СИЛАБУС НАВЧАЛЬНОЇ ДИСЦИПЛІНИ

1. Загальна інформація про навчальну дисципліну

Повна назва навчальної дисципліни	Клієнт-серверні технології в ігровій індустрії
Повна офіційна назва закладу вищої освіти	Сумський державний університет
Повна назва структурного підрозділу	Навчально-науковий інститут бізнесу, економіки та менеджменту. Кафедра економічної кібернетики
Розробник(и)	Гриценко Костянтин Григорович
Рівень вищої освіти	Перший рівень вищої освіти, НРК – 6 рівень, QF-LLL – 6 рівень, FQ-EHEA – перший цикл
Тривалість вивчення навчальної дисципліни	один семестр
Обсяг навчальної дисципліни	Обсяг становить 5 кред. ЄКТС, 150 год. Для денної форми навчання 72 год. становить контактна робота з викладачем (32 год. лекцій, 40 год. практичних занять), 78 год. становить самостійна робота.
Мова викладання	Українська

2. Місце навчальної дисципліни в освітній програмі

Статус дисципліни	Обов`язкова навчальна дисципліна для освітньої програми "Кіберспорт та розробка комп`ютерних ігор"
Передумови для вивчення дисципліни	Передумови для вивчення відсутні
Додаткові умови	Додаткові умови відсутні
Обмеження	Обмеження відсутні

3. Мета навчальної дисципліни

Метою навчальної дисципліни є формування у студентів професійних компетенцій щодо застосування клієнт-серверних технологій в ігровій індустрії

4. Зміст навчальної дисципліни

Тема 1 Базові концепції клієнт-серверної взаємодії

Модель клієнт-сервер. Клієнт-серверна топологія. Авторитетний сервер. Такти сервера та оновлення стану гри. Затримка мережі. Методи синхронізації мережі. Етапи розробки клієнт-серверних застосунків. Порівняння ігрових рушіїв. Порівняння мов програмування. Мережні пакети. Мережні сервіси зі встановленням та без встановлення з'єднання. Маршрутизатори. IP-адресація. Таблиця маршрутизації. Шлюз по замовчуванню. Доменна система імен (DNS). Сокети та порти. Модель мережної взаємодії OSI. Мережні утиліти командного рядка.

Тема 2 Серіалізація та десеріалізація

Поняття серіалізації та десеріалізації. Основні принципи серіалізації. Дані гравця. Телеметрія. Мережна взаємодія. Проблеми та ризики. Вразливості серіалізації. Втрата даних. Порушення конфіденційності. Непередбачувана поведінка. Основні класи Unity для роботи з форматом JSON. Основні методи класу JsonUtility. Переваги класу JsonConvert перед класом JsonUtility. Бінарна серіалізація та десеріалізація.

Тема 3 REST API

Мережне програмування. Використання веб-сервісів для отриманняч даних для вашої гри (таблиці лідерів, списки друзів). Операції CRUD (Create, Read, Update, Delete) з даними. Поняття REST API. REST запити. Структура URL-адреси, яка формує запит на потрібні вам дані. Коренева кінцева точка та шлях до ресурсу. Документація API сервісу, що містить шляхи, які він пропонує. HTTP-заголовки. HTTP-запити POST, PUT, PATCH та DELETE. REST відповіді. GET-запит на таблицю лідерів та повернена JSON-відповідь. Автентифікація та обмеження. Відкрита автентифікація OAuth. Токени. Асинхронне надсилання запитів та отримання відповідей. Корутина. Клас UnityWebRequest. Створення об'єкту UnityWebRequest за допомогою рядка URL-адреси. Методи UnityWebRequest.Get(), UnityWebRequest.Post(), UnityWebRequest.Put(). Надсилання запиту на сервер, використовуючи метод SendWebRequest(). Оброблення відповіді сервера. Перевірка властивостей об'єкту UnityWebRequest. Властивість isNetworkError. Властивість isHttpError. Властивість downloadHandler.text. Використання бібліотеки бібліотеки JsonUtility. Створення класів за допомогою JsonUtility.FromJson(). Метод Debug.Log(). Приклад використання корутини для здійснення виклику API за допомогою UnityWebRequest. Виклик корутини з іншого скрипта з використанням методу StartCoroutine(). Отримання HTML-коду веб-сторінки з Інтернет. Отримання інших ресурсів з Інтернет. HTTP-запит POST. Загальний клієнт REST API.

Тема 4 UDP та TCP-з'єднання

UDP-пакети. Переваги використання UDP у застосунках реального часу, особливо у динамічних іграх. Заголовок та корисне навантаження пакету UDP. Поняття інкапсуляції. Метадані заголовків пакетів IP і UDP. Помилки пакетів UDP. Втрата пакетів UDP. Дублювання. Зміна порядку. Пошкодження. Порівняння протоколів UDP та TCP. Методи зменшення ненадійності протоколу UDP. Тристороннє рукостискання TCP. Повідомлення синхронізації (SYN) і підтвердження (ACK). Клієнт-серверні з'єднання TCP. Використання класу TcpListener. Сокетне з'єднання з використанням класу Socket. Встановлення сокетного з'єднання з використанням методу Connect() і блоку try/catch. Приклад підключення до локального сервісу. Способи прийняття вхідного сокетного з'єднання за допомогою TcpListener. Синхронне та асинхронне надсилання та отримання даних. Синхронний метод AcceptSocket(). Асинхронний метод BeginAcceptSocket(). Синхронний метод Send(). Асинхронний метод BeginSend(). Об'єкт стану та метод зворотного виклику (callback). Надсилання великого обсягу даних кількома порціями. Синхронний метод Receive(). Асинхронний метод BeginReceive(). Приклад отримання великого файлу через TCP за допомогою сокета. Приклад об'єкта стану з сокетом та буфером. Використання TcpClient та TcpListener замість сокетів. З'єднання та надсилання даних за допомогою TcpClient. Асинхронні методи BeginWrite() та BeginRead() класу NetworkStream. Метод BeginAcceptTcpClient().

