МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.
"_____"_____2021 р.

ДИПЛОМНИЙ ПРОЕКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: «Модуль керування рухом БПЛА за наявності перешкод на маршруті польоту»

Виконавець: Ященко Костянтин Анатолійович

Керівник: Глазок Олексій Михайлович

Нормоконтролер: Тупота Євгеній Вікторович

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Освітнього ступеня бакалавр
Напрям (спеціальність) 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ О.Є.Литвиненко

«_____» _____2021 р.

ЗАВДАННЯ

на виконання дипломного проекту

Ященка Костянтина Анатолійовича
(П.І.Б. випускника)

1. Тема роботи: «Модуль керування рухом БПЛА за наявності перешкод на маршруті польоту»

затверджена наказом ректора від «04» лютого 2021 року №135/ст

2. Термін виконання проекту (роботи): з 17.05.2021 по 20.06.2021

3. Вихідні дані до проекту: завдання на дипломний проект, програмні аналоги для відображення керованих рухомих графічних об'єктів, керування БПЛА, мова програмування *C++*, мультимедійна бібліотека *SFML*

4. Зміст пояснювальної записки:
1) огляд предметної області;
2) проектування модуля;
3) розробка модуля;

5. Перелік обов'язкового ілюстративного матеріалу:
1) блок-схема загального алгоритму роботи графічних кнопок модуля;
2) блок-схема загального алгоритму зчитування даних з файлу;
3) блок-схема загального алгоритму керування БПЛА;
4) блок-схема загального алгоритму фрагментарного показу карти;

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Написати перший розділ пояснювальної записки	17.05.2021 – 18.05.2021	
2	Сформувати попередню структуру модуля	19.05.2021 – 20.05.2021	
3	Написати другий розділ пояснювальної записки	21.05.2021 – 22.05.2021	
4	Розробити модуль	23.05.2021 – 02.06.2021	
5	Написати третій розділ пояснювальної записки	03.06.2021 – 04.06.2021	
6	Завершити оформлення пояснювальної записки	05.06.2021 – 06.06.2021	
7	Підготувати графічні матеріали та презентацію	09.06.2021 – 13.06.2021	

7. Дата видачі завдання: 17.05.2021р.

Керівник дипломної роботи (проекту) _____ Глазок О.М.

(підпис керівника) (П.І.Б.)

завдання прийняв до виконання _____ Ященко К.А.

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Модуль керування рухом БПЛА за наявності перешкод на маршруті польоту» містить: 55 с., 22 рис., 10 літературних джерел.

SFML, КООРДИНАТИ, МУЛЬТИМЕДІА, СПРАЙТ, ТРАЄКТОРІЯ, ЗЧИТУВАННЯ.

Об’єкт дослідження – керування БПЛА за розрахованою траєкторією з використанням графічного двовимірного відображення.

Предмет дослідження – модуль керування рухом БПЛА за наявності перешкод на маршруті польоту.

Мета дипломного проекту – створення модуля керування рухом БПЛА за наявності перешкод на маршруті польоту.

Результати: розроблено модуль керування рухом БПЛА за наявності перешкод на маршруті польоту, який може коректно відображати політ БПЛА за траєкторією.

Практичне значення: модуль дозволяє будувати безпечну траєкторію польоту БПЛА в двох вимірах та по розрахованим даним виводити візуальне відображення польоту, що наочно дозволить оцінити результат дій модуля.

Галузь застосування: логістика, моніторинг, розробка симуляторів та комплексів керування.

Прогнозні припущення щодо розвитку об’єкта дослідження: покращення функціональних можливостей, оновлення графічного інтерфейсу в бік інтуїтивно зрозумілої будови та призначення, підтримка кросплатформності.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

SFML – *Simple and Fast Multimedia Library*

VS2017 – *Visual Studio 2017*

MIT – *Massachusetts Institute of Technology*

БПЛА – безпілотний літальний апарат

MPEI – макет розташування елементів інтерфейсу

ПК – персональний комп’ютер

ОС – операційна система

ВСТУП

Людство з давніх часів прагнуло удосконалювати та відточувати навички, які допомагали б у виживанні: будь то мисливство чи мирна обробка земель. З часом наші предки створювали все складніші знаряддя праці за будовою, але або простіші у використанні, або ефективніші у результаті, або навіть й те й інше одразу. Таким чином людина здавна намагалася зробити власне існування комфортним, а виконання потрібних для цього задач – максимально простим та ефективним для себе. Це прагнення актуальне й тепер.

Можна довго дискутувати щодо корисності тих чи інших винаходів, але з впевненістю можна стверджувати, що переважна більшість з них має робочу цінність лише тоді, коли людина приймає безпосередню участь у застосуванні, і йдеться саме про основну діяльність, а не про обслуговування чи створення такого винаходу. Наприклад, якщо взяти морську справу, то для того, щоб перепливти з одного острову далеко на інший слід мати хоча б на чому плити, та з допомогою чого орієнтуватися в морі. Нехай такий мандрівник матиме човен з веслами та компас, якими вміє користуватися. Виходить, що мандрівник використовує порівняно просту будову системи «човен-весла» та реакцією стрілки компаса на магнітне поле Землі (географічна Північ). Але зв’язок між цими знаряддями здійснює саме людина, крім того, навіть якщо б модифікувати човен таким чином, щоб той плив лише по механічним показам компасу, це не вирішить проблему базового керування, наприклад реакцію на перешкоди або покращення курсу в цілому. Це вказує на те, що система такого складу не зможе виконувати без участі людини алгоритми навіть невисокої складності. Такий приклад може здатися смішним та легковирішуваним у сучасних реаліях, але ж були часи, коли прогрес людства дозволяв вважати компас або навіть човен «верхом» інженерної думки, і повсякчас користуватися саме таким набором для мандрівок.

З плином часу люди створювали все складніші системи, де взаємодії між елементами залишали все менше «місця» для участі людини. При створенні ЕОМ люди об’єднували чи не всі доступні знання з досліджуваних сфер. Таким чином створена система могла хоч поки й не осмислено, але виконувати ряд обчислень, які зазвичай людина робить довше. З розвитком ЕОМ такі системи вже здатні обробляти величезні масиви даних, а що найголовніше – без участі людини під час основних обчислень. Людина поки потрібна для таких систем або для перевірки/тестування правильності роботи системи при початковому введені у експлуатацію, для обслуговування, як джерело, або як наступна ланка для передачі оброблених даних.

Крім збільшення надійності, точності та швидкості обчислень ЕОМ, її складові також з часом через розвиток технологій набувають все більшу портативність, компактність. Це дозволяє зробити застосування більш поширеним та вирішує багато проблем, зв’язаних з фактичними габаритами та вагою таких систем.

В результаті сьогодні ми маємо можливість використовувати порівняно потужні системи в невеликих, здатних до портативності корпусах. А з розвитком елементів живлення, можливо використовувати описаного виду ЕОМ ще й без залежності від електричних мереж, використовуючи, наприклад альтернативні джерела енергії як сонячна, вітрова, тощо. Крім того теперішні технічні сенсори, як фото-відео реєстратори й інші датчики сучасного порядку зможуть забезпечити якісне забезпечення таких систем різного роду даними, що допоможе як у обробці інформації зі зовнішніми джерелами, так і у внутрішній діагностиці. Все це дає змогу створити різного роду технічні засоби, що не лише швидше й точніше людини зможуть виконувати обчислення, а й зможуть дістатися таких місць, та функціонувати в таких місцевостях або навіть середовищах, які важкодоступні або й зовсім недоступні безпосередньо для людей.

Дана робота створюється з метою забезпечення надійними алгоритмами розрахунку траєкторії руху якраз таких ЕОМ, вид яких за свої особливості названо БПЛА. Для попередньої перевірки та/або наочного представлення

польоту по визначеній траєкторії дана робота також передбачає забезпечення відповідного графічного відображення.

Об’єкт дослідження – керування БПЛА за розрахованою траєкторією з використанням графічного двовимірного відображення.

Предмет дослідження – модуль керування рухом БПЛА за наявності перешкод на маршруті польоту.

Мета дипломного проекту – створення модуля керування рухом БПЛА за наявності перешкод на маршруті польоту.

Розрахунок траєкторії руху є основним процесом для переміщення БПЛА зі стартової точки польоту в кінцеву.

Щоб досягти вказаної мети, треба виконати такі пункти:

- зрозуміти важливість та актуальність БПЛА, виконати аналіз існуючих програмних аналогів
- сформулювати вимоги та попередню будову модуля
- обрати мову програмування та бібліотеки для написання модуля
- створити модуль

Для виконання наведених пунктів дипломну роботу поділено на чотири розділи:

Перший розділ про важливість БПЛА та про аналогічні за функціоналом програми

Другий розділ містить попередні підготування до фактичного створення модуля (формування цілей, вимог та будови).

Третій розділ має опис створення та пояснення до основних програмних частин модуля.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Важливість та актуальність БПЛА

Створення безпілотних літаючих апаратів зараз перетворюється з вузьконаправленого сегменту розробки та виробництва у масову індустрію, що має величезний потенціал для більшості сфер діяльності людини. Зрозуміло, що такі дрони вже можуть приносити користь самим пересуванням у повітряному просторі та дистанційною взаємодією з системою управління, але щодо різних специфічних завдань все дещо складніше.

Багато сучасних БПЛА оснащені не лише потрібними для польоту, живлення та комунікації апаратними компонентами, а й продвинутою програмною частиною, яка не просто на базовому рівні обслуговує системи дрона, а й спеціалізується на певних інтелектуальних задачах – тобто такі БПЛА мають в своєму складі спеціалізовані програмні засоби для тієї чи іншої діяльності.

Звісно, перше на думку приходить універсальне рішення – встановити на такі машини систему штучного інтелекту, яку можна потім навчити виконувати різні види робіт, крім того – можливість самовдосконалення такою системою своїх алгоритмів може зекономити чимало коштів та часу, а дрони зробити універсальними за напрямком діяльності. Але в силу того, що розробка ШІ хоч і має чималі успіхи, але вона, по-перше ще не дозволяє в потрібній мірі здійснювати всестороннє навчання, яке знадобиться для швидкого «перепрофілювання» дронів, а по-друге – подібні системи з універсальним ШІ

потребуватимуть не лише відносно ефективної (та дорогої) апаратної складової для обробки даних на борту, а й відповідне енергетичне забезпечення, тобто ефективний елемент живлення – який в сучасних реаліях і сильно гріється (проблеми охолодження системи), і робить дрон важким та неповоротким (за рахунок нелегкої маси такого елементу). Звичайно, зараз розробляються прототипи й з іншими елементами живлення (замість розповсюджених літій-іонних чи літій-полімерних акумуляторів використовується, наприклад, водневе паливо), але на даний момент в серійному виробництві БПЛА доводиться зберігати тонкий баланс між масою, маневреністю та конструкцією, а як наслідок й часом автономної роботи.

Тож поки універсальний ШІ та збалансоване джерело живлення знаходяться на етапі розробки, повернемося до ідеї спеціалізованих програмних засобів. Таке ПЗ розробляється для конкретних предметних областей, і навіть для окремих задач, та є добре класифікованим. Не є сюрпризом, що таке ПЗ розроблюється й для БПЛА залежно від майбутньої діяльності такої машини. Важливо зазначити, що кінцева обробка специфічних даних не обов’язково має виконуватися силами самого дрона, добуті дані будуть передані до ланки подальшої обробки, для якої вже не буде критичним фактор автономної роботи чи відносно обмежених обчислювальних потужностей, що б стало на заваді проведення таких операцій силами лише безпілотника. Але навіть беручи до уваги такий процес отримання, перенесення та обробки даних потрібно, щоб всі ланки обробки діяли злагоджено та безпомилково – початкові дані добуті дроном мають бути перетворені в кінцеву форму, а це досягається саме комплексними рішеннями спеціалізованих програмних засобів (СПЗ), які діють як в БПЛА, так і в інших ланках (якщо такі є).

Можна також навести такий перелік основних напрямів розробки СПЗ для безпілотників:

1. Контроль взаємодії між БПЛА як мінімум дозволяє елементарно повідомляти дронам про свою присутність для уникнення зіткнень між ними, або в повній мірі, якщо мати доступ до групи дронів це дозволить створювати цілі

мережі з БПЛА, які можуть формувати «рій». Розробкою даного ПЗ зайняті *Airware* та *Yuneec* при спонсорстві *Intel*. Також залучено компанію *AT&T* для інтеграції мереж *4G* для обміну інформації між безпілотниками.

2. Виявлення та попередження зіткнень з перешкодами вже може й не передбачати двостороннього обміну інформацією для налагодження дій. Крім того, у різних країнах можуть бути різні правила щодо пересування в повітряному просторі, а ще може знадобитись інтеграція БПЛА в певні авіадиспетчерські системи. Розробкою відповідного СПЗ займаються *Airware* та *PixiePath*. Таке програмне забезпечення передбачає не лише системи ручного керування та аналітики, а й алгоритми ШІ для автономного функціонування безпілотника. В даній дипломній роботі розроблюється модуль, який саме відноситься до даного пункту.

