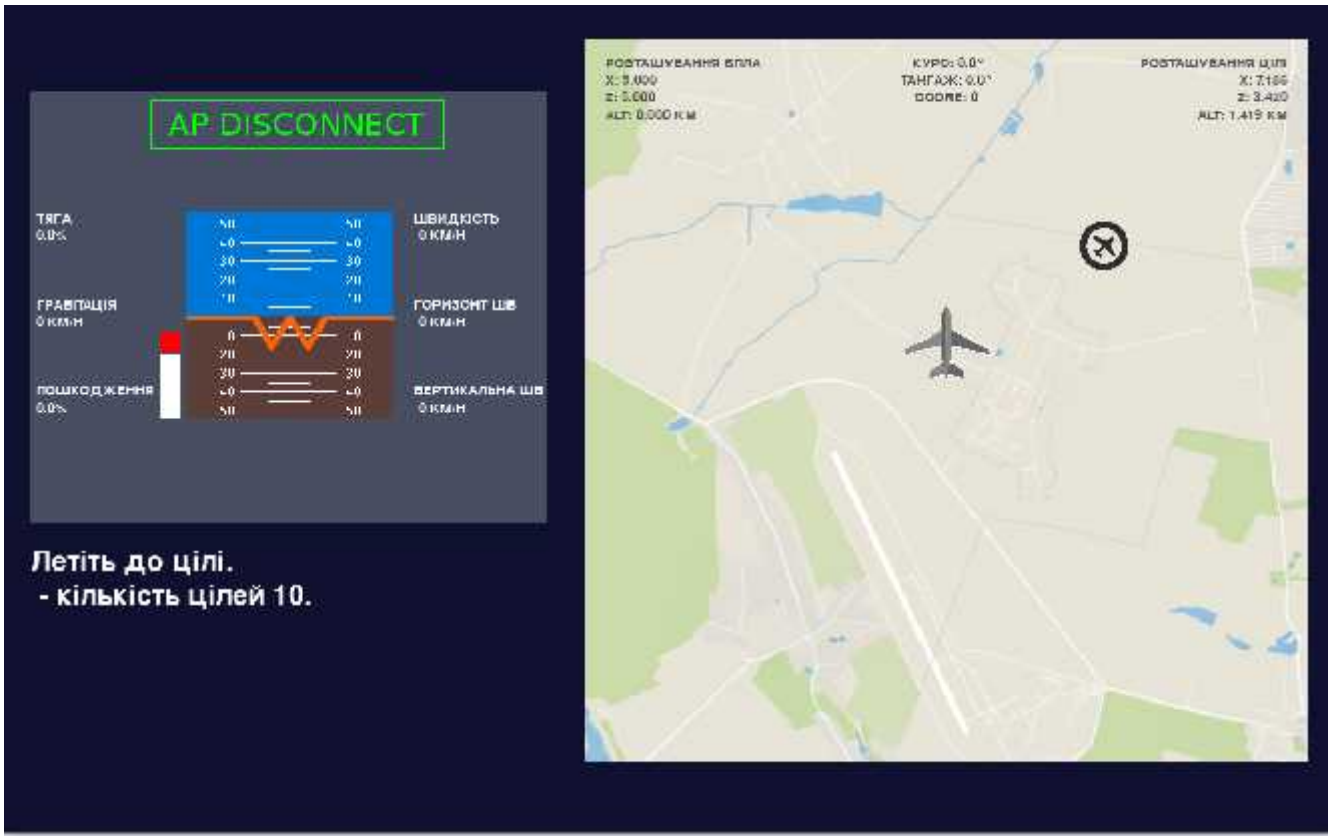

Рис. 12. Зображення вікна симуляції

Як бачимо вікно симуляції (зображено на рис. 12) містить основні показники про БПЛА, які надають оператору необхідну інформацію для керування та моніторингу польоту. Нижче наведено детальний опис цих показників:

Тяга двигунів - це сила, яку генерують двигуни бпла.

Швидкість - це швидкості руху бпла відносно середовища, в якому він перебуває. Вона виражається в одиницях відстані, таких як метри на секунду (м/с) або кілометри на годину (км/год).

Горизонтальна швидкість - це компонента швидкості, що вказує на швидкість руху в горизонтальному напрямку.

Вертикальна швидкість - це компонента швидкості, що вказує на швидкість руху в вертикальному напрямку, вгору або вниз.

Швидкість вільного падіння - це швидкість, з якою об’єкт вільно падає під впливом сили тяжіння.

Пошкодження - це більш ігровий параметр, який відображає технічний стан бпла.

Також як бачимо симулятор має штурвал для відображення положення горизонту відносно бпла, він дозволяє оператору визначати орієнтацію та стабільність бпла під час польоту.