Тема 5 Проблеми мережних з'єднань

Авторитетні сервери. Синхронні та асинхронні ігри. Планування багатокористувацьких ігор. Завантаження даних. Шахрайство. Локальний сервер. Пропускна здатність мережного з'єднання. Симетричні та асиметричні з'єднання. Фактори, які можуть впливати на продуктивність мережі. Затримка в мережі. Фактори, які спричиняють затримку в мережі. Затримка між клієнтом і сервером. Затримка розповсюдження. Затримка доставки. Затримка оброблення. Лінійна затримка. Приклад визначення затримки в мережі за допомогою утиліти командного рядка traceroute. Прогнозування на стороні клієнта. Проблема синхронізації. Узгодження з сервером. Пріоритетне оновлення.

Тема 6 Ігрові сервери

Використання серверів у іграх. Публічні сервери. Віртуальні приватні сервери (VPS). Характеристики виділених серверів. Масштабованість. Безпека. Швидкість. Контроль. Виділені сервери в ігровій індустрії. Безголові сервери в ігровій індустрії. Віддалене адміністрування безголових серверів із використанням SSH(Secure Shell). Сервер прослуховування, розміщений на клієнті. Розподілені повноваження. Локальний багатокористувацький режим. Однорангова топологія (P2P). Переваги однорангових мереж. Відсутність центрального сервера. Відсутність управління чергами. Відсутність простоїв. Відсутність втрати гравців. Переваги багатосерверної архітектури. Гібридна архітектура. Функціональність балансувальника навантаження. Апаратні та програмні балансувальники навантаження. Ігри в локальних комп'ютерних мережах. Налаштування локальної комп'ютерної мережі з кількома пристроями та маршрутизатором. Віртуальні приватні мережі (VPN). Віртуальне тунелювання. Шифрування даних. Використання VPN сервісу Hamachi.

Тема 7 Програмні рішення Unity для розробки багатокористувацьких ігор Переваги Unity для розробки багатокористувацьких ігор. Прототипування. Екосистема. Кросплатформенність. Спільнота та документація. Оптимізація та продуктивність. Інтеграція фізики. Інсталяція бібліотеки Unity Netcode for GameObjects. Додавання до проєкту компоненти NetworkManager. Додавання NetworkObject до GameObject. Ідентифікатори GlobalObjectIdHash, NetworkObjectId, OwnerClientId. Netcode компоненти NetworkObject, NetworkBehaviour, NetworkAnimator, NetworkTransform. Створення NetworkObject гравця. Використання NetworkBehaviour для керування рухом гравця. Властивості повноважень та власності IsClient, IsServer, IsHost, IsLocalPlayer, IsOwner, IsPlayerObject, IsSceneObject. Застосування повноважень клієнта з використанням NetworkTransform. Компоненти режиму авторизації власника. Синхронізація з авторизацією сервера. Мережева синхронізація. Порівняння RPC (Remote Procedure Call) та NetworkVariable. Моделювання затримки мережі. Налаштування Debug Simulator у компоненті UnityTransport. Використання Network Simulator. Антиципація на стороні клієнта із використанням бібліотеки Netcode for GameObjects. Прогнозування на стороні клієнта із використанням бібліотеки Netcode for Entities.
Тема 8 Тестування та налагодження мережевих ігор Локальне тестування. Збірки гравців. Режим багатокористувацької гри. Імітація мережевих умов. Інструмент Network Simulator. Тестування клієнтських підключень. Методи налагодження багатокористувацьких ігор. Налагодження малювання. Рендерер ліній. Текстове ведення журналу. Записи екрану. Збільшення фіксованого часового кроку. Використання точок зупинки. Створення помічника командного рядка для тестування. Сервіси Unity для створення багатокористувацьких ігор. Сервіс Matchmaker для зв'язування гравців з іншими гравцями або ігровими сесіями на основі таких критеріїв, як рівень майстерності чи географічне розташування, для забезпечення збалансованого ігрового досвіду. Визначення критеріїв підбору партнерів. Профілі та рейтинги гравців. Запити на підбір партнерів. Динамічні коригування. Використання бібліотек Lobby (передігрова зона, де гравці можуть зібратися, налаштувати параметри гри та підготуватися до початку гри) та Relay (забезпечення простого та безпечного з'єднання між клієнтами) для створення способу знаходити спільну мову між гравцями та взаємодіяти один з одним в різних багатокористувацьких ігрових сценаріях. Багатокористувацький хостинг. Спілкування гравців у чаті Vivox.

5. Очікувані результати навчання навчальної дисципліни

Після успішного вивчення навчальної дисципліни здобувач вищої освіти зможе:

РН1	Знати базові поняття, терміни та технології мережного програмування
РН2	Розуміти основні принципи розробки клієнт-серверних програмних систем
РН3	Вміти пояснити суть застосування клієнт-серверних технологій в ігровій індустрії
РН4	Застосувати вміння працювати з сучасними засобами розробки комп'ютерних ігор

6. Роль навчальної дисципліни у досягненні програмних результатів

Програмні результати навчання, досягнення яких забезпечує навчальна дисципліна. Для спеціальності 121 Інженерія програмного забезпечення:

ПР1	ПРН6. Уміння вибирати та використовувати відповідну задачі методологію створення програмного забезпечення.
ПР2	ПРН8. Вміти розробляти людино-машинний інтерфейс.
ПР3	ПРН15. Мотивовано обирати мови програмування та технології розробки для розв’язання завдань створення і супроводження програмного забезпечення.
ПР4	ПРН18. Знати та вміти застосовувати інформаційні технології обробки, зберігання та передачі даних.