3. Обробка добутих даних. Як вже було зазначено, йдеться про категорію даних, які не є критично важливими для систем дрона, так як для такої базової навігації розробляється СПЗ з пункту 2. Наприклад, *Pix4D* розробила СПЗ, що допомагає безпілотникам здійснювати безперервну топографічну зйомку в різних умовах, воно перетворює масив фото відзнятих під час польоту у двовимірні знімки, та тривимірні моделі місцевості.

Можна з упевненістю сказати, що у безпілотників є величезний потенціал, бо навіть зараз видно їх надзвичайну ефективність, а питання подібних розробок є як ніколи актуальним особливо зараз – під час пандемії, коли технології дистанційної взаємодії є життєво важливими.

1.2. Аналіз переваг та недоліків існуючих програмних аналогів

1.2.1. *Unreal Engine 4*

Unreal Engine 4 – мультимедійний рушій, що підтримується компанією-розробником *Epic Games*. Цей рушій має чи не найкращий набір опцій, який відповідає за відображення графічних об’єктів (особливо в тривимірному

віртуальному просторі), що пов’язано з гарно прописаною системою освітлення. Останнім часом багато розробників використовують його для розробки комерційних проектів, спрямованих на сферу розваг та різного роду симуляторів. *Unreal Engine 4* має вбудований візуальний редактор (рис. 1.1), що дозволяє взаємодіяти користувачеві з рушієм під час розробки на більш високому рівні, тобто налаштовувати більшість параметрів не пишучи програмний код вручну, а користуючись зручним графічним інтерфейсом редактора.

Рис. 1.1. Загальний вигляд редактора в *Unreal Engine 4*

Щодо логіки застосунків, то рушій надає два варіанти щодо їх створення: стандартне прописування логіки на мові програмування *C++*, та власну альтернативу – *Blueprint*, що являє собою систему візуального скриптинга (рис. 1.2). Скрипти – це код, редагування або видалення якого не потребує того ж від системи. Програма – це код, який якраз і являє собою цю систему, і при редагуванні якого те ж потребується від системи. По суті, другий варіант дозволяє використовувати майже весь потенціал програмування через візуальні блоки, таким чином піднімаючи процес створення логіки програми на ще вищий рівень, та тим самим збільшуючи ефективність створення того чи іншого програмного

додатку. Недоліком такого варіанту є певна неуніверсальність, так як *Blueprint* як такий використовується лише в даному рушієві. *Unreal Engine 4* крім створення тривимірних об’єктів може використовуватися й для *2D*.

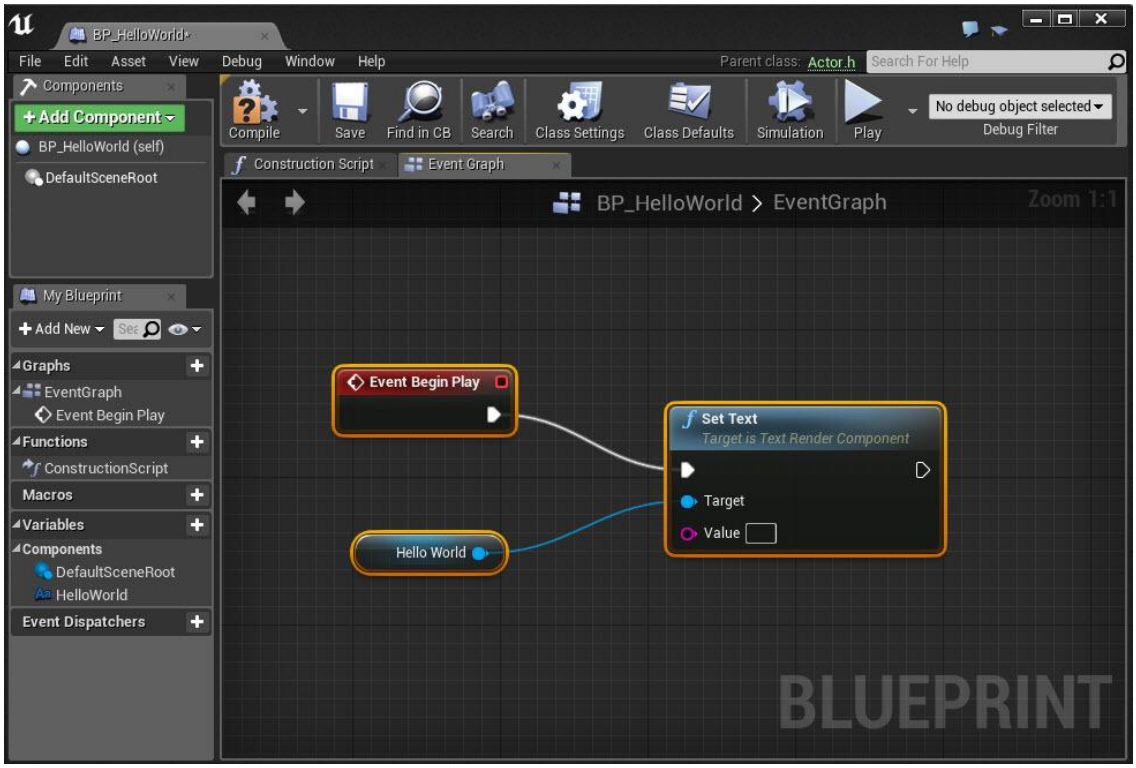

Рис. 1.2. Скриптинг через візуальну систему *Blueprint*

Переваги:

– Гарна оптимізація для сучасних ПК та ігрових приставок
– Велика кількість документації та активна підтримка розробниками
– Одне з найкращих обробок зображення в сфері мультимедійних рушіїв
– Наявність візуальної системи скриптингу *Blueprint* за замовчуванням
– Безкоштовне користування повною версією до отримання прибутку 3000$ в квартал, далі потрібно відраховувати 5% розробникам рушія
– Відкритий вихідний код рушія

Недоліки:

– Малорозвинена розробка продуктів під мобільні ОС (*Android, iOS*)
– Інтерфейс середньої складності користування

1.2.2. *Unity*

Даний рушій популярний завдяки можливості розробки програм на різні платформи, також з ним можливо створювати веб-застосунки. *Unity* користується домінуючим попитом серед розробників ігор, особливо під мобільні платформи. Рушій вирізняється серед інших частими оновленнями, непогано спроектованим інтерфейсом редактора (рис. 1.3) та підтримкою готових рішень для комерційної складової в своїй додатках, тобто готові рішення у вигляді різного роду плагінів для розробки: реклама, аналітика, внутрішні покупки, ігрові центри й інше. Саме тому *Unity* здобув популярність особливо серед розробників мобільних додатків.

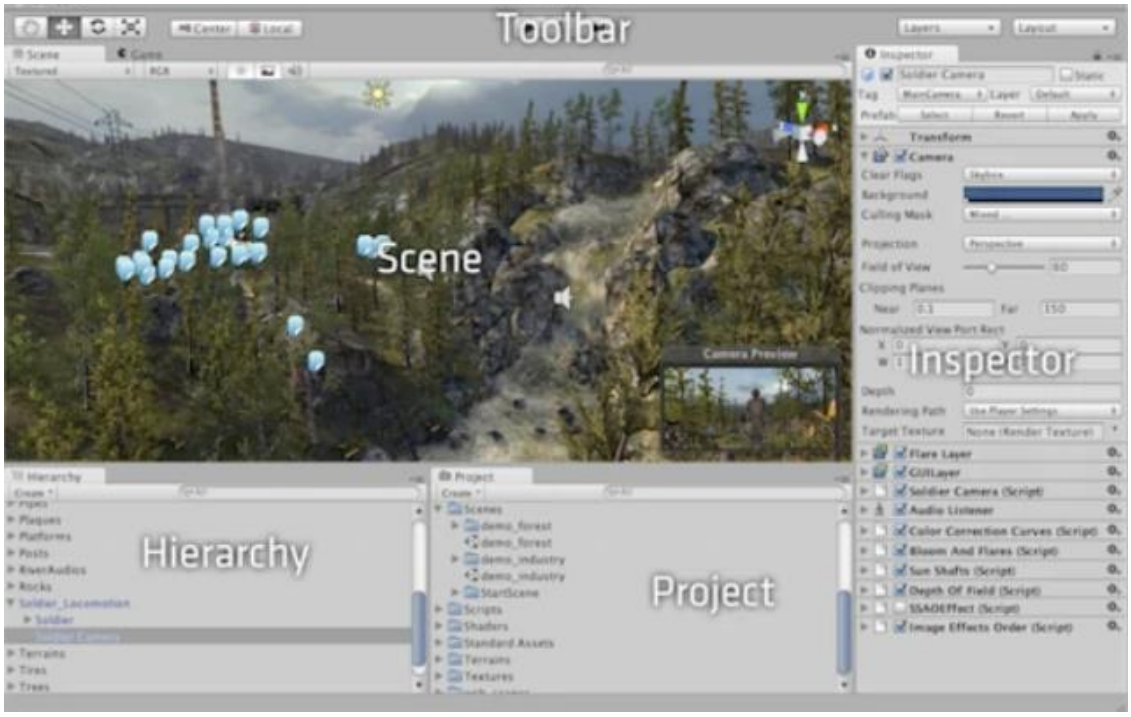

Рис. 1.3. Інтерфейс редактора *Unity*

В плані графіки даний рушій задовольняє потребу користувачів у приємній для ока картинці, але все ж для проектів з кінематографічною графікою він не підійде. З іншого боку, якщо згадати можливість направленності на портативні, й в більшій мірі не такі потужні пристрої на базі ОС *Android* чи *iOS*, та враховуючи їх фактичні розміри екранів, то не така висока деталізація зображення може стати навіть перевагою. А враховуючи можливість створення повноцінної *2D*-графіки з

допомогою рушія, тоді його можливості видачі зображення взагалі зрівнюються з тим же *Unreal Engine 4*.

Щодо можливих способів створення програмного коду, то підтримуються мови *C++*, *C*, *JavaScript* й деякі інші. Але якщо працювати з рушієм потрібно на професійному рівні, доведеться добре оволодіти мовою *C#*.

Присутній аналог *Blueprint* під назвою *playMaker* для *Unity*, але на відміну від першого, *Playmaker* розповсюджується як розширення для редактора, та ще й на платній основі.

Рис. 1.4. Вигляд візуального скриптингу з розширенням *playMaker* для *Unity*

Сам рушій має умовно безкоштовну модель поширення: якщо користувач отримує прибуток менше 100 000*$* в рік з продажу програмного продукту, створеному на поточному рушієві, тоді користувач має повне право користуватися базовою (неповною версією) *Unity* безкоштово. В іншому випадку потрібно купувати повну версію за 1500*$* єдиноразово, або користуватися підпискою, що коштує 75*$* в місяць.

Незважаючи на очевидні переваги, є й доволі неприємний недолік, в якому солідарні майже всі користувачі даного рушія – відсутність своєчасного усунення помилок, привнесених під час оновлень. Таким чином, якщо проблема розробки того чи іншого застосунку пов’язане саме з інструментарієм *Unity*, тоді доводиться або по можливості переписувати код створений користувачем, щоб не

зачіпати проблемні засоби рушія, або й взагалі зупиняти розробку допоки розробники не усунуть відповідні помилки з наступними оновленнями.

Переваги:

– Величезна кількість довідкових джерел щодо розробки на поточному рушієві
– Розвинений напрям створення мобільних додатків
– Легке вбудовування готових рішень для монетизації проектів
– Велика кількість цільових платформ для розробки
– Неперевантеженість основного інтерфейсу

Недоліки:

– Базова неповна по можливостям версія
– Хоч регулярні, але часом проблемні оновлення
– Відсутність можливості візуального скриптингу за замовчуванням
– Закритість вихідного коду рушія

1.2.3. *Gamemaker Studio 2*

Рушій створено для користувачів, які не мають можливості чи бажання вчити складні мови програмування. Тобто сам рушій максимально простий при створені логіки та графічної частини застосунків. Дуже часто для взаємодії з інтерфейсом тут використовується спосіб взаємодії *Drag-and-Drop* (з англ. «тягни-та-кидай», фактично це перетягування графічних елементів, наприклад комп'ютерною мишкою). Таким чином робота з *Gamemaker Studio 2* інтуїтивно зрозуміла. З усіх розглянутих аналогів у даному рушієві найлегший для сприйняття інтерфейс редагування (рис. 1.5), що забезпечить швидке звикання до процесу створення застосунків. Крім того, тут не лише є стандартні інструменти розробки (графіка, звук, мережевий зв'язок), а навіть вбудований графічний редактор редактор, який може працювати, наприклад, зі спрайтами. Спрайт – зображення, яке є окремим графічним об'єктом у застосунку.