З правого боку розташована карта повітряного простору, яка надає відображення розташування бпла та його цілей. На карті відображаються символи, що позначають положення бпла та цілей у віртуальному повітряному просторі. Крім того, на карті вказуються координати місця, на якому знаходиться бпла, поточна висота, кут тангажу (кут нахилу вздовж поперечної вісі), кут курсу (кут повороту в горизонтальній площині) та ігровий параметр, що показує кількість пройдених цілей. Ці дані допомагають пілоту візуалізувати положення та стан бпла у віртуальному просторі, сприяють зручному навігації та досягненню поставлених цілей.

Після завершення симуляції будується відповідний графік основних показників під час польоту (Рис. 13). Графік відображає зміну значень чотирьох параметрів протягом часу польоту.

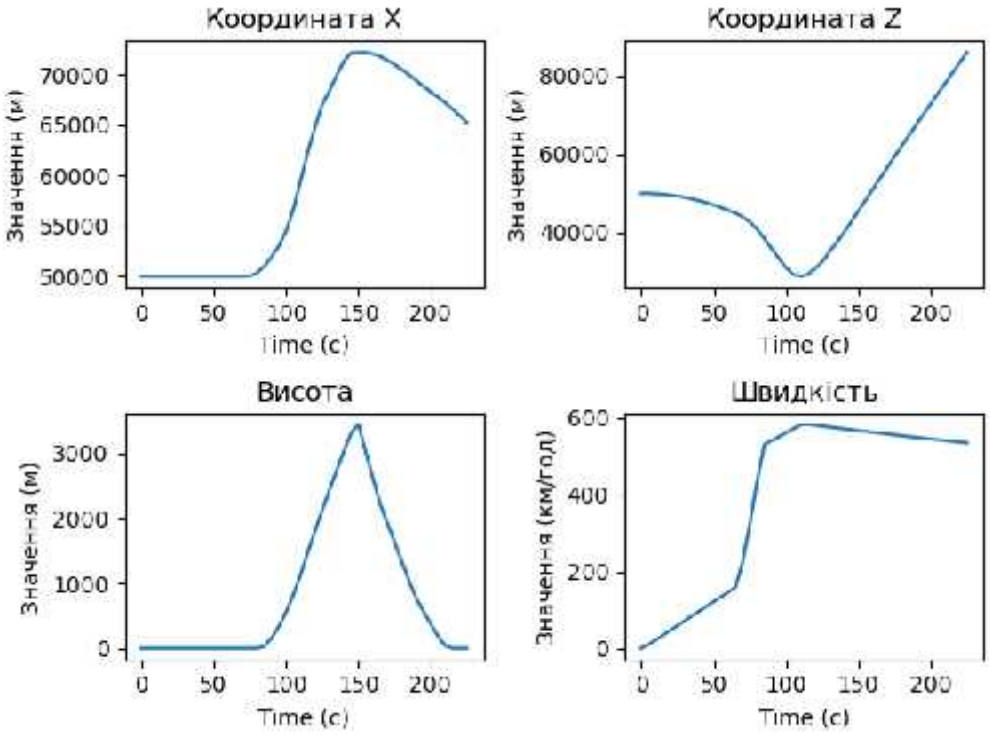

Рис. 13. Графік зміни параметрів протягом всього польоту

Перший графік представляє залежність координати X (горизонтальна компонента місця) від часу. Вісь X відповідає часу у секундах, а вісь Y показує значення координати X у метрах. Цей графік дозволяє візуально відслідковувати зміну горизонтального положення об’єкта протягом польоту.

Другий графік відображає залежність координати Z (вертикальна компонента місця) від часу. Вісь X показує час у секундах, а вісь Y показує значення вертикальної компоненти місця у метрах. Цей графік дозволяє спостерігати зміну вертикальної компоненти місця об’єкта протягом польоту.

Третій графік відображає залежність висоти польоту від часу. Вісь X представляє час у секундах, а вісь Y показує значення висоти польоту у метрах. Цей графік дозволяє аналізувати зміну загальної висоти польоту об’єкта протягом часу.

Четвертий графік відображає залежність швидкості польоту від часу. Вісь X представляє час у секундах, а вісь Y показує значення швидкості польоту в метрах за секунду. Цей графік дозволяє візуально аналізувати зміну швидкості об’єкта протягом польоту.

Графіки основних показників польоту надають важливу інформацію про рух об’єкта під час симуляції. Вони можуть бути використані для аналізу та визначення закономірностей у поведінці бпла, а також для порівняння різних симуляційних сценаріїв.