7. Роль навчальної дисципліни у досягненні програмних компетентностей

Програмні компетентності, формування яких забезпечує навчальна дисципліна:
Для спеціальності 121 Інженерія програмного забезпечення:

ПК1	ЗК02. Здатність застосовувати знання у практичних ситуаціях.
ПК2	ЗК05. Здатність вчитися і оволодівати сучасними знаннями.
ПК3	ФК3. Здатність розробляти архітектури, модулі та компоненти програмних систем.
ПК4	ФК11. Здатність реалізовувати фази та ітерації життєвого циклу програмних систем та інформаційних технологій на основі відповідних моделей і підходів розробки програмного забезпечення.
ПК5	ФК13. Здатність обґрунтовано обирати та освоювати інструментарій з розробки та супроводження програмного забезпечення.

8. Види навчальних занять

Тема 1. Базові концепції клієнт-серверної взаємодії
Лк1 "Особливості розробки клієнт-серверних застосунків у ігровій індустрії" (денна) Модель клієнт-сервер. Клієнт-серверна топологія. Авторитетний сервер. Такти сервера та оновлення стану гри. Затримка мережі. Методи синхронізації мережі. Етапи розробки клієнт-серверних застосунків. Порівняння ігрових рушіїв. Порівняння мов програмування.
Лк2 "Базові концепції клієнт-серверної взаємодії" (денна) Мережні пакети. Мережні сервіси зі встановленням та без встановлення з’єднання. Маршрутизатори. IP-адресація. Таблиця маршрутизації. Шлюз по замовчуванню. Доменна система імен (DNS). Сокети та порти. Модель мережної взаємодії OSI. Мережні утиліти командного рядка.
Тема 2. Серіалізація та десеріалізація
Лк3 "Серіалізація та десеріалізація" (денна) Поняття серіалізації та десеріалізації. Основні принципи серіалізації. Дані гравця. Телеметрія. Мережна взаємодія. Проблеми та ризики. Вразливості серіалізації. Втрата даних. Порушення конфіденційності. Непередбачувана поведінка.

Лк4 "Робота з форматом JSON в Unity" (денна)

Використання формату JSON в розробці ігор. Файли manifest.json і packages-lock.json в Unity. Основні класи Unity для роботи з форматом JSON. Основні методи класу JsonUtility. Переваги класу JsonConvert перед класом JsonUtility. Бінарна серіалізація та десеріалізація.

Пр1 "Серіалізація та десеріалізація даних" (денна)

Створення 2D-проєкту в Unity, що використовуватиме серіалізацію та десеріалізацію. Створення об’єкту BasicObject, що буде серіалізовано та десеріалізовано в JSON та з JSON. Створення скрипта JsonSerializationExample, який буде серіалізувати та десеріалізувати об’єкт BasicObject у JSON та з JSON. Створення бінарної версії JSON-серіалізатора. Створення 2D-проєкту в Unity, що використовуватиме бінарну серіалізацію та десеріалізацію. Оголошення структури MyData з атрибутами StructLayout і MarshalAs. Створення класу BinarySerializationExample. Створення екземпляру структури MyData та заповнення її даними. Створення копії структури даних. Виведення вмісту оригінального екземпляра та копії на консоль під час запуску гри. Створення мережевої бібліотеки NetLib для зберігання допоміжних функцій та класів, які будуть основою для розробки мережевих ігор, щоб уникнути дублювання великої кількості коду. Використання методів розширення ToJsonBinary() та FromJsonBinary() для серіалізації/десеріалізації об’єкта в бінарний формат JSON та з нього.

Пр2 "Серіалізація та десеріалізація даних (продовження)" (денна)

Створення 2D-проєкту в Unity, що використовуватиме серіалізацію та десеріалізацію. Створення об’єкту BasicObject, що буде серіалізовано та десеріалізовано в JSON та з JSON. Створення скрипта JsonSerializationExample, який буде серіалізувати та десеріалізувати об’єкт BasicObject у JSON та з JSON. Створення бінарної версії JSON-серіалізатора. Створення 2D-проєкту в Unity, що використовуватиме бінарну серіалізацію та десеріалізацію. Оголошення структури MyData з атрибутами StructLayout і MarshalAs. Створення класу BinarySerializationExample. Створення екземпляру структури MyData та заповнення її даними. Створення копії структури даних. Виведення вмісту оригінального екземпляра та копії на консоль під час запуску гри. Створення мережевої бібліотеки NetLib для зберігання допоміжних функцій та класів, які будуть основою для розробки мережевих ігор, щоб уникнути дублювання великої кількості коду. Використання методів розширення ToJsonBinary() та FromJsonBinary() для серіалізації/десеріалізації об’єкта в бінарний формат JSON та з нього.

Пр3 "Серіалізація та десеріалізація даних (продовження)" (денна)

Створення 2D-проєкту в Unity, що використовуватиме серіалізацію та десеріалізацію. Створення об’єкту BasicObject, що буде серіалізовано та десеріалізовано в JSON та з JSON. Створення скрипта JsonSerializationExample, який буде серіалізувати та десеріалізувати об’єкт BasicObject у JSON та з JSON. Створення бінарної версії JSON-серіалізатора. Створення 2D-проєкту в Unity, що використовуватиме бінарну серіалізацію та десеріалізацію. Оголошення структури MyData з атрибутами StructLayout і MarshalAs. Створення класу BinarySerializationExample. Створення екземпляру структури MyData та заповнення її даними. Створення копії структури даних. Виведення вмісту оригінального екземпляра та копії на консоль під час запуску гри. Створення мережевої бібліотеки NetLib для зберігання допоміжних функцій та класів, які будуть основою для розробки мережевих ігор, щоб уникнути дублювання великої кількості коду. Використання методів розширення ToJsonBinary() та FromJsonBinary() для серіалізації/десеріалізації об’єкта в бінарний формат JSON та з нього.