Варто також зазначити, що поточний рушій працює в основному лише з *2D*-графікою, хоча є можливість роботи й з *3D*. При цьому слід пам’ятати, що *Gamemaker Studio 2* все-таки *2D*-орієнтований рушій, тому процес розробки тривимірної графіки тут вельми непопулярний та малоописаний, а картинка при обробці об’ємних об’єктів виглядає не дуже добре, плюс процес рендерингу (отримання зображення по моделям за допомогою комп'ютерної програми) не є оптимізованим.

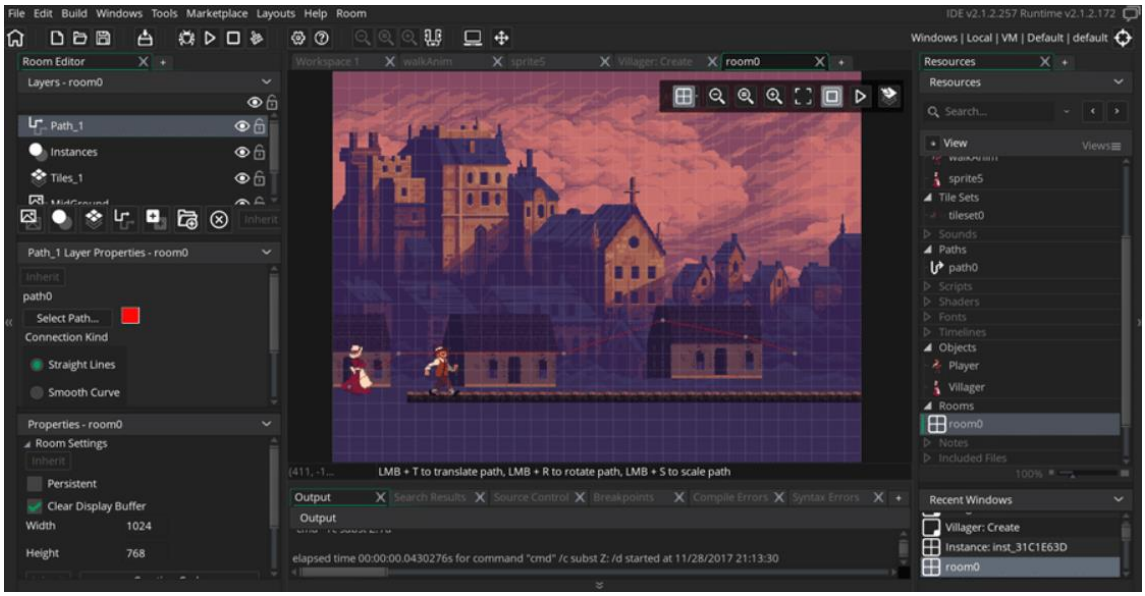

Рис. 1.5 Інтерфейс редактору *Gamemaker Studio 2*

Для створення застосунків на базі даного рушія також доступна його власна мова програмування *GML (GameMaker Language)*. Дана скриптова мова використовує інтерпретатор (програма, що оброблює код по кожному рядку або команді й відразу його виконує; компілятор, до прикладу, виконує код лише після повної його обробки), по синтаксису виявляються схожості з *C, C++* й *JavaScript*. Розробники відмічають, що хоч *GML* є мовою програмування, але набагато простішою, ніж згадані *C* чи *C++*.

Цікаво також те, що поскільки *GML* – це скриптова мова, то саме її застосування та в поєднанні з дружнім до користувача інтерфейсом редактора виконує всі умови для візуального скриптингу, який не завжди присутній у подібних рушіях, але являє з себе дуже комфортний інструмент для створення

коду не лише в текстовому редакторі, а через взаємодію візуальних блоків (рис. 1.6).

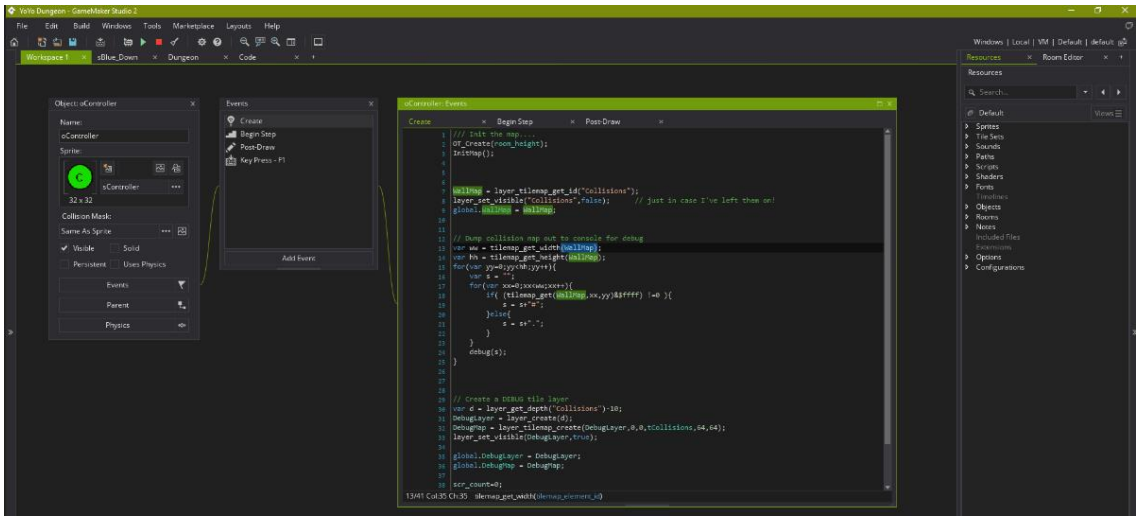

Рис. 1.6. Візуальний скриптинг у *Gamemaker Studio 2*

Одна з особливостей роботи рушія ж є те, що вся графіка для майбутнього застосунку завантажується у відеопам’ять у вигляді атласу – великого зображення, куди внесені всі графічні елементи для відображення в *2D*. Таким чином економляться ресурси системи, яка один раз завантажує всю графіку й протягом роботи застосунку ресурси не «перевідкриваються». Крім того, атласи можна повністю налаштовувати - це знадобиться для розділення атласів на декілька фрагментів для поступового або взаємозамінного завантаження у відеопам’ять, або ж це можна використати при переносі застосунку між різними платформами: наприклад, якщо потрібно перенести програму з систем з великими екранами (ПК, ігрові консолі) на менші екрани (мобільні пристрої), тоді варто зменшити роздільну здатність атласа, що буде генеруватись вже на системах і з меншою відеопам’яттю, і з меншими розмірами екрану.

Gamemaker Studio 2 розповсюджується в трьох версіях: *Standard* (безкоштовна, неповний інструментарій для розробки, найгірше – зникає можливість переносу проектів на більшість систем – губиться кросплатформність, робота лише на системах *Windows*), *Professional* (коштує 100*$*, повний інструментарій, кросплатформність – додається можливість роботи під *macOS*,

Ubuntu та *Android*), та *Master Collection* (ціна 800*$*, кросплатформність та наявність всіх додаткових модулів для розробки).

Переваги:

– Дуже зрозумілий та зручний інтерфейс редактора
– Наявність оптимізації для *2D* графіки
– Активна спільнота користувачів, які завжди готові допомогти
– Велика кількість доступної інформації для користування рушієм
– Власна мова програмування, якою легко розробляти додатки в рамках базових можливостей рушія
– Поскільки вбудована мова скриптова, можливість візуального скриптингу доступна відразу ж

Недоліки:

– Безкоштовна версія рушія дуже «урізана» по можливостях
– Майже повна відсутність зовнішніх бібліотек для розширення інструментарію
– Робота зі шрифтами вельми складна (масштабування, оформлення, тощо)
– Підтримка постобробки зображення хоч і наявна, але не працює належним чином

1.2.4. *Cocos2D-x*

Оригінальний рушій *Cocos2D* створений в 2008 році на мові програмування Python, потім був портований для розробки під iPhone (на мові *Objective C*). Через два роки була випущена перша версія *Cocos2D* для кросплатформної розробки, яка підтримувала мову *C++*. Дана версія отримала назву *Cocos2D-x.*

На даний момент ця версія рушія підтримує розробку на мовах програмування *Java, JavaScript, Lua* та *C#* для створення додатків на різні ОС. Існує також розширення для візуалу *3D* графіки *Cocos 3D*, але як і в випадку з попереднім розглянутим рушієм можливості такого додатку доволі малорозвинені через основну орієнтованість *Cocos2D* на двовимірну графіку.

Цей рушій користується величезною популярністю не тільки у ентузіастів та інді-розробників, а й у гігантів індустрій, як-от *Google*, *Microsoft* та *Intel*. Це зумовлено насамперед тим, що *Cocos2D-x* зберігає «золоту середину»: в першу чергу це добре продумана мультимедійна бібліотека, яка надає можливість роботи з графікою, звуком чи пам'яттю на рівні вже готових компонентів при написанні коду. Але крім того, для даного рушія був створений спеціальний редактор *Cocos Creator* (рис. 1.7), який включає в себе набір необхідних інструментів для анімації, менеджменту ресусів, тощо. Цей редактор офіційно рекомендовано використовувати власне самими творцями рушія. При тому використання наданого розробниками редактора, дебагера, компілятора й іншого інструментарію, який зазвичай нав'язується при встановленні рушіїв - зовсім не обов'язковий. Як середовище можна використати, наприклад, *Visual Studio* від *Microsoft* та підключити *Cocos2D-x* як додаткову бібліотеку до проекту.

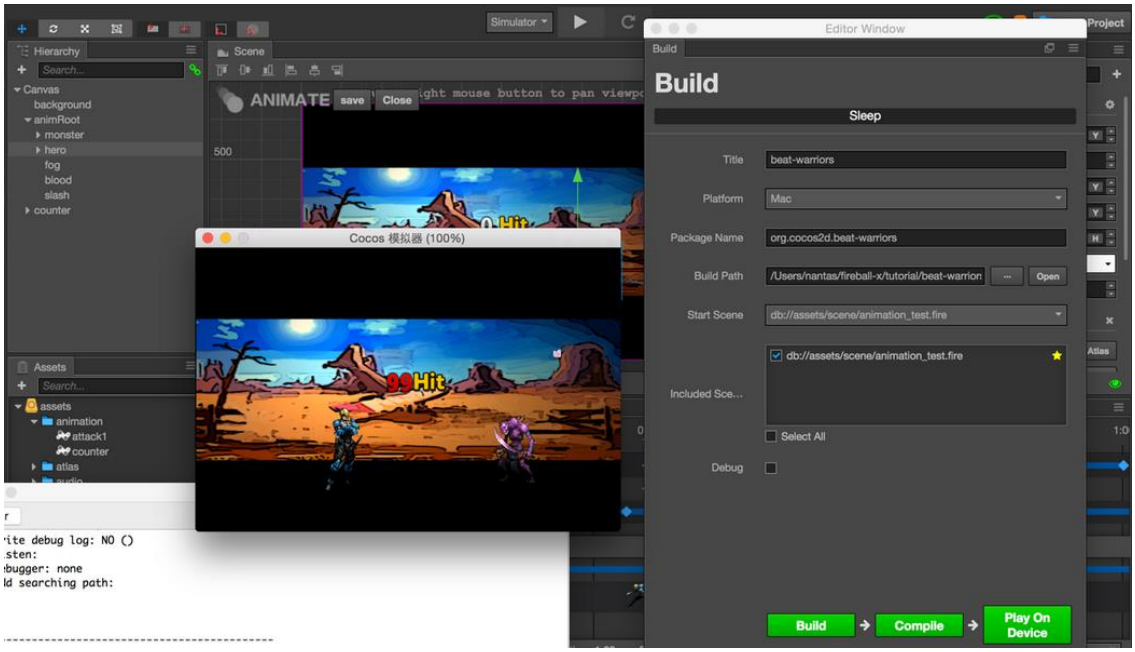

Рис. 1.7. Вигляд інтерфейсу *Cocos Creator*

Така гнучкість використання, безкоштовна та вільна модель розповсюдження й наявність можливості опціонально підключати офіційні інструменти, як «рідний» для рушія редактор з графічним інтерфейсом й

вирізняють *Cocos2D-x* не лише серед інших *2D*-рушіїв, а й серед ПЗ для розробки мультимедійних додатків в цілому.

Переваги:

– Вільна ліцензія використання та розповсюдження (*MIT License*)
– Відкритий вихідний код
– Можливість використання в інших середовищах програмування
– Наявність офіційного програмного редактора від розробників рушія
– Популярність серед гігантів індустрії, як наслідок перспективність
– Порівняно велика кількість мов доступна для програмування з рушієм

Недоліки:

– Відсутність можливості вбудованого перенесення проектів на ігрові приставки
– Порівняно невелика аудиторія користувачів рушія
– Мала кількість офіційних додаткових інструментів та матеріалів для розробки
– Нечасті оновлення рушія

1.3. Висновки до розділу

Перший розділ описує важливість та актуальність БПЛА в сьогоднішніх реаліях. Також були проаналізовані програмні аналоги візуальної частини модуля керування БПЛА, а саме мультимедійні рушії, які призначені для створення та відтворення *3D* чи *2D* графіки у вказаний спосіб та по наданим даним.