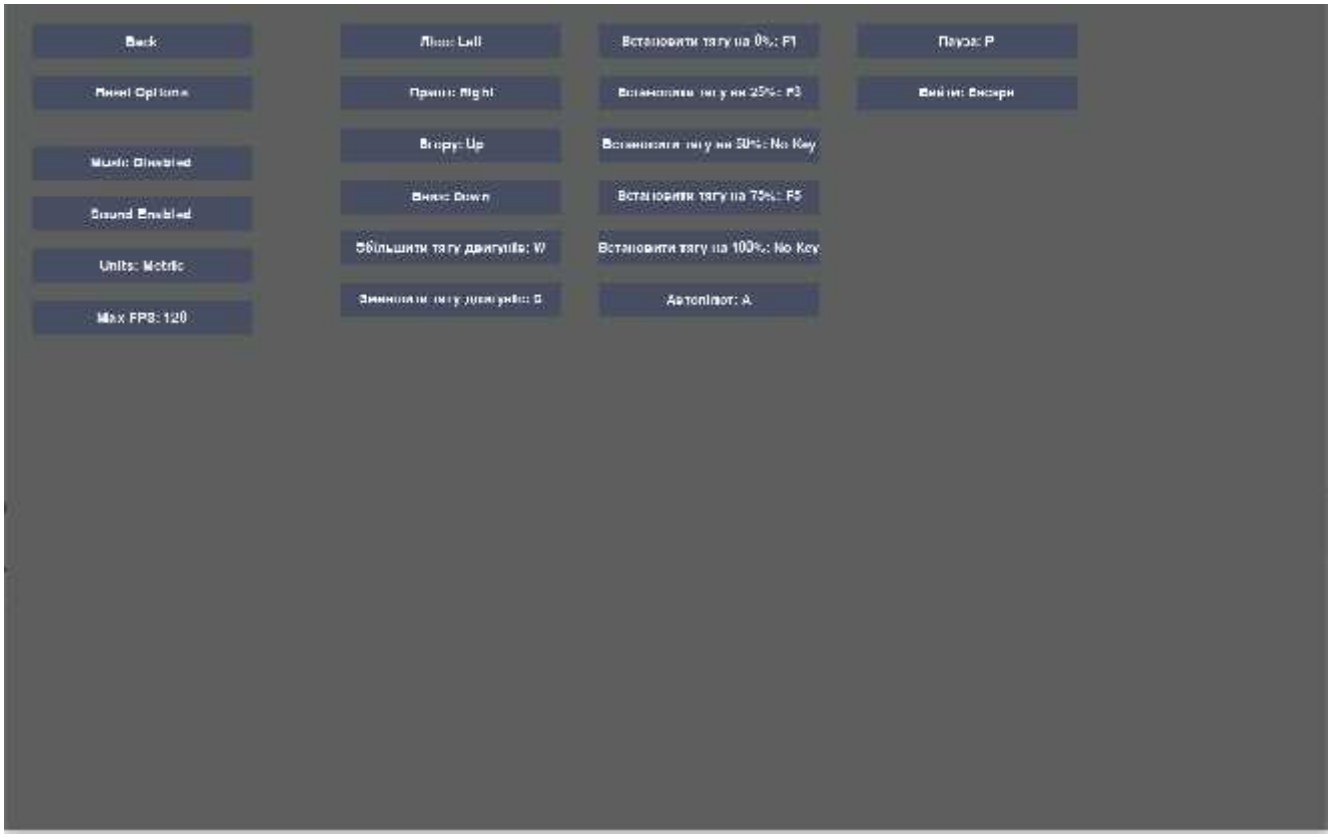

Рис. 14. Зображення вікна налаштувань

Вікно налаштувань (зображено на рис. 14), містить наступні параметри:

- Back: кнопка, яка повертає до попереднього екрану.
- Reset Option: кнопка, яка скидає налаштування до значень за замовчуванням.
- Music (вимкнено, увімкнено): параметр, який дозволяє управляти відтворенням музики.
- Sound (вимкнено, увімкнено): параметр, який дозволяє управляти звуковими ефектами в тренажері.
- Units (SI, Metric...): параметр, який дозволяє вибрати одиниці вимірювання для відображення даних (наприклад, система одиниць SI або метрична система).

- Max FPS: параметр, який дозволяє обмежити кількість кадрів на секунду для поліпшення продуктивності.
- Pause: кнопка, яка призупиняє час в симуляторі.
- Exit: кнопка, для виходу, вона закриває симулятор.

Налаштування управління бпла дозволяють змінити прив’язку кнопок до конкретних дій у тренажері. Наприклад:

- Left: кнопка, яка відповідає за рух вліво.
- Right: кнопка, яка відповідає за рух вправо.
- Up: кнопка, яка відповідає за рух вгору(збільшити кут тангажу).
- Down: кнопка, яка відповідає за рух вниз(зменшити кут тангажу).
- Збільшити тягу (W): кнопка, яка збільшує тягу двигуна.
- Зменшити тягу двигунів (S): кнопка, яка зменшує тягу двигуна.
- Автопілот (A): кнопка, яка активує режим автопілота бпла.
- Встановити тягу на 0%, 25%, 50%, 75%, 100%: кнопки, які встановлюють тягу двигуна на відповідний відсоток від максимальної потужності.