Пр4 "Серіалізація та десеріалізація даних (продовження)" (денна)

Створення 2D-проєкту в Unity, що використовуватиме серіалізацію та десеріалізацію. Створення об’єкту BasicObject, що буде серіалізовано та десеріалізовано в JSON та з JSON. Створення скрипта JsonSerializationExample, який буде серіалізувати та десеріалізувати об’єкт BasicObject у JSON та з JSON. Створення бінарної версії JSON-серіалізатора. Створення 2D-проєкту в Unity, що використовуватиме бінарну серіалізацію та десеріалізацію. Оголошення структури MyData з атрибутами StructLayout і MarshalAs. Створення класу BinarySerializationExample. Створення екземпляру структури MyData та заповнення її даними. Створення копії структури даних. Виведення вмісту оригінального екземпляра та копії на консоль під час запуску гри. Створення мережевої бібліотеки NetLib для зберігання допоміжних функцій та класів, які будуть основою для розробки мережевих ігор, щоб уникнути дублювання великої кількості коду. Використання методів розширення ToJsonBinary() та FromJsonBinary() для серіалізації/десеріалізації об’єкта в бінарний формат JSON та з нього.

Тема 3. REST API

Лк5 "Мережне програмування" (денна)

Мережне програмування. Використання веб-сервісів для отримання даних для вашої гри (таблиці лідерів, списки друзів). Операції CRUD (Create, Read, Update, Delete) з даними. Поняття REST API. REST запити. Структура URL-адреси, яка формує запит на потрібні вам дані. Коренева кінцева точка та шлях до ресурсу. Документація API сервісу, що містить шляхи, які він пропонує. HTTP-заголовки. HTTP-запити POST, PUT, PATCH та DELETE. REST відповіді. GET-запит на таблицю лідерів та повернена JSON-відповідь. Автентифікація та обмеження. Відкрита автентифікація OAuth. Токени.

Лк6 "Асинхронні програми" (денна)

Асинхронне надсилання запитів та отримання відповідей. Корутина. Клас UnityWebRequest. Створення об’єкту UnityWebRequest за допомогою рядка URL-адреси. Методи UnityWebRequest.Get(), UnityWebRequest.Post(), UnityWebRequest.Put(). Надсилання запиту на сервер, використовуючи метод SendWebRequest(). Оброблення відповіді сервера. Перевірка властивостей об'єкту UnityWebRequest. Властивість isNetworkError. Властивість isHttpError. Властивість downloadHandler.text. Використання бібліотеки JsonUtility. Створення класів за допомогою JsonUtility.FromJson(). Метод Debug.Log(). Приклад використання корутини для здійснення виклику API за допомогою UnityWebRequest. Виклик корутини з іншого скрипта з використанням методу StartCoroutine(). Отримання HTML-коду веб-сторінки з Інтернет. Отримання інших ресурсів з Інтернет. HTTP-запит POST. Загальний клієнт REST API.

Пр5 "REST API" (денна)

Створіть новий 2D-проєкт в Unity. Додайте порожній GameObject до сцени під назвою Logo. Створіть скрипт для отримання зображення з вебсайту за допомогою UnityWebRequestTexture. Після запуску гри потрібно, щоб завантажувалось зображення та створювався спрайт. Для отримання статусу персонажів та відображення його в грі потрібно використати MockAPI. Створіть новий проєкт MockAPI. Створіть схему даних для вашого API. Додайте статуси персонажів, такі як: HP (Healt Point), MP (Magic Point), Attack, Defense та Agility. Створіть новий проєкт Unity. У Unity Hub оберіть 3D-шаблон для своєї гри. Створіть скрипт PlayerStatus.cs для даних вашої моделі, яка отримуватиме дані з REST API. Створіть скрипт GetApiData.cs для отримання та обробки даних у форматі JSON з REST API. Створіть текстовий інтерфейс користувача гри, кнопку та поле введення. Застосуйте ваш скрипт GetApiData.cs до вашої гри. Потрібно, щоб кнопка інтерфейсу користувача гри викликала функцію GetData() скрипту GetApiData.cs. Створіть власну гру з отриманням даних у форматі JSON і використанням публічного REST API або MockAPI.

Пр6 "REST API (продовження)" (денна)

Створіть новий 2D-проєкт в Unity. Додайте порожній GameObject до сцени під назвою Logo. Створіть скрипт для отримання зображення з вебсайту за допомогою UnityWebRequestTexture. Після запуску гри потрібно, щоб завантажувалось зображення та створювався спрайт. Для отримання статусу персонажів та відображення його в грі потрібно використати MockAPI. Створіть новий проєкт MockAPI. Створіть схему даних для вашого API. Додайте статуси персонажів, такі як: HP (Healt Point), MP (Magic Point), Attack, Defense та Agility. Створіть новий проєкт Unity. У Unity Hub оберіть 3D-шаблон для своєї гри. Створіть скрипт PlayerStatus.cs для даних вашої моделі, яка отримуватиме дані з REST API. Створіть скрипт GetApiData.cs для отримання та обробки даних у форматі JSON з REST API. Створіть текстовий інтерфейс користувача гри, кнопку та поле введення. Застосуйте ваш скрипт GetApiData.cs до вашої гри. Потрібно, щоб кнопка інтерфейсу користувача гри викликала функцію GetData() скрипту GetApiData.cs. Створіть власну гру з отриманням даних у форматі JSON і використанням публічного REST API або MockAPI.

Пр7 "REST API (продовження)" (денна)

Створіть новий 2D-проєкт в Unity. Додайте порожній GameObject до сцени під назвою Logo. Створіть скрипт для отримання зображення з вебсайту за допомогою UnityWebRequestTexture. Після запуску гри потрібно, щоб завантажувалось зображення та створювався спрайт. Для отримання статусу персонажів та відображення його в грі потрібно використати MockAPI. Створіть новий проєкт MockAPI. Створіть схему даних для вашого API. Додайте статуси персонажів, такі як: HP (Healt Point), MP (Magic Point), Attack, Defense та Agility. Створіть новий проєкт Unity. У Unity Hub оберіть 3D-шаблон для своєї гри. Створіть скрипт PlayerStatus.cs для даних вашої моделі, яка отримуватиме дані з REST API. Створіть скрипт GetApiData.cs для отримання та обробки даних у форматі JSON з REST API. Створіть текстовий інтерфейс користувача гри, кнопку та поле введення. Застосуйте ваш скрипт GetApiData.cs до вашої гри. Потрібно, щоб кнопка інтерфейсу користувача гри викликала функцію GetData() скрипту GetApiData.cs. Створіть власну гру з отриманням даних у форматі JSON і використанням публічного REST API або MockAPI.