Були наведені аргументи на користь підтримки розвитку БПЛА, коротко описані основні труднощі в їх створенні та напрямки застосування БПЛА.

При розбиранні переваг та недоліків програмних аналогів для візуалізації та контролю графічних об’єктів було зроблено висновки щодо розробки майбутнього проекту, а саме:

– Візуалізацію потрібно проводити у віртуальному *2D*-просторі задля простоти сприйняття та меншого навантаження на систему

– Часто у подібних програм перевантажений інтерфейс користувача, тому потрібно використовувати мінімалістичний підхід
– Для меншої трати часу при освоєнні рушіїв з власними середовищами розробки слід використати одну з мов програмування в поєднанні з мультимедійною бібліотекою
– Для проекту з відображенням руху дрону візуальний скриптинг зовсім необов'язковий
– Розробку коду всіх програмних механізмів візуального відображення немає сенсу робити з нуля стандартними можливостями мови програмування, для ефективного використання часу потрібно застосувати графічну бібліотеку
– Рушій має бути простим та ефективним, тому бажано використати графічну бібліотеку, можливо з підтримкою *C* або *C++*

Таким чином майбутній модуль крім базового призначення повинен бути простим у використанні, мати *2D*-візуалізацію польоту БПЛА, простий графічний інтерфейс користувача та автоматично. Бажано по можливості оптимізувати виведення візуальної інформації.

РОЗДІЛ 2
ПРОЕКТУВАННЯ МОДУЛЯ

2.1. Цілі та вимоги до модуля

2.1.1. Цілі, яких повинен досягти модуль

Будь-яка розробка має ціль, досягнення якої сприяє або подальшому розвитку й досягнення наступних поставлених цілей, або завершення процесу розробки для випуску готового програмного продукту. Логічно буде поставити як основну кінцеву ціль назву дипломного проекту: модуль керування рухом БПЛА за наявності перешкод на маршруті польоту. Таким чином модуль повинен забезпечувати рух БПЛА по траєкторії, що буде коректно обходити перешкоди. Тепер треба уточнити декілька моментів щодо розуміння основної цілі розробки.

По-перше, як показувати результати прийняття потрібної траєкторії? Можна просто згенерувати набір послідовних координат перебування дрону і виводити ці дані в інтерфейс командного рядка набором символів в консолі, можна записати дані у текстовий файл для зберігання, або вивести на екран через графічний інтерфейс. Таке виведення дійсно може мати місце для певних задач, як мінімум для відладки та тестування працездатності та правильності роботи модуля. Однак, «сухі» показники буде куди приємніше та наглядніше спостерігати в поєднанні з візуальним супроводом.

По-друге, що з себе представлятиме візуальний супровід? Як вже наводилось, дані можна виводити на екран через графічний інтерфейс модуля у вигляді тих же колонок, тобто просто згруповувати різним чином. Але це все ж ті самі «сухі» дані, які хотілося б чимось доповнити. І якнайкращим доповненням до

цього буде візуалізація процесу польоту БПЛА по вищезгаданим даним. Враховуючи, що дані якраз створюються для побудови правильної траєкторії, чому б відразу не запустити по ній віртуальний безпілотник?

По-третє, важливо розуміти, що поняття «керуючий модуль» та «модуль керування» принципово різні. «Керуючий модуль» - це модуль, який відповідальний безпосередньо лише за керування. «Модуль керування» - це модуль, який включає в себе функцію керування, але може не обмежуватися лише нею. Таким чином, дана дипломна саме про модуль керування, основна задача якого – скерувати безпілотник по правильним координатам, потім відобразити це користувачеві.

Візуально слід обрати між двовимірним та тривимірним відображенням польоту. Враховуючи, що *2D*-графіку підтримує більшість систем, вона потребує менше ресурсів та її легше реалізувати, було прийнято рішення саме на користь двовимірного відображення.

Тож, якщо з основною ціллю з’ясовано, слід сформулювати проміжні цілі, поступове виконання яких зробить зрозумілим плани на розробку:

– Графічний простий інтерфейс користувача для запуску модуля
– Створення двовимірної системи координат (*x*, *y*)
– Зчитування даних для формування траєкторії
– Скеровування графічного об’єкту БПЛА по вказаній траєкторії
– Відображення поверхні, над якою умовно переміщується БПЛА

2.1.1. Вимоги до модуля

Модуль повинен гарантовано запускатися на ОС *Windows* (7, 8, 10). Керування повинно бути надано користувачеві лише під час запуска модуля, під час польоту БПЛА по траєкторії керування здійснюється автоматично і не потребує втручання, тому під час процесу демонстрації напрямок та швидкість польоту користувач задавати не повинен. Однак, якщо виникне нагальна потреба

призупинити або зупинити демонстрацію, у користувача має бути така можливість незалежно від того досяг дрон кінцевої точки польоту чи ні.

Для плавності роботи модуля при візуальному відображенні зображення оновлюється в середньому 60 раз в секунду, таким чином при роботі модуля система не має сильно навантажуватися, щоб частота оновлення кадрів залишалася стабільною. Якщо робота модуля на певному етапі розробки спричиняє нижчу кадрову частоту, то в пріоритет ставиться вирішення проблеми плавності відображення через заміну, видалення або оптимізацію існуючих програмних механізмів графічного відображення або інших.

Задля керування БПЛА має отримувати координати точок, та, можливо, іншу додаткову інформацію. Для того, щоб зробити внесення таких даних зручнішим, треба щоб модуль мав змогу зчитувати їх набори з впорядкованого за структурою текстового файлу.

2.2. Будова модуля

Перш за все потрібно реалізувати виведення графічного інтерфейсу. Для створення функцій, які зможуть відкривати графічне вікно в ОС *Windows* та відмальовувати у ньому графічні об’єкти. Враховуючи, що існують численні мультимедійні бібліотеки, використати їх інструментарій буде слушніше, ніж програмувати візуальне виведення «з нуля», по аналогії з використанням стандартних бібліотек різних мов програмування.

Всього інтерфейс матиме три стани для відображення:

– Головне меню модуля
– Супроводжуючий вивід інформації під час демонстрації керування польотом БПЛА
– Меню паузи демонстрації керування

В такому разі щоб розуміти в якому напрямку слід розробляти інтерфейс, слід створити його можливі макети розташування елементів інтерфейсу (далі MPEI).

При створенні макетів слід прийняти до уваги, що призначення самого модуля не потребує складних та численних елементів інтерфейсу, які можуть заважати правильно сприймати інформацію під час польоту дрона. Достатньо буде створити інтерфейс максимально простим, зрозумілим, та в загальному притримуватися мінімалістичного дизайну.

Головне меню модуля має надавати користувачеві нагоду вибрати з двох варіантів: розпочати демонстрацію або завершити роботу модуля. Крім того, поскільки головне меню це перше, що побачить користувач під час роботи з модулем, буде доречно виводити лаконічну інформацію про нього. Це може бути, наприклад, власна назва модуля, номер версії, інформація про розробників чи дата створення поточної версії модуля керування.

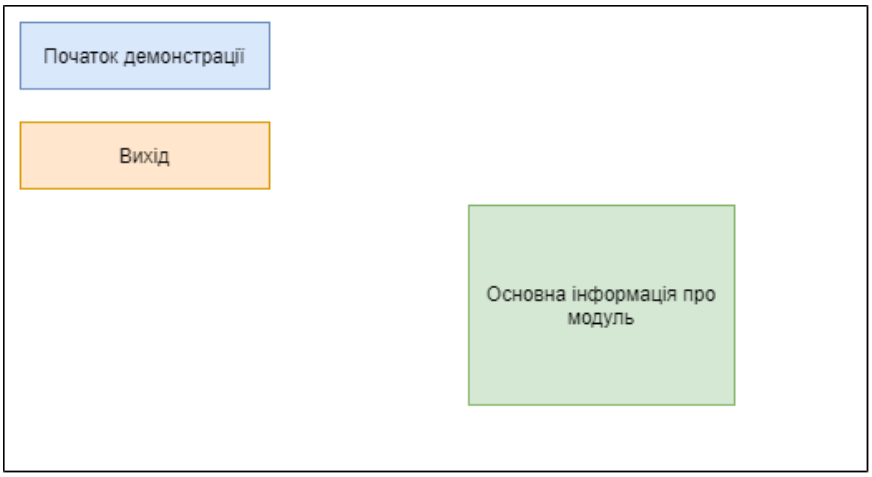

Рис. 2.1. Перший варіант MPEI для головного меню

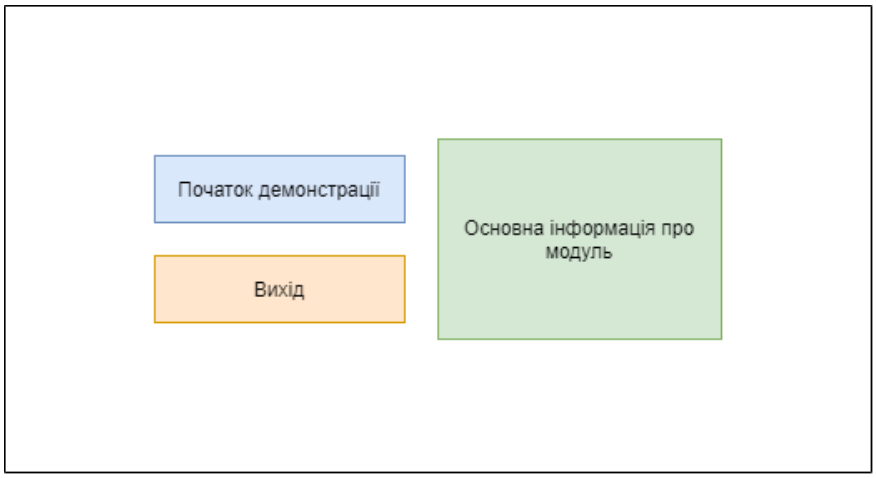

Рис. 2.2. Другий варіант MPEI для головного меню

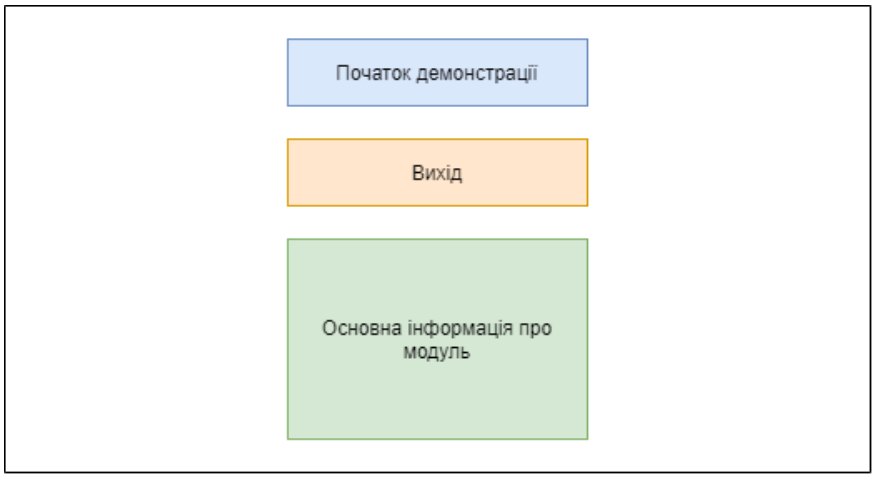

Рис. 2.3. Третій варіант MPEI для головного меню

Зі складених трьох варіантів найдоречнішим є другий, так як наведене розташування влучно використовує наданий простір та виглядає збалансовано.

Щодо MPEI для двох інших станів, вони дуже прості на вигляд: інтерфейс для супроводжуючого виводу інформації під час демонстрації керування польотом БПЛА має займати мінімум місця на екрані. В меню паузи взагалі має бути лише виведення назви меню та візуального пункту меню, що повертатиме модуль до стану демонстрації. Таким чином прийняти вигляд потрібних макетів дуже легко (рис. 2.4, 2.5).

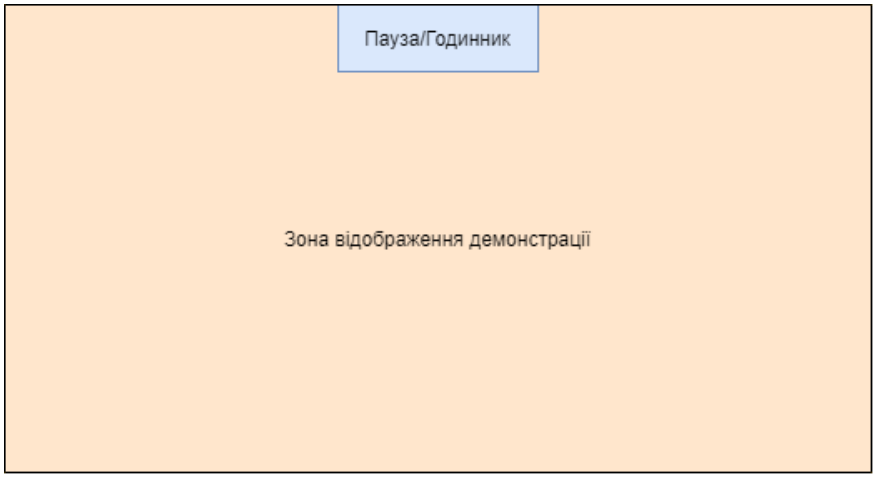

Рис. 2.4 MPEI для стану демонстрації польоту БПЛА

Тож, коли створені потрібні макети для графічного інтерфейсу користувача, при фактичній розробці елементів інтерфейсу та подальшому їх застосуванні у модулі буде зрозуміло, як їх розташовувати.