Ці налаштування дозволяють користувачеві змінити клавіші та прив’язки до певних дій у тренажері залежно від їх власних вподобань та комфорту.

ВИСНОВКИ

Ця робота присвячена теоретичним основам, проєктуванню та розробці комп'ютерного тренажера. Метою цього проєкту було розробити концепт тренажера дрона в 2D просторі, який може використовуватися для навчання та тренування людей без реального досвіду пілотування.

Основним завданням було виявити можливості реалізації подібного проєкту з використанням мови програмування Python та бібліотеки Pygame. Цей дослідницький процес дозволив мені збагнути архітектуру симуляторів, моделювання фізичних параметрів та обробку введення користувача.

У процесі розробки тренажера були використані різні підходи, включаючи роботу з 2D графікою, звуком та настройками моделі БПЛА та повітряного простору. В результаті було створено імітаційну систему, яка забезпечує відображення основних параметрів бпла, таких як тяга двигунів, швидкість, горизонтальна та вертикальна швидкість, швидкість вільного падіння та технічний стан БПЛА. Також, була розроблена карта повітряного простору з відображенням розташування БПЛА, цілей, координат місця, висоти, кута тангажу, кута курсу та кількості пройдених цілей.

В цій роботі я також приступив до розуміння і практичного використання симуляторів як інструменту навчання та тренування. Використовуючи мову програмування Python, я зрозумів, як створити досить реалістичний тренажер, який може наблизити користувача до віртуального політного досвіду.

Перспективи цього проєкту можуть бути не дуже великими, оскільки він є лише концептуальним тренажером і не претендує на повну точність і реалізм. Проте, він може мати певне застосування в контексті навчання та демонстрації того, як саме реалізуються симулятори на прикладі простого тренажера.

Цей проєкт може використовуватися для набуття базових знань та розуміння основних принципів розробки симуляторів. Незважаючи на його обмеження, концептуальний тренажер може послужити важливим інструментом для початківців, що бажають зануритися в світ симуляцій та отримати уявлення про те, як вони працюють. Він може допомогти користувачам отримати загальне уявлення про основні принципи фізики, управління і візуалізації в контексті симуляції.

Оскільки проєкт є простим і спрощеним, його також можна використовувати для демонстрації розробки та реалізації симуляторів на базі мови Python. Це дає можливість вивчити основи розробки симуляційних програм та розширити свої знання у цій області.

Враховуючи ці аспекти, проєкт може виконувати свою основну мету - показати принципи розробки симуляторів на прикладі простого тренажера і дати користувачам загальне розуміння цієї області.

Загалом, ця робота дозволила мені поглибитися в сферу симуляцій та розробки комп'ютерних тренажерів, а також здобути практичні навички у розробці імітаційних систем. Вона також підтвердила важливість моїх знань у програмуванні та розумінні основ симуляцій для створення реалістичних віртуальних середовищ для навчання та тренування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Application Specific Drone Simulators: Recent Advances and Challenges Aakif Mairaj, Asif I. Baba, Ahmad Y. Javaid.

[2] lker Bekmezci, E. entrk, T. Trker, Security Issues in Flying Ad-hoc Network (FANETs), Journal of Aeronautics and Space Technologies 9 (2), 2016.

[3] Офіційний сайт DJI [Електронний ресурс] - Режим доступу : https://www.dji.com/global/simulator

[4] Офіційний сайт X-plane 12 [Електронний ресурс] - Режим доступу : https://www.x-plane.com

[5] Unmanned aircraft system simulation [Електронний ресурс] - Режим доступу : https://en.wikipedia.org/wiki/Unmanned_aircraft_system_simulation

[6] D. Allerton, Principles of flight simulation, John Wiley & Sons, 2009.

[7] B. Kate, J. Waterman, K. Dantu, M. Welsh, Symbiotic: A simulator and testbed for micro aerial vehicle swarm experiments, 2012.

[8] S. Shah, D. Dey, C. Lovett, A. Kapoor, Airsim: High-fidelity visual and physical simulation for autonomous vehicles, in: Field and Service Robotics, 2.

[9] Офіційна документація Python: Python Software Foundation [Електронний ресурс] - Режим доступу : https://docs.python.org/3/

[10] Офіційна документація Pygame: Pygame Community [Електронний ресурс] - Режим доступу : https://www.pygame.org/docs/