Пр8 "REST API (продовження)" (денна)

Створіть новий 2D-проєкт в Unity. Додайте порожній GameObject до сцени під назвою Logo. Створіть скрипт для отримання зображення з вебсайту за допомогою UnityWebRequestTexture. Після запуску гри потрібно, щоб завантажувалось зображення та створювався спрайт. Для отримання статусу персонажів та відображення його в грі потрібно використати MockAPI. Створіть новий проєкт MockAPI. Створіть схему даних для вашого API. Додайте статуси персонажів, такі як: HP (Healt Point), MP (Magic Point), Attack, Defense та Agility. Створіть новий проєкт Unity. У Unity Hub оберіть 3D-шаблон для своєї гри. Створіть скрипт PlayerStatus.cs для даних вашої моделі, яка отримуватиме дані з REST API. Створіть скрипт GetApiData.cs для отримання та обробки даних у форматі JSON з REST API. Створіть текстовий інтерфейс користувача гри, кнопку та поле введення. Застосуйте ваш скрипт GetApiData.cs до вашої гри. Потрібно, щоб кнопка інтерфейсу користувача гри викликала функцію GetData() скрипту GetApiData.cs. Створіть власну гру з отриманням даних у форматі JSON і використанням публічного REST API або MockAPI.

Тема 4. UDP та TCP-з'єднання

Лк7 "Протоколи транспортного рівня UDP та TCP" (денна)

UDP-пакети. Переваги використання протоколу транспортного рівня UDP у застосунках реального часу, особливо у динамічних іграх. Заголовок та корисне навантаження пакету UDP. Поняття інкапсуляції. Метадані заголовків пакетів IP і UDP. Помилки пакетів UDP. Втрата пакетів UDP. Дублювання. Зміна порядку. Пошкодження. Порівняння протоколів транспортного рівня UDP та TCP. Методи зменшення ненадійності протоколу UDP. Механізм тристороннього рукостискання протоколу TCP. Повідомлення синхронізації (SYN) і підтвердження (ACK). Клієнт-серверні з'єднання TCP.

Лк8 "Класи Unity для роботи з UDP та TCP-з'єднаннями" (денна)

Використання класу TcpListener. Сокетне з’єднання з використанням класу Socket. Встановлення сокетного з'єднання з використанням методу Connect() і блоку try/catch. Приклад підключення до локального сервісу. Способи прийняття вхідного сокетного з'єднання за допомогою TcpListener. Синхронне та асинхронне надсилання та отримання даних. Синхронний метод AcceptSocket(). Асинхронний метод BeginAcceptSocket(). Синхронний метод Send(). Асинхронний метод BeginSend(). Об'єкт стану та метод зворотного виклику (callback). Надсилання великого обсягу даних кількома порціями. Синхронний метод Receive(). Асинхронний метод BeginReceive(). Приклад отримання великого файлу через TCP за допомогою сокета. Приклад об'єкта стану з сокетом та буфером. Використання TcpClient та TcpListener замість сокетів. З’єднання та надсилання даних за допомогою TcpClient. Асинхронні методи BeginWrite() та BeginRead() класу NetworkStream. Метод BeginAcceptTcpClient().

Пр9 "TCP-з'єднання" (денна)

Створіть новий 2D-проєкт в Unity. Створіть скрипт StateObject.cs, який використовується класом TcpListenSocketBehaviour. Створіть скрипти TcpListenSocketBehaviour.cs () і TcpSocketAsyncBehaviour.cs. TcpListenSocketBehaviour – це сервер, який очікує на підключення сокета та початок зв’язку. TcpSocketAsyncBehaviour – це клієнт. Він підключається до сервера та надсилає просте повідомлення. Воно перетворюється на масив байтів та надсилається на сервер за допомогою протоколу транспортного рівня TCP. На сервері повідомлення отримується та поміщається в локальний буфер. Щоб проілюструвати синхронний мережевий виклик, створіть програму, яка може копіювати файл з локального хоста на віддалений хост. Це зручно, коли потрібно скопіювати файли з одного комп’ютера на інший, але не хочеться налаштовувати FTP-сервер або мережевий ресурс. Створіть нову консольну програму у Visual Studio Code, використовуючи найновішу версію фреймворка .Net. Створіть файл класу з назвою Config.cs (аналізатор параметрів командного рядка), що аналізує такі параметри командного рядка: Filename, Port, IsServer, Debug, ServerIP, AskForHelp. Створіть файл класу з назвою Receiver.cs, що отримує дані від віддаленого клієнта та використовує буфери. Створіть файл класу з назвою Sender.cs, що використовується для надсилання даних на віддалений сервер. Створіть файл класу з назвою Program.cs (точка входу в програму).