Рис. 2.5. MPEI для меню паузи

Тепер треба сформулювати модель поведінки модуля з точки зору користувача. Це робиться для того, щоб мати представлення функціонування системи, а точніше відобразити це у зручному для сприйняття вигляду діаграми. При цьому стане в пригоді *UML* (*Unified Modeling Language*) – уніфікована мова проектування. Це зручно не тільки принципово візуальним представленням, що спрощує уявлення про потрібну програмну систему, а й простотою формулювання поведінкової моделі, що полегшить роботу при розробці.

Потрібна діаграма зветься діаграмою прецедентів – тобто фактично діаграма випадків. Вона має в собі зв'язки між випадками використання й іншими системами або ж дійовими особами (рис. 2.6).

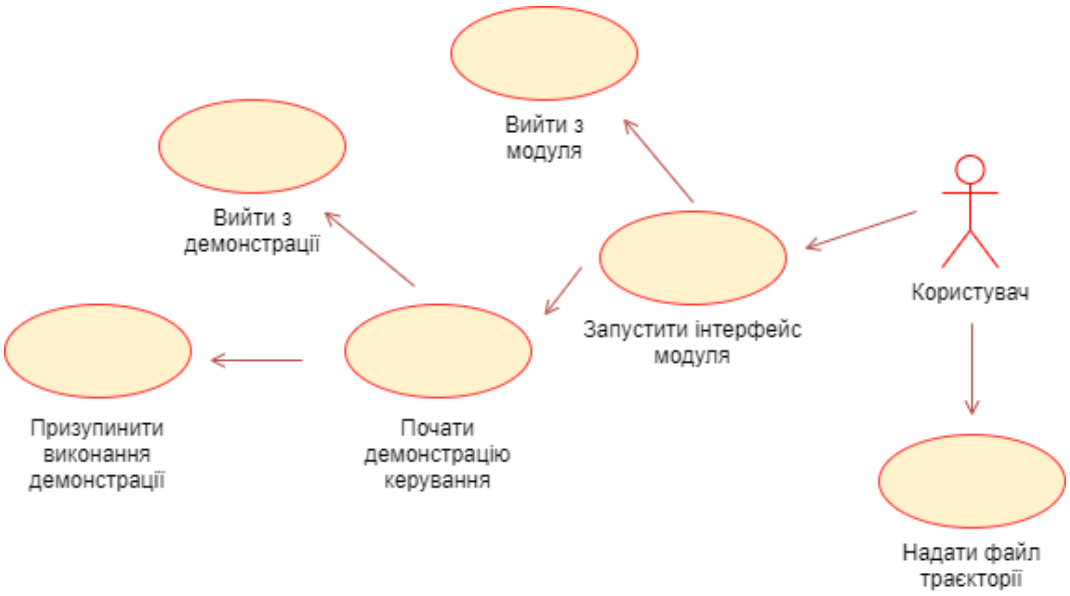

Рис. 2.6. *UML*-діаграма прецедентів модуля

Можна також уточнити щодо станів модуля, для яких створювалися макети інтерфейсу. Модуль матиме три стани: головне меню, демонстрація керування БПЛА та пауза. Стани можуть замінювати або добавлятися один до одного. Наприклад беручи стан головного меню, він додається в момент запуску модуля. При переході в стан демонстрації стан головного меню видаляється, а замість нього здійснюється демонстрація. Таким чином, другий стан замінює перший. Якщо ж під час демонстрації виникне потреба її призупинення, то до існуючого стану додається стан паузи з відповідним меню, але стан демонстрації не видаляється. Тобто активним станом стає меню паузи, а демонстрація тимчасово призупиняється, і коли користувач вийде з меню паузи, не буде потреби ще раз завантажувати стан демонстрації – а це економія ресурсів системи і часу. Іншими словами, коли стан потрібно лише призупинити із збереженням даних, здійснюється додавання, а якщо стан просто треба «перемкнути», здійснюється додавання з попереднім видаленням, тобто заміна.

Діаграма переходу між станами модуля наведена на рис. 2.7.

Рис. 2.7. Діаграма переходів між станами модуля

Далі слід сформувати загальні алгоритми модуля. Перш за все користувач буде бачити саме графічний інтерфейс, тому доцільно буде почати з нього. Крім індикації основної інформації типу годинника переміщення чи назв меню інтерфейс повинен мати інтерактивні елементи, через які користувач зможе взаємодіяти з модулем. Таким елементом буде графічна кнопка, яка згадувалася

ще при формуванні макетів вигляду інтерфейсу. Вона має являти собою графічний об’єкт простої геометричної форми, яка реагує на положення курсору та натискання лівої кнопки миші. При цьому, графічна кнопка має вирізнятися через контрастний по кольору контур або зону натискання, та мати своє позначення у вигляді символу (стрілки, дві паралельні вертикальні лінії – знак паузи, тощо) або короткого напису (рис. 2.8).

Рис. 2.8. Приклади графічних кнопок з написом (зліва) та з символом (справа)

Алгоритм роботи графічних кнопок наведено на рис. 2.9.

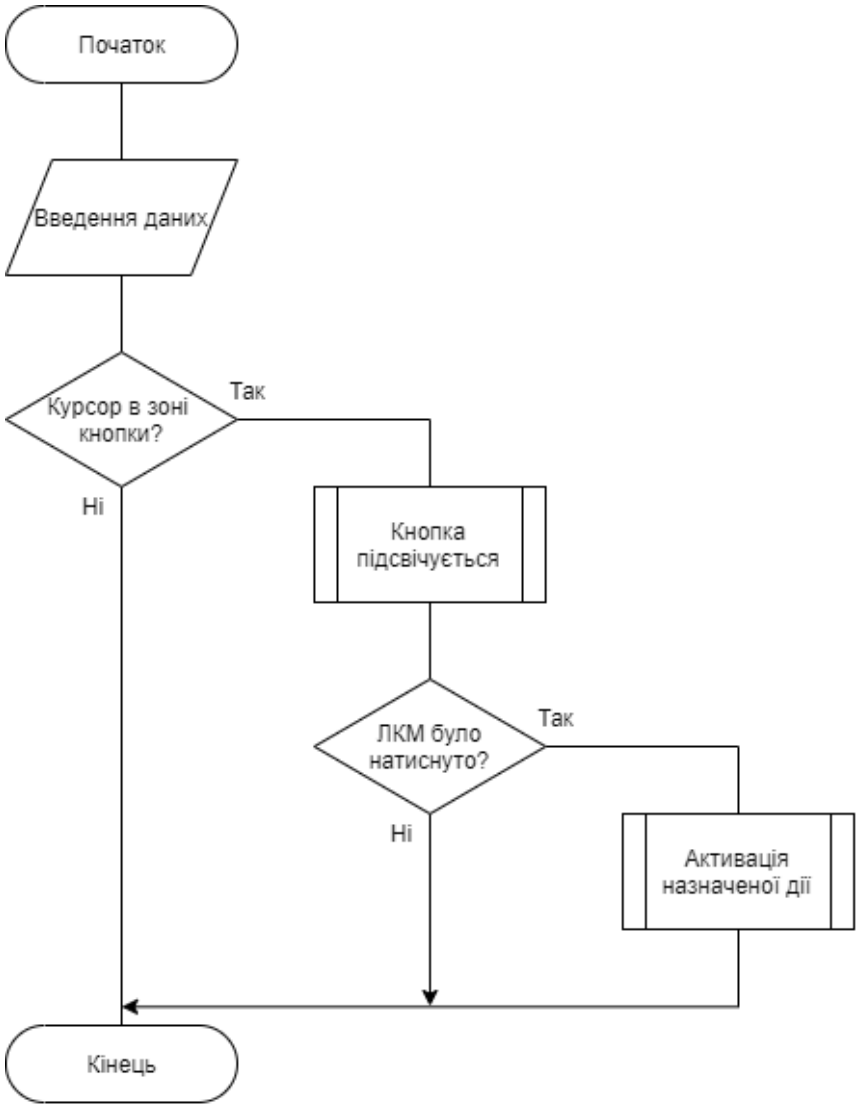

Рис. 2.9. Блок-схема загального алгоритму роботи графічних кнопок модуля

Принцип взаємодії з графічними кнопками полягає у тому, що, по-перше, користувач скористається курсором та наведе його на зону кнопки. Як підтвердження, що кнопка активна, при наведенні вона повинна поміняти своє оформлення – стати іншого кольору, або якось інакше візуально проінформувати користувача про готовність до дії.

По-друге, коли користувач курсором знаходиться в зоні дії графічної кнопки та натискає при цьому ліву кнопку миші має активуватися певна дія, запрограмована після натискання графічної кнопки. Найкраще в такому підході до взаємодії з графічним інтерфейсом це інтуїтивно зрозуміла робота користувача з модулем.

Щодо керування дроном, то цей процес ділиться на два етапи: зчитування даних для траєкторії (рис. 2.10) та безпосереднє керування БПЛА по спланованому курсу (рис. 2.11).

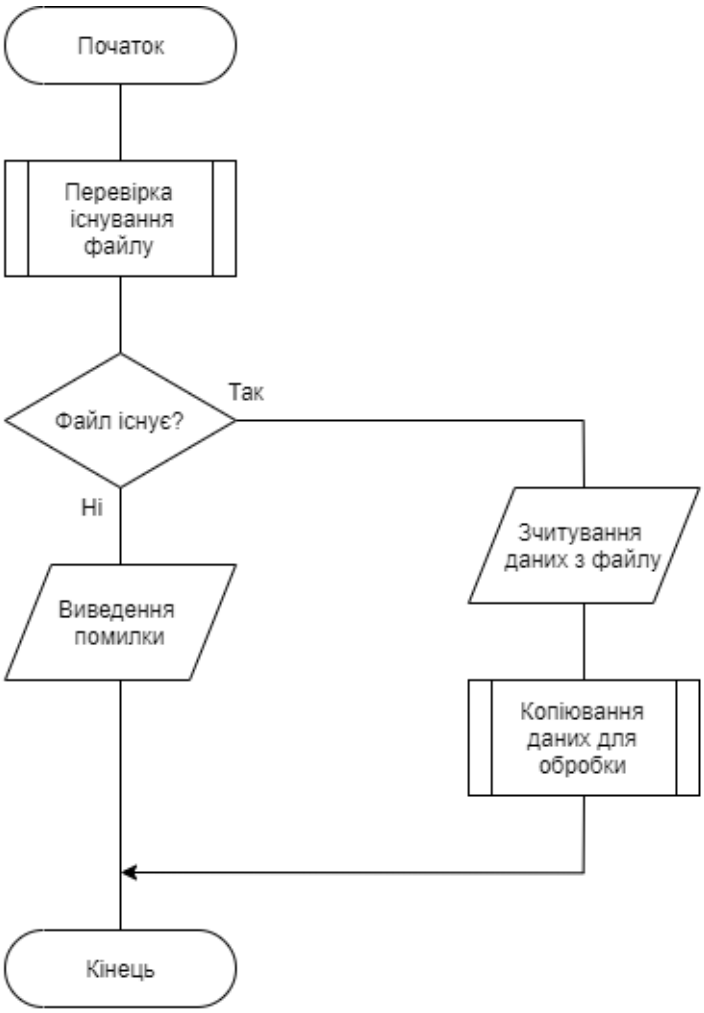

Рис. 2.10. Блок-схема загального алгоритму зчитування даних з файлу

Основна задача при зчитуванні файлу – внести копію даних до пам’яті для подальшої обробки. Крім того, потрібно перевірити, чи потрібний файл взагалі існує. Якщо ні, то й зчитувати щось нема потреби але потрібно вивести інформацію про відсутність файлу.

Для керування безпілотником потрібно задати коректну траєкторію польоту, а точніше набір даних, які описують стан БПЛА у певних точках. Попередньо зчитана інформація, що якраз і описує такі точки, стає джерелом для надання безпілотнику інформації про подальше переміщення. Крім траєкторії польоту по точками, які мають двовимірні координати (x та y) має надаватися інформація про швидкість або час переміщення по вузловим точкам.