Пр10 "TCP-з'єднання (продовження)" (денна)

Створіть новий 2D-проєкт в Unity. Створіть скрипт StateObject.cs, який використовується класом TcpListenSocketBehaviour. Створіть скрипти TcpListenSocketBehaviour.cs () і TcpSocketAsyncBehaviour.cs. TcpListenSocketBehaviour – це сервер, який очікує на підключення сокета та початок зв'язку. TcpSocketAsyncBehaviour – це клієнт. Він підключається до сервера та надсилає просте повідомлення. Воно перетворюється на масив байтів та надсилається на сервер за допомогою протоколу транспортного рівня TCP. На сервері повідомлення отримується та поміщається в локальний буфер. Щоб проілюструвати синхронний мережевий виклик, створіть програму, яка може копіювати файл з локального хоста на віддалений хост. Це зручно, коли потрібно скопіювати файли з одного комп'ютера на інший, але не хочеться налаштовувати FTP-сервер або мережевий ресурс. Створіть нову консольну програму у Visual Studio Code, використовуючи найновішу версію фреймворка .Net. Створіть файл класу з назвою Config.cs (аналізатор параметрів командного рядка), що аналізує такі параметри командного рядка: Filename, Port, IsServer, Debug, ServerIP, AskForHelp. Створіть файл класу з назвою Receiver.cs, що отримує дані від віддаленого клієнта та використовує буфери. Створіть файл класу з назвою Sender.cs, що використовується для надсилання даних на віддалений сервер. Створіть файл класу з назвою Program.cs (точка входу в програму).

Пр11 "TCP-з'єднання (продовження)" (денна)

Створіть новий 2D-проєкт в Unity. Створіть скрипт StateObject.cs, який використовується класом TcpListenSocketBehaviour. Створіть скрипти TcpListenSocketBehaviour.cs () і TcpSocketAsyncBehaviour.cs. TcpListenSocketBehaviour – це сервер, який очікує на підключення сокета та початок зв'язку. TcpSocketAsyncBehaviour – це клієнт. Він підключається до сервера та надсилає просте повідомлення. Воно перетворюється на масив байтів та надсилається на сервер за допомогою протоколу транспортного рівня TCP. На сервері повідомлення отримується та поміщається в локальний буфер. Щоб проілюструвати синхронний мережевий виклик, створіть програму, яка може копіювати файл з локального хоста на віддалений хост. Це зручно, коли потрібно скопіювати файли з одного комп'ютера на інший, але не хочеться налаштовувати FTP-сервер або мережевий ресурс. Створіть нову консольну програму у Visual Studio Code, використовуючи найновішу версію фреймворка .Net. Створіть файл класу з назвою Config.cs (аналізатор параметрів командного рядка), що аналізує такі параметри командного рядка: Filename, Port, IsServer, Debug, ServerIP, AskForHelp. Створіть файл класу з назвою Receiver.cs, що отримує дані від віддаленого клієнта та використовує буфери. Створіть файл класу з назвою Sender.cs, що використовується для надсилання даних на віддалений сервер. Створіть файл класу з назвою Program.cs (точка входу в програму).

Пр12 "TCP-з'єднання (продовження)" (денна)

Створіть новий 2D-проєкт в Unity. Створіть скрипт StateObject.cs, який використовується класом TcpListenSocketBehaviour. Створіть скрипти TcpListenSocketBehaviour.cs () і TcpSocketAsyncBehaviour.cs. TcpListenSocketBehaviour – це сервер, який очікує на підключення сокета та початок зв’язку. TcpSocketAsyncBehaviour – це клієнт. Він підключається до сервера та надсилає просте повідомлення. Воно перетворюється на масив байтів та надсилається на сервер за допомогою протоколу транспортного рівня TCP. На сервері повідомлення отримується та поміщається в локальний буфер. Щоб проілюструвати синхронний мережевий виклик, створіть програму, яка може копіювати файл з локального хоста на віддалений хост. Це зручно, коли потрібно скопіювати файли з одного комп’ютера на інший, але не хочеться налаштовувати FTP-сервер або мережевий ресурс. Створіть нову консольну програму у Visual Studio Code, використовуючи найновішу версію фреймворка .Net. Створіть файл класу з назвою Config.cs (аналізатор параметрів командного рядка), що аналізує такі параметри командного рядка: Filename, Port, IsServer, Debug, ServerIP, AskForHelp. Створіть файл класу з назвою Receiver.cs, що отримує дані від віддаленого клієнта та використовує буфери. Створіть файл класу з назвою Sender.cs, що використовується для надсилання даних на віддалений сервер. Створіть файл класу з назвою Program.cs (точка входу в програму).

Тема 5. Проблеми мережних з'єднань

Лк9 "Проблеми мережних з'єднань" (денна)

Авторитетні сервери. Синхронні та асинхронні ігри. Планування багатокористувацьких ігор. Завантаження даних. Шахрайство. Локальний сервер. Пропускна здатність мережного з’єднання. Симетричні та асиметричні з'єднання. Фактори, які можуть впливати на продуктивність мережі.

Лк10 "Затримка в мережі" (денна)

Фактори, які спричиняють затримку в мережі. Затримка між клієнтом і сервером. Затримка розповсюдження. Затримка доставки. Затримка оброблення. Лінійна затримка. Приклад визначення затримки в мережі за допомогою утиліти командного рядка traceroute. Прогнозування на стороні клієнта. Проблема синхронізації. Узгодження з сервером. Пріоритетне оновлення.

Тема 6. Ігрові сервери

Лк11 "Використання серверів у ігровій індустрії" (денна)

Використання серверів у іграх. Публічні сервери. Віртуальні приватні сервери (VPS). Характеристики виділених серверів. Масштабованість. Безпека. Швидкість. Контроль. Виділені сервери в ігровій індустрії. Безголові сервери в ігровій індустрії. Віддалене адміністрування безголових серверів із використанням SSH(Secure Shell). Сервер прослуховування, розміщений на клієнті. Розподілені повноваження. Локальний багатокористувацький режим.

Лк12 "Сучасні мережні рішення в ігровій індустрії" (денна)

Однорангова топологія (P2P). Переваги однорангових мереж. Відсутність центрального сервера. Відсутність управління чергами. Відсутність простоїв. Відсутність втрати гравців. Переваги багатосерверної архітектури. Гібридна архітектура. Функціональність балансувальника навантаження. Апаратні та програмні балансувальники навантаження. Ігри в локальних комп’ютерних мережах. Налаштування локальної комп'ютерної мережі з кількома пристроями та маршрутизатором. Віртуальні приватні мережі (VPN). Віртуальне тунелювання. Шифрування даних. Використання VPN сервісу Hamachi.

Пр13 "Створення клієнт-серверної гри" (денна)

Потрібно створити клієнт-серверну версію гри «Хрестики-нулики» з використанням класів TcpClient та TcpListener. Один гравець виступає в ролі сервера, а інший – підключається як клієнт. Клас TcpClient використовує поток NetworkStream для читання та запису даних. Можна використовувати метод BinaryWriter для запису даних та метод BinaryReader для читання даних з потоку NetworkStream. Вони дозволяють передавати об’єкт стану, щоб встановити контекст у зворотному виклику (callback). Наприклад, під час завершення запису об’єктом стану буде клієнт, який виконує операцію запису. Щоб додати більше функціональності, ці класи потрібно загорнути у ваші власні класи, які повинні вести список клієнтів, підключених до сервера, обмін повідомленнями про події та серіалізацію/десеріалізацію. Вам потрібно буде включити до свого протоколу передачі даних спосіб, щоб повідомити одержувачу, який розмір корисного навантаження ви маєте намір надіслати. Для передачі даних між клієнтом і сервером можна застосувати простий протокол серіалізації. Повідомлення, що передаються, - прості текстові повідомлення, перший рядок яких містить заголовок із розміром корисного навантаження. Воно включає стан ігрової дошки «хрестики-нулики» та поточного гравця, а також команду, яка повідомляє клієнтській програмі, в якому стані вона має бути. Клієнт і сервер використовують кінцевий автомат для керування поточним станом програми. Створіть власну клієнт-серверну гру.

Пр14 "Створення клієнт-серверної гри (продовження)" (денна)

Потрібно створити клієнт-серверну версію гри «Хрестики-нулики» з використанням класів TcpClient та TcpListener. Один гравець виступає в ролі сервера, а інший – підключається як клієнт. Клас TcpClient використовує поток NetworkStream для читання та запису даних. Можна використовувати метод BinaryWriter для запису даних та метод BinaryReader для читання даних з потоку NetworkStream. Вони дозволяють передавати об’єкт стану, щоб встановити контекст у зворотному виклику (callback). Наприклад, під час завершення запису об’єктом стану буде клієнт, який виконує операцію запису. Щоб додати більше функціональності, ці класи потрібно загорнути у ваші власні класи, які повинні вести список клієнтів, підключених до сервера, обмін повідомленнями про події та серіалізацію/десеріалізацію. Вам потрібно буде включити до свого протоколу передачі даних спосіб, щоб повідомити одержувачу, який розмір корисного навантаження ви маєте намір надіслати. Для передачі даних між клієнтом і сервером можна застосувати простий протокол серіалізації. Повідомлення, що передаються, - прості текстові повідомлення, перший рядок яких містить заголовок із розміром корисного навантаження. Воно включає стан ігрової дошки «хрестики-нулики» та поточного гравця, а також команду, яка повідомляє клієнтській програмі, в якому стані вона має бути. Клієнт і сервер використовують кінцевий автомат для керування поточним станом програми. Створіть власну клієнт-серверну гру.

Пр15 "Створення клієнт-серверної гри (продовження)" (денна)

Потрібно створити клієнт-серверну версію гри «Хрестики-нулики» з використанням класів TcpClient та TcpListener. Один гравець виступає в ролі сервера, а інший – підключається як клієнт. Клас TcpClient використовує поток NetworkStream для читання та запису даних. Можна використовувати метод BinaryWriter для запису даних та метод BinaryReader для читання даних з потоку NetworkStream. Вони дозволяють передавати об’єкт стану, щоб встановити контекст у зворотному виклику (callback). Наприклад, під час завершення запису об’єктом стану буде клієнт, який виконує операцію запису. Щоб додати більше функціональності, ці класи потрібно загорнути у ваші власні класи, які повинні вести список клієнтів, підключених до сервера, обмін повідомленнями про події та серіалізацію/десеріалізацію. Вам потрібно буде включити до свого протоколу передачі даних спосіб, щоб повідомити одержувачу, який розмір корисного навантаження ви маєте намір надіслати. Для передачі даних між клієнтом і сервером можна застосувати простий протокол серіалізації. Повідомлення, що передаються, - прості текстові повідомлення, перший рядок яких містить заголовок із розміром корисного навантаження. Воно включає стан ігрової дошки «хрестики-нулики» та поточного гравця, а також команду, яка повідомляє клієнтській програмі, в якому стані вона має бути. Клієнт і сервер використовують кінцевий автомат для керування поточним станом програми. Створіть власну клієнт-серверну гру.

Пр16 "Створення клієнт-серверної гри (продовження)" (денна)

Потрібно створити клієнт-серверну версію гри «Хрестики-нулики» з використанням класів TcpClient та TcpListener. Один гравець виступає в ролі сервера, а інший – підключається як клієнт. Клас TcpClient використовує поток NetworkStream для читання та запису даних. Можна використовувати метод BinaryWriter для запису даних та метод BinaryReader для читання даних з потоку NetworkStream. Вони дозволяють передавати об’єкт стану, щоб встановити контекст у зворотному виклику (callback). Наприклад, під час завершення запису об’єктом стану буде клієнт, який виконує операцію запису. Щоб додати більше функціональності, ці класи потрібно загорнути у ваші власні класи, які повинні вести список клієнтів, підключених до сервера, обмін повідомленнями про події та серіалізацію/десеріалізацію. Вам потрібно буде включити до свого протоколу передачі даних спосіб, щоб повідомити одержувачу, який розмір корисного навантаження ви маєте намір надіслати. Для передачі даних між клієнтом і сервером можна застосувати простий протокол серіалізації. Повідомлення, що передаються, - прості текстові повідомлення, перший рядок яких містить заголовок із розміром корисного навантаження. Воно включає стан ігрової дошки «хрестики-нулики» та поточного гравця, а також команду, яка повідомляє клієнтській програмі, в якому стані вона має бути. Клієнт і сервер використовують кінцевий автомат для керування поточним станом програми. Створіть власну клієнт-серверну гру.