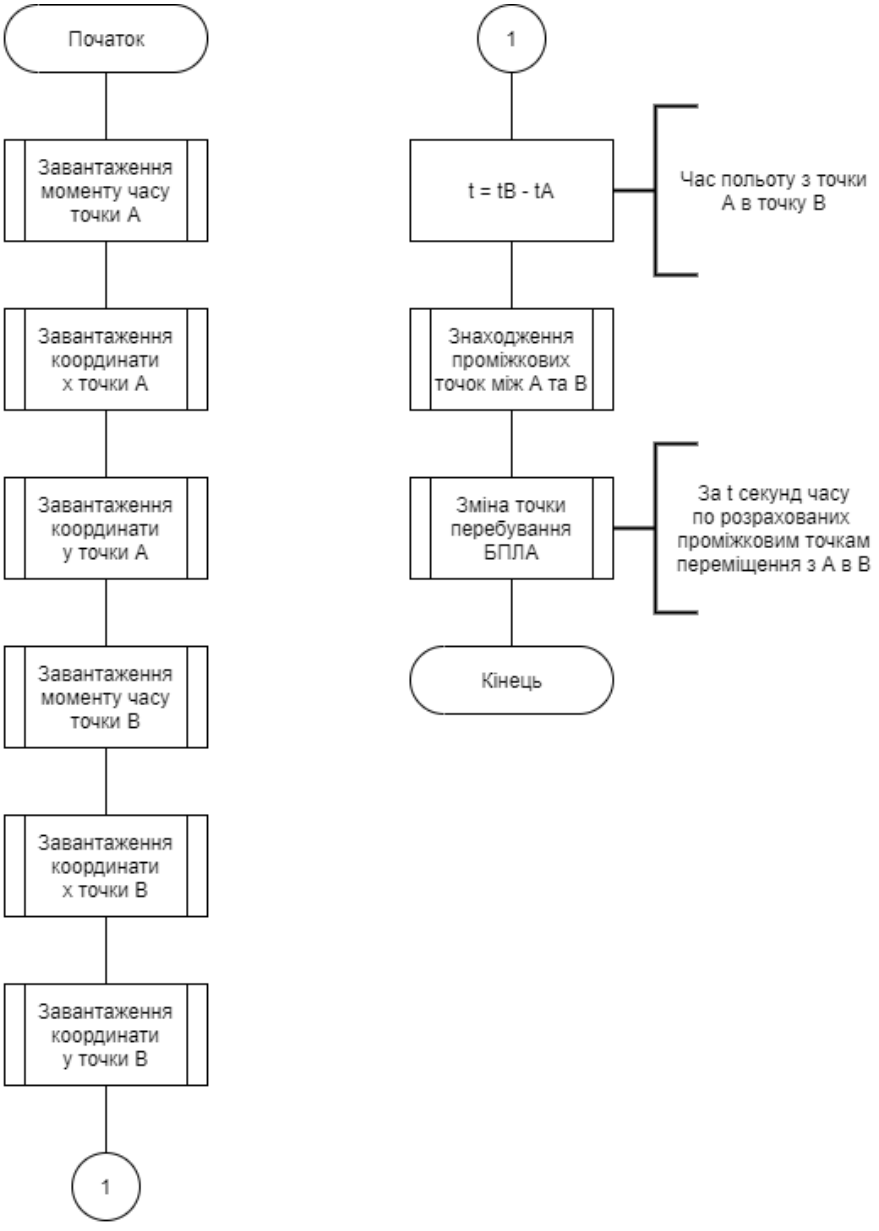

Рис. 2.11. Блок-схема загального алгоритму керування БПЛА

Також треба розуміти, що траєкторія, яка надається для безпілотника може мати різну щільність точок. Такі точки, що надаються безпосередньо як частина даних для переміщення безпілотника і є вузловими. Це означає, що дрон сам переміщується по вузловим точкам, тобто проміжкові точки між послідовими вузловими мають розраховуватися по ходу роботи модуля.

Крім того, слід подумати про плавність роботи модуля. Вже багато інформації було наведено про графічну частину модуля, а це означає що на обробку зображення в реальному часі потребуються додаткові ресурси системи. Якщо зображення декількох елементів інтерфейсу та самого віртуального БПЛА майже не впливають на швидкість роботи модуля, то от відображення карти, над якою літає безпілотник, може заважати користувачеві спостерігати плавну зміну кадрів при демонстрації керування.

Суть карти полягає в тому, що це набір квадратних зображень, які в певній послідовності завантажуються як поверхня, над якою існує графічний об’єкт БПЛА. Зображення формують атлас - набір зображень в певній послідовності, які зберігаються в одному файлі. Крім очевидних переваг в завантаженні при такому способі зберігання, це ще й дає можливість роботи не з усією картою одразу, а лише з деякими її частинами. Враховуючи, що користувач матиме обмежене поле зору при спостеріганні польоту БПЛА, тобто бачитиме при цьому певну множину квадратів карти, це можна використати для зменшення навантаження на систему.

Працювати це має таким чином: спочатку відстежуються поточні координати дрону. На основі цих координат визначається центр області провантаження карти. З врахуванням поля зору спостереження користувача на екрані з’являється обмежена кількість квадратів карти, тобто формуються деяка область видимих фрагментів карти біля самого безпілотника. При цьому, модулю не потрібно витрачати ресурси на інші квадрати карти, які не входять до області провантаження, тим самим не навантажуючи систему.

Загальний алгоритм такого фрагментарного показу карти приведено на рисунку 2.12.

Рис. 2.12 Блок-схема загального алгоритму фрагментарного показу карти

Для того щоб краще сформулювати уявлення про принцип поступового відображення карти, це можна зобразити схематично (рис. 2.13). Якщо умовно вся карта займатиме 13 квадратів по горизонталі та 7 по вертикалі (13x7) то формуватиме прямокутник з 91 квадрату. БПЛА переміщуватиметься над цими квадратами під час польоту, а за цим переміщенням спостерігатиме користувач. Поле зору користувача зображене як напівпрозора помаранчева зона, в центрі якої знаходиться умовний безпілотник (блакитний круг). Як можна побачити, в зв’язку з тим, що поле зору користувача обмежене воно не покриває всю карту, а лише певну область, яка пов’язана з координатами де знаходиться безпілотник. Ті квадрати карти, які «зачіпає» поле зору, зображені зеленим кольором, а інші – сірим. Таким чином модуль візуалізує лише зелені – активні квадрати, які бачить користувач. Сірі ж квадрати вважаються неактивними, і модуль на екран їх не

виводить. Тобто виходить, замість 91 квадрату в певний момент часу для відображення на екрані застосовуються лише 35 квадратів (зона розміром 7x5), а це майже втричі менше площі для рендеру.

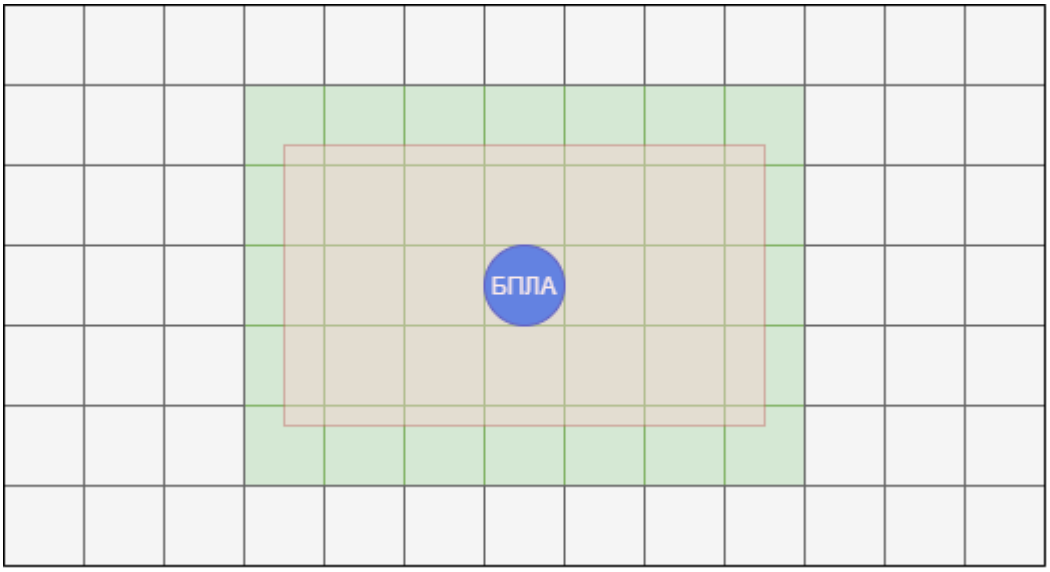

Рис. 2.13. Схема часткового провантаження карти

Також, поскільки область провантаження пов’язана з координатами БПЛА, це дозволяє в реальному часі переміщати її в потрібні місця. Якщо враховувати, що поле зору користувача не впливає на розміри повної карти, то можна дійти думки, що більші розміри карти не будуть впливати на роботу модуля через фіксовану зону провантаження квадратних фрагментів карти.

2.3. Висновки до розділу

В даному розділі було сформовано цілі модуля:

– Модуль має скеровувати БПЛА по траєкторії, що обходить перешкоди
– Графічний простий інтерфейс користувача для запуску модуля
– Створення двовимірної системи координат (x, y)
– Зчитування даних для формування траєкторії
– Відображення поверхні, над якою умовно переміщується БПЛА

Також вказано, які вимоги має задовольняти модуль:

– Можливість запуску під системами *Microsoft Windows* (7,8,10)

– Правильність та візуальна плавність роботи

– Можливість обробки файлів з розширенням *.txt*

Крім того, описувалася будова модуля, де були створені макети розташування елементів інтерфейсу, описані три стани модуля, їх інтерфейсів та принцип зміни самих станів:

– Головне меню

– Демонстрація

– Меню паузи

Наведено *UML*-діаграму прецедентів модуля для розуміння його поведінки з точки зору користувача та приведено блок-схеми алгоритмів, які потрібні модулю для роботи з користувачем (графічні кнопки), завантаження файлу траєкторії, керуванням БПЛА та візуальним відображенням фрагментів поверхні, над якою здійснюватиме польоти безпілотник.

Алгоритми були також описані, а принцип фрагментарного відображення поверхні був порівняний з її повною візуалізацією.

РОЗДІЛ 3
РОЗРОБКА МОДУЛЯ

3.1. Вибір мови програмування та додаткових бібліотек

Для створення модуля було обрано мову програмування *C++* через її універсальність та досвід роботи з нею під час поза навчанням. На користь такого вибору також зіграли роль швидкість виконання коду та актуальність мови.

Для роботи з графікою вибрано мультимедійну бібліотеку *SFML*, яка має версію для роботи на C++. *SFML* – (*Simple and Fast Multimedia Library*) проста та швидка мультимедійна бібліотека. Вона вирізняється своєю простотою використання, гарною документованістю, та, що головне, легкістю освоєння. Це означає, що при написанні модуля можна буде зосередитись на реалізації та покращенні алгоритмів, а не довго освоювати бібліотечні функції. Бібліотека безкоштовна для використання та має відкритий вихідний код, кросплатформна (*Windows, MacOS X, Linux,* планується випуск під *Android* та *iOS*), має версії для розрядностей систем *x*64 й *x*86. Дана бібліотека спрямована на розробку мультимедійних програм з *2D*-графікою. Для відображення візуалу використовується спрайтова анімація, реалізація карт за допомогою кліткового заповнення простору на екрані, використовується прямокутна система координат з двома осями.

Складається сама бібліотека з декількох основних модулей:

– *System* (управління програмним часом, робота з потоками)
– *Window* (управління графічними вікнами та взаємодія з користувачем)
– *Graphics* (відображення графічних примітивів та зображень)
– *Audio* (управління звуком)

– *Network* (налаштування мережевого з’єднання)

В дипломному проекті використовуються модулі бібліотеки *System*, *Window* та *Graphics*.

3.2. Стани модуля

Модуль керування польотом БПЛА містить три стани:

– Головне меню

– Демонстрація

– Меню паузи

Кожен стан створюється як об’єкт, описаний в класі *State*, де наявні віртуальні функції *init(), handleInput(), update(), render()* та *pause()* й *resume()*.

Функція *init()* відповідає за ініціалізацію змінних та попереднє завантаження ресурсів сцени (спрайти, шрифти, тайлмапи тощо). Функція *init()* виконується один раз на початку створення нового об’єкта стану, тобто якщо стан призупинено, а потім відновлено, *init()* виконуватися не буде.

Під час роботи цієї функції можна привести приклад функцій бібліотеки *SFML* для роботи з текстом: для цього потрібно через клас *sf::Text* створити об’єкт текстового виводу, наприклад під назвою *_text*. Далі через бібліотечний метод *setFont()* обирається файл шрифту, яким буде оформлено текст. Щоб задати сам напис, використовується метод *setString()*, для регулювання розміру символів метод *setCharacterSize()*, для вибору кольору тексту метод *setFillColor()*, аргументом можна використовувати клас *SFML sf::Color*. Для розміщення самого тексту використовується метод *setPosition()*, аргументами якого є координати по горизонталі та вертикалі в пікселях. Слід також зазначити, що метод *setPosition()* чи не найчастіше використовується в коді для формування зображення, так як відповідає за розташування всіх графічних об’єктів на екрані.

Для ініціалізації часових лічильників використовується клас *sf::Clock*, який може надавати інформацію про час у секундах, мілісекундах або мікросекундах. Такі значення потрібні для внутрішніх таймерів, годинників, секундомірів чи

загальної синхронізації програмного часу з фактичним. Часовий лічильник можна перезапустити через метод *restart()*, це буде корисно для взяття не просто значення часу, а для певних його інтервалів, щоб виключити можливість переповнення часового лічильника.

Зрозуміло, що при початковій ініціалізації в певному стані модуля можна використовувати й інші функції, які направлені на задання стартових позицій чи значень. Функція *spwnCrdCen()* формує двовимірну систему координат з чотирма четвертями, за якою буде відбуватися управління БПЛА. Варто зазначити, що за замовчуванням *SFML* створює свою систему координат, де вісі виходять з верхнього лівого кута екрану або вікна, при тому вертикальна вісь що йде донизу має додатні значення координат. Крім того, як вже було зазначено координати в *SFML* за замовчуванням вимірюються в пікселях, а для даного проекту мірою довжини польотів дрону прийнято саме метр. Через це всі обраховування, зв'язані з координатами та що в подальшому будуть виведені користувачу чи зчитані з файлу траєкторії проходять перерахунок з пікселів в умовні метри та навпаки.

Ще, крім стандартних типів мови *C++* в *init()* також проходить ініціалізація спрайтів, тобто зображень простих графічних об'єктів. Для спрайтів (бібліотечний клас *sf::Sprite*) потрібно завантажити спеціальне зображення, яке перед цим дано об'єкту, що описується бібліотечним класом *sf::Texture*. Тобто практично файл зображення спочатку надається (метод *getTexture()*) об'єкту класу *sf::Texture*, а новостворена таким чином текстура використовується для надання спрайтові методом *setTexture()*. Для спрайтових об'єктів доступні метод *scale()* – розміри по x та y, та по аналогії з текстом *SFML* методи *setPosition()*, *setColor()*, та деякі інші.

Ще при роботі функції *init()* дрон розміщується на вказаних координатах (в умовних метрах на відповідній системі координат) через *spwnDrnOnMap()*, ініціалізується поверхня, над якою буде здійснювати польоти БПЛА - *initMap()*, та деякі елементи користувацького інтерфейсу, як верхня графічна кнопка паузи - *button_2()*.

Далі про віртуальну функцію *handleInput()*. Ця функція містить в собі програмні засоби, що мають обробляти ручне управління під час стану, таке як,

наприклад, натискання фізичних кнопок (на зовнішніх маніпуляторах, як клавіатура та миша), чи графічних кнопок (безпосередньо взаємодія з графічним інтерфейсом через графічні кнопки). Наприклад, в стані демонстрації оброблюються як фізичні кнопки через подію *sf::Event::KeyPressed* (подія натискання фізичної кнопки) та клас *sf::Keyboard*, який дає доступ до визначення коду кнопки, яка була натиснута, наприклад *sf::Keyboard::Escape* відповідає за взаємодію з кнопкою *ESC* на клавіатурі. Також використовуються власні функції взаємодії з графічним кнопками, наприклад *butPau.b_2_hi()*, яка відповідає за роботу графічної кнопки паузи при демонстрації.

Функція станів *update()* має за мету оновлення основної логіки модуля в певному стані. Наприклад, можна було б використати елементи коду для ручного управління безпілотником, якби це було потрібно при демонстрації, так можливість управління мала б підтримуватись протягом всієї роботи стану. Саме у *update()* відбувається процес керування безпілотником (функції *floatFromVector(), moveToPoint()*) та робота часових лічильників (за потребою).

За відображення зображення користувачеві відповідає функція *render()*. По суті, саме тут зібрано інші функції які відповідають за простий або поступовий вивід графічних об’єктів. Для оновлення кадрів використовується бібліотечна функція *clear()*, яка працює безпосередньо з об’єктом графічного вікна модуля. Наприклад інструкція *window.clear(sf::Color::White)*, де window – об’єкт графічного вікна, видаляє весь візуал та заміщує його білим кольором. Таке «прибирання» потрібно після відмальовки кожного кадру, так як без цього зображення різних кадрів можуть накладатися один на одного, що спричинить проблеми сприйняття візуальної роботи модуля користувачем. Метод *SFML draw()* відповідає за створення кінцевих графічних об’єктів, що готові для виведення на екран. Метод *display()* якраз і здійснює виведення підготовлених графічних об’єктів на екран. Знову ж таки, в *render()* можуть використовуватись не лише вказані методи, а й функції користувача, які спрямовані безпосередньо на виведення зображень на екран.

Крім перечислених чотирьох функцій станів ще наявні *pause()* та *resume()*, але вони використовується лише при зміні станів модуля, відповідно для того щоб призупинити або відновити активність того чи іншого стану.

Для пояснення принципу роботи станів за основу був взятий головний стан модуля – стан демонстрації (фактичний показ користувачеві процесу керування БПЛА). Два інших стани, а саме головне меню та меню паузи мають порівняно простішу будову, так як їх призначення зводиться до простого інформування користувача про поточне меню та надання можливості простої взаємодії з інтерфейсом через графічні кнопки. Таким чином, ці два стани працюють аналогічним чином, як стан демонстрації за винятком лише того, що ні головне меню модуля, ні меню паузи не використовує функцію для оновлення основної логіки *update()*, так як фактично в двох згаданих станах меню інтерактивними є лише графічні кнопки, для логіка яких поміщається в функцію ручного управлвління стану *handleInput().*

3.3. Графічні кнопки

Як вже було описано, основним інтерактивним елементом користувацького інтерфейсу є графічна кнопка. Для її створення застосовується окремий клас Button, в якому описані методи, відповідальні за:

- Ініціалізацію кнопки (*button_1()* – з безпосереднім передаванням спрайта, *button_2()* – пердаванням спрайта через структуру)
- Контроль при ручному управлнні кнопки (*b_1_hi()* та *b_2_hi()*)
- Візуальне відображення кнопки (*b_1_render()* та *b_2_render()*)

Ініціалізація застосовується у функціях станів *init()*. Для кнопки передаються такі аргументи, як координати її розташування по горизонталі та вертикалі, напис, який має бути на кнопці, шрифт, яким цей напис має бути здійснено, текстуру, яка буде являти з себе фон та форму кнопки, розмір рядка для переносу й деякі інші. Функція *button_1()* та *button_2()* завантажує ресурси для свого об’єкту класу Button, а різниця між двома цими функціями в тому, що

друга може працювати з структорою, в яку записано дані для кнопки. Надалі нумерація в назвах функцій графічних кнопок означатимуть ту ж різницю.

Методи *b_1_hi()* та *b_2_hi()* частково відповідають за взаємодію графічних кнопок з користувачем. Застосовуєть бібліотечний метод *sf::IntRect()* який відповідає за формування зони взаємодії на екрані користувача, для цього й потрібно пердавати координати знаходження кнопки. Функції повертають цілочисельне значення, що має надалі визначития яка з кнопок буда в зоні курсора, або для іншого способу обробки. Безпосередня обробка натискання кнопок миші у визначеній зоні програмно прописується в самому стані у функції *handleInput().*

Методи *b_1_render()* та *b_2_render()* забезпечують кінцеву підготовку кнопок інтерфейсу до візуального виведення на екран. Застосовується бібліотечна функція *setColor()* для зміни кольору фону кнопки (реакція на наведення курсору), та *draw()* для підготовки до виводу фону кнопки та тексту, який буде на ній зображено.

3.4. Завантажувач файлу траєкторії

Керування БПЛА основане на даних, що надаються модулю у вигляді текстовго файлу (розширення *.txt*) який має розташовуватися в директорії для зчитування та мати фіксоване визначене ім’я. Такий файл за перелік певних вимог до нього та з певним порядком подання даних у ньому можна назвати протоколом (рис. 3.1).

Протокол зчитується в пам’ять системи за допомогою методу *fullTxtLoader()* класу *Data_op*, цей клас створено для розміщення допоміжних функцій для всього модуля. Метод *fullTxtLoader()* посимвольно зчитує протокол та копіює його до вектору символів (вектор – це структура даних, яка являє собою динамічний масив, який сам може керувати виділеною для нього памяттю, на відміну від простих динамічних масивів, де потрібно вручну виділяти пам’ять через оператори new та delete).

Перед зчитуванням метод також перевіряє наявність файлу у вказаній директорії, і якщо файлу нема – інформує про це через текстове повідомлення.

0.00	108.00	0.00
1.00	107.95	1.08
2.00	107.82	2.16
3.00	107.59	3.23
4.00	107.28	4.29
5.00	106.89	5.35
6.00	106.41	6.39
7.00	105.86	7.42
8.00	105.24	8.44
9.00	104.55	9.43
10.00	103.80	10.41
11.00	103.00	11.38
12.00	102.16	12.32
13.00	101.28	13.24
14.00	100.37	14.14
15.00	99.44	15.03
16.00	98.49	15.89
17.00	97.54	16.74
18.00	96.60	17.58
19.00	95.66	18.40
20.00	94.74	19.21
21.00	93.85	20.00
22.00	93.00	20.80
23.00	92.18	21.58
24.00	91.40	22.37
25.00	90.68	23.15
26.00	90.01	23.95
27.00	89.41	24.74
28.00	88.86	25.55
29.00	88.38	26.37

Рис. 3.1. Фрагмент даних протоколу

Будова протоколу являє собою три колонки даних, що мають фіксовану точність дробової частини.

Перша колонка – момент часу опису положення БПЛА в секундах, починаючи від старту польоту.

Друга колонка – положення БПЛА в двовимірній системі координат по горизонталі, вимірюється в метрах.

Третя колонка - положення БПЛА в двовимірній системі координат по вертикалі, вимірюється теж в метрах.

Також, в зв’язку зі специфічним розташуванням даних для траєкторії, кожен рядок у протоколі описує БПЛА у певний момент часу зі старту польоту. Для підрахунку кількості таких описів використовується функція *getPosN()*. Це робиться для того, щоб контролювати кількість пройдених вузлових точок траєкторії до кінця польоту.

3.5. Реалізація керування БПЛА

Основуючись на згаданому файлі траєкторії здійснюється керування графічим представленням БПЛА під час демонстрації. Для того, щоб безпілотник почав рухатися по траєкторії в потрібний момент, початок автоматичного керування назначено на одну з кнопок клавіатури. Сам факт натиску на кнопку та оновлення деяких значень початкових оброблюється в функції *handleInput()* стану демонстрації, а інша логіка польоту – в функції стану *update()*.

Факт автоматичного руху надає булева зміна mv, яка при значенні true дозволяє працювати дрону автоматично, а при значенні false в логіці польоту функції, відповідальні за автоматичне переміщення БПЛА не активні. Якщо дана команда, що автоматичний політ по траєкторії має здійснюватись (*mv==true*), тоді першим ділом модуль звертається до копії протоколу через функцію *floatFromVector()*, таким чином одним із аргументів вказуючи, які саме дані йому потрібні.

Це працює таким чином: для знаходження через цю функцію потрібного значення знадобиться знання будови протоколу. Як вже згадувалося, дані для траєкторії в файлі розділені на три колонки, а саме момент з початку польоту (секунди), координати *x* (метри) та *y* (метри) положення дрону в цей момент. Для переміщення по такому набору даних потрібно знати, по-перше, позицію дрону, а

це рівносильно номеру рядка в протоколі, так як кожен перелік для свого моменту часу (момент часу, *x*, *y*) має вигляд рядка з відступами. І переміщення по таким «наборам» даних здійснюється за допомогою номера позиції (цілочисельний параметр *targ_pos*).

Але якщо навіть знати номер потрібного рядка (потрібний стан дрону), то треба ще вибрати один із трьох параметрів: момент часу, x чи y цього стану. Для цього стає в пригоді параметр функції *floatFromVector()*, а саме цілочисельний *targ_col*, тобто номер колонки. А так як будова протоколу фіксована, то колонка під номером 1 буде відповідати за момент часу, під номером 2 – за координату *x*, а під номером 3 – за координату *y*. Таким чином за номером позиції та номером колонки можна взяти необхідні дані з протоколу.

Також, потрібно враховувати, що копія протоколу, з якою веде роботу функція *floatFromVector()* це набір символів. А символьний тип даних не годиться для подальших розрахунків, тому перед поверненням результату, в функції *floatFromVector()* здійснюється перетворення символьного типу в потрібний тип дробового числа. Слід також зазначити, що для формування потрібного значення здійснюється спочатку перетворення з набору символів у символьний рядок (string) шляхом поступової конкатенації (злиття рядків). Коли значення сформоване у вигляді символьного рядка, вже саме воно перетворюється у дробове число через стандартну функцію *stof()*.

Тепер, коли *floatFromVector()* забезпечує модуль підготовленими даними, далі вони обраховуються для здійснення руху БПЛА. Функція *moveToPoint()* відповідає за переміщення графічного об’єкту БПЛА по вузловим точкам (тобто по тим, координати яких надані в протоколі). Крім різних лічильників та булевих змінних, основне, що приймає функція – це час, за який треба перемістити з першої точки в другу, поточні координати дрону та координати наступної точки для переміщення. Тому якщо початкові та кінцеві координати для двох вузлових точок можна відразу зчитати з протоколу, то щоб визначити час польоту між цими точками треба від моменту прибуття в точку B (*tB*) відняти момент часу

знаходження в точці А (*tA*). Таким чином різниця моментів й буде представляти потрібний для перельоту час між двома вузловими точками (*t*).

Важливо звернути увагу, що автоматичний політ БПЛА здійснюється етапами, тобто функція *moveToPoint()* викликається циклічно в ті моменти, коли безпілотник досягає вузлових точок, і це працює по принципу конвеєра: точка В минулого відрізку шляху стає точкою А наступного, за винятком першого та останнього значення координат всього маршруту польоту. Але якби безпілотник переміщався лише по вузловим точкам, без можливості саме перемішуватися між ними, а не відразу з'являтися на них, тоді б плавність польоту та й всієї демонстрації залежала б від «щільності» значення даних, наданих в протоколі. Модуль же розроблений таким чином, що навіть якщо дистанція між точками буде достатньо велика, що користувач побачив би ривки при русі БПЛА в разі малої «щільності» даних протоколу, програмно такі дані будуть доповнюватись, і безпілотник буде плавно переміщуватися від точки до точки. Як це досягається? По суті, здійснюється розрахунок проміжкових значень координат точок, куди має переміщуватися дрон. За це відповідає функція *interpolate()*, яка через інструментарій бібліотеки SFML здатна працювати з векторами руху. Таким чином в даному випадку вектор руху в двовимірному просторі має в собі дві величини: вертикальні та горизонтальні координати. Для роботи функції створюється два вектора руху: вектор А (містить координати першої вузлової точки) та вектор В (містить координати другої вузлової точки). Далі від вектору В віднімається вектор А та їх різниця множиться на лічильник *factor*, що й змінює з кожним циклом координати положення графічного об'єкта. Потім результуючий вектор додається до вектору А, щоб розраховані координати були прив'язані саме до початкової точки. У функції *moveToPoint() interpolate()* надає координати, в яких дрон має з'явитися через метод *SFML setPosition()*, що силами біблотеки розміщує спрайт дрону у визначених координатах. Крім того треба розуміти, що траєкторія польоту дрону це множина точок, як вузлових, які описано у протоколі, так і проміжкових, переміщення по яким модуль керування обчислює сам.

Тоді коли дрон досягає кінцевої точки відрізку, функції *moveToPoint()* надаються нові дані щодо нової точки кінця руху по відрізку траєкторії, а початковою точкою стає кінцева точка минулого відрізку, при цьому лічильник функції *interpolate()* обнуляється для обчислення нової множини точок на наступному відрізку шляху. Так здійснюється до тих пір, поки у протоколі наступна точка буде відсутня, тоді БПЛА припиняє рух по траєкторії.

Також відбувається округлення значення координат до трьох знаків після коми для зручності виводу та спостерігання за інформацією для відладки. Ще при старті автоматичного польоту запускається годинник польоту (рис. 3.2), який крім інформування про час польоту (хвилини та секунди) може наочно показувати, що час проходження по траєкторії відповідає теоретичному.

Рис. 3.2. Вигляд годинника польоту на графічному інтерфейсі модуля

3.6. Реалізація завантаження карти поверхні

Як вже було зазаначено, карта поверхні завантажує не просто картинку для кожного фрагменту, а працює з атласом, де в певному порядку розміщено графічний матеріал фіксованого розміру. Вибіркове зчитування здійснюється програмним способом, заснованим на знанні розмірів того чи іншого зображення на атласі та його розміщення відносно верхнього лівого кута атласа в пікселях. Знання таких даних дає змогу здійснювати виведення не всього атласа, а вибірково певних його зон.

Опрацювання карти здійснюєтьсяв двох функціях стану демонстрації: *init()* та *render()*. Відповідно в першій функції відбувається ініціалізація – завантаження

ресурсів та підготовка змінних для карти, а другій візуалізація, при чому фрагментарна.

Спочатку створюється об’єкт класу *Map*, потім у функції ініціалізації стану демонстрації застосовується метод *initMap()* класу *Map*. Основне що там відбувається – завантаження атласу через бібліотечну функцію *SFML setTexture(),* зміна розміру зображення через *scale()* та створення масиву символьних рядків, по яких будуть обиратися фрагменти для виводу.

Потім у функції візуалізації стану використовується метод *drawMap()*, що відповідає не лише за підготовку, а й за попередній вибір квадратних фрагментів, які потрібно візуалізувати біля графічного об’єкту дрону. Такий вибір напряму залежить від положення БПЛА, бо саме від нього залежить, де показувати фрагмети карти. Суть полягає в тому, що циклічно проходячись масиву символьних рядків (спосіб опису розташування фрагментів карти) та базуючись на координатах перебування БПЛА в поточний момент часу визначаються індекси потрібних символів у переліку (тобто індекс розташування по умовній вертикалі та горизонталі в масиві рядків). Під час руху безпілотника лічильники, відповідальні за зміну діапазону потрібних індексів змінюють значення в межах розміру масиву. Якщо безпілотник по якихось причинах вилетить за межі карти, лічильники все одно мають обмеження щоб не міняти своє значення на те, яке виходить за межі індексів масиву.

Після визначення потрібного переліку індексів, починається процес розташовуання фрагментів за координатами. Ті фрагменти, які не входять до діапазону далі не оброблюються, а потрібний діапазон співставляється зі своїми фрагментами. Справа в тому, що так як інформація про загальне розташування фрагментів на карті внесена до символьного масиву, за кожен фрагмент відповідає символ, який вкаже яке саме зображення потрібно виводити, коли цикл дійде до відповідного символу. Наприклад, коли символом є «s», фрагменту карти присвоюється одне зображення, коли «0» - інше. Символи в переліку не обов’язково мають бути унікальними для кожного фрагменту.

Також треба розуміти, що карта виводиться вся не відразу, навіть та зона, яку бачить користувач на екрані. Візуально справжньої послідовності виведення не видно через високу кадрову частоту візуалізації модуля. Насправді виконується цикл, який зліва-направо малює послідовність квадратних візуальних фрагментів, що часто повторюються – тайлів. Коли кількість пройдених по рядку квадратів дорівнює довжині рядка масиву символів, координати наступного тайлу переводяться під початок верхнього ряду пройдених квадратів. Але завдяки перевірці потрібного діапазону не кожен тайл виводиться на екран, хоча він залишає «пусте» місце на карті, яке не потребує відмальовки і тим самим зменшує навантаження на систему.

Таким чином, кожного разу функція *drawMap()* малює тайл (метод *draw()*) відповідно до символу у потрібному діапазоні знаходження БПЛА, а коли індекси символу не входять до діапазону, залишається «пусте» місце без тайла. Потрібна зона з атласу для тайлу обирається функцією *setTextureRect()*, де треба вказати координати верхнього лівого кута (початок) та нижнього правого кута (кінець) зони для взяття зображення.

3.7. Висновки до розділу

Даний розділ описував процес створення модуля, пояснював роботу важливих функцій та їх вплив на модуль в цілому. За допомогою такої реалізації модуль задовольняє всім раніше поставленим до нього вимогам.

Для роботи модуля велику роль грають класи:

– *Button*

– *Map*

– *Data_op*

Де в класі *Button* описується реалізація основного інтерактивного елементу користувацькому інтерфейсі – графічних кнопок.

Клас *Map* описує реалізацію карти поверхні під час демонстрації, фрагментарне виведення залдя економії ресурсів системи.

Клас *Data_op* створений для надавання користувацьких методів різного призначення (від округлення числа до керування БПЛА). Цей клас в тому числі використовується для завантаження файлу протоколу та передачі графічному об’єкту БПЛА даних про траєкторію польоту.

Під час створення модуля мультимедійна програмна бібліотека *SFML* надала можливість зосередитися саме на логіці роботи модуля, що дозволило зробити розробку швидшою та приємнішою. *SFML* повністю оправдано називається «простою та швидкою», так як на освоєння роботи з нею пішло відносно мало часу, а наявність великого рівня підтримки цієї бібліотеки розробниками та велика кількість інформації про роботу з нею залишили приємний досвід розробки.

ВИСНОВКИ

Під час даного дипломного проекту було створено модуль керування БПЛА, який дозволяє обминати наявні перешкоди на маршруті польоту. Крім здійснення самого процесу керування, модуль має власний інтерактивний графічний інтерфейс та може візуально демонструвати політ БПЛА по заданій траєкторії.

Також була оглянута предметна область, пов’язана з модулем. Були проаналізовані програмні аналоги для контрольваного руху візуальних об’єктів на екрані, вирізнені їх переваги та недоліки, які були взяті до уваги під час проектування та подальшої розробки модуля.

Були сформовані цілі до модуля керування:

– Модуль має скеровувати БПЛА по траєкторії, що обходить перешкоди
– Графічний простий інтерфейс користувача для запуску модуля
– Створення двовимірної системи координат (x, y)
– Зчитування даних для формування траєкторії
– Відображення поверхні, над якою умовно переміщується БПЛА

Та вимоги до нього:

– Можливість запуску під системами *Microsoft Windows* (7,8,10)
– Правильність та візуальна плавність роботи
– Можливість обробки файлів з розширенням *.txt*

Для кращого розуміння вектору розробки були складені макети розташування елементів інтерфейсу, блок-схеми основних задач, які потрібні для роботи та досягнення цілей модуля та складена поведінкова *UML*-діаграма.

Модуль розроблявся на мові програмування *C++* з використанням додаткової мультимедійної бібліотеки *SFML*. Дана бібліотека використовувалася для графічної частини модуля, що дозволило зробити розробку більш ефективною, так як *SFML* надала весь потрібний набір можливостей для візуалізації роботи модуля.

Модуль має такі основні переваги над аналогами:

– Простота користуванням модуля
– Інтуїтивно зрозумілий графічний інтерфейс користувача
– Мінімалістичний дизайн
– Проста й зрозуміла структура протоколів керування
– Плавність руху графічного об’єкта навіть при малій «щільності» точок польоту
– Оптимізований вивід зображення

Коли проводилася робота над створенням дипломного проекта, було здобуто нових корисних навичок у:

– Створенні програмного коду на мові *C++*
– Роботі з мультимедійною бібліотекою *SFML*
– Формуванням алгоритмів у вигляді блок-схем
– Розробці дизайну графічного інтерфейсу користувача
– Розумінні роботи з об’єктами у *2D*-просторі

Також було здобуто унікальний досвід створення дипломного проекту за усіма загальноприйнятими нормами.

Розроблений проект має високий потенціал розвитку у вигляді розширення можливостей модуля, додавання нових елементів інтерфейсу, подільшої оптимізації роботи та переносу на інші ОС.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *SFML: Frequently Asked Questions (FAQ)* [Електронний ресурс]. – Режим доступу: *https://www.sfml-dev.org/faq.php* (дата звернення 19.05.2021). – Назва з екрану.

2. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

3. ДСТУ ГОСТ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення».

4. Огляд сфер використання БПЛА в повсякденному житті [Електронний ресурс]. – 2016. – Режим доступу: *http://www.50northspatial.org/ua/uavs-everyday-life/* (дата звернення 19.05.2021). – Назва з екрану.

5. Галузі майбутнього: як безпілотники підкорюють Україну [Електронний ресурс]. – 2018. – Режим доступу: *https://mind.ua/publications/20187343-galuzi-majbutnogo-yak-bezpilotniki-pidkoryuyut-ukrayinu* (дата звернення 19.05.2021). – Назва з екрану.

6. Кращі рушії для створення власних 2D інді-ігор [Електронний ресурс]. – 2017. – Режим доступу: *https://genapilot.ru/best-2d-game-engines* (дата звернення 20.05.2021). – Назва з екрану.

7. Уроки по графічній бібліотеці *SFML* [Електронний ресурс]. – Режим доступу: *https://ravesli.com/uroki-po-sfml/* (дата звернення 20.05.2021). – Назва з екрану.

8. Векторна алгебра в *SFML* [Електронний ресурс]. – Режим доступу: *https://ps-group.github.io/cxx/sfml_vector_math* (дата звернення 21.05.2021). – Назва з екрану.

9. Чому C++ крутий, актуальний та безсмертний [Електронний ресурс]. – 2018. – Режим доступу: *https://vc.ru/hr/50161-pochemu-c-krut-aktualen-i-bessmerten* (дата звернення 21.05.2021). – Назва з екрану.

10. *Comparing Game Engines: Unity vs Unreal vs the Rest* [Електронний ресурс]. – 2018. – Режим доступу: *https://www.pubnub.com/blog/comparing-game-engines-unity-unreal-corona-gamemaker/* (дата звернення 25.05.2021). – Назва з екрану.