Тема 7. Програмні рішення Unity для розробки багатокористувацьких ігор

Лк13 "Програмні рішення Unity для розробки багатокористувацьких ігор" (денна)

Переваги Unity для розробки багатокористувацьких ігор. Прототипування. Екосистема. Кросплатформенність. Спільнота та документація. Оптимізація та продуктивність. Інтеграція фізики. Інсталяція бібліотеки Unity Netcode for GameObjects. Додавання до проєкту компоненти NetworkManager. Додавання NetworkObject до GameObject. Ідентифікатори GlobalObjectIdHash, NetworkObjectId, OwnerClientId. Netcode компоненти NetworkObject, NetworkBehaviour, NetworkAnimator, NetworkTransform. Створення NetworkObject гравця. Використання NetworkBehaviour для керування рухом гравця. Властивості повноважень та власності IsClient, IsServer, IsHost, IsLocalPlayer, IsOwner, IsPlayerObject, IsSceneObject. Застосування повноважень клієнта з використанням NetworkTransform.

Лк14 "Програмні рішення Unity для моделювання мережевої взаємодії" (денна)

Компоненти режиму авторизації власника. Синхронізація з авторизацією сервера. Мережева синхронізація. Порівняння RPC (Remote Procedure Call) та NetworkVariable. Моделювання затримки мережі. Налаштування Debug Simulator у компоненті UnityTransport. Використання Network Simulator. Антиципація на стороні клієнта із використанням бібліотеки Netcode for GameObjects. Прогнозування на стороні клієнта із використанням бібліотеки Netcode for Entities.

Пр17 "Створення багатокористувацької гри" (денна)

Створіть імітацію простого 3D-геймплея з гуманоїдним персонажем за допомогою Universal Render Pipeline. Проєкт включає невелику сцену для тестування ігрового майданчика та налаштовуваний контролер від третьої особи. Мета проєкту полягає в тому, щоб запустити кілька копій цієї програми, а потім забезпечити взаємодію різних клієнтів в одному середовищі. Студенти використовуватимуть стартовий пакет Starter Assets – ThirdPerson з Unity Asset Store. Його потрібно імпортувати за допомогою Package Manager. Для розробки багатокористувацької гри портібно встановити та використати такі пакети Unity: Netcode for GameObjects (додає можливості багатокористувацького режиму до існуючого робочого процесу GameObject/MonoBehaviour), Multiplayer Tools Window (покращує робочі процеси для розробки багатокористувацького режиму), Multiplayer Play Mode (дозволяє тестувати функціональність багатокористувацької гри). Студенти використовуватимуть компонент NetworkManager для підтримки мережного багатокористувацького режиму. Він керує станом мережі проєкту, обробляючи з’єднання та конфігурації мережі. У компоненті NetworkManager необхідно налаштувати рівень мережного транспорту як UnityTransport. Логіка гравця включає комбінацію MonoBehaviours для безпосередньої ігрової механіки та NetworkBehaviours для керування станами мережі. Немережні компоненти, такі як контролери персонажів та аніматори, функціонують на локальному екземплярі кожного гравця.

Пр18 "Створення багатокористувацької гри (продовження)" (денна)

Створіть імітацію простого 3D-геймплея з гуманоїдним персонажем за допомогою Universal Render Pipeline. Проєкт включає невелику сцену для тестування ігрового майданчика та налаштовуваний контролер від третьої особи. Мета проєкту полягає в тому, щоб запустити кілька копій цієї програми, а потім забезпечити взаємодію різних клієнтів в одному середовищі. Студенти використовуватимуть стартовий пакет Starter Assets – ThirdPerson з Unity Asset Store. Його потрібно імпортувати за допомогою Package Manager. Для розробки багатокористувацької гри портібно встановити та використати такі пакети Unity: Netcode for GameObjects (додає можливості багатокористувацького режиму до існуючого робочого процесу GameObject/MonoBehaviour), Multiplayer Tools Window (покращує робочі процеси для розробки багатокористувацького режиму), Multiplayer Play Mode (дозволяє тестувати функціональність багатокористувацької гри). Студенти використовуватимуть компонент NetworkManager для підтримки мережного багатокористувацького режиму. Він керує станом мережі проєкту, обробляючи з'єднання та конфігурації мережі. У компоненті NetworkManager необхідно налаштувати рівень мережного транспорту як UnityTransport. Логіка гравця включає комбінацію MonoBehaviours для безпосередньої ігрової механіки та NetworkBehaviours для керування станами мережі. Немережні компоненти, такі як контролери персонажів та аніматори, функціонують на локальному екземплярі кожного гравця.

Пр19 "Створення багатокористувацької гри (продовження)" (денна)

Створіть імітацію простого 3D-геймплея з гуманоїдним персонажем за допомогою Universal Render Pipeline. Проєкт включає невелику сцену для тестування ігрового майданчика та налаштовуваний контролер від третьої особи. Мета проєкту полягає в тому, щоб запустити кілька копій цієї програми, а потім забезпечити взаємодію різних клієнтів в одному середовищі. Студенти використовуватимуть стартовий пакет Starter Assets – ThirdPerson з Unity Asset Store. Його потрібно імпортувати за допомогою Package Manager. Для розробки багатокористувацької гри портібно встановити та використати такі пакети Unity: Netcode for GameObjects (додає можливості багатокористувацького режиму до існуючого робочого процесу GameObject/MonoBehaviour), Multiplayer Tools Window (покращує робочі процеси для розробки багатокористувацького режиму), Multiplayer Play Mode (дозволяє тестувати функціональність багатокористувацької гри). Студенти використовуватимуть компонент NetworkManager для підтримки мережного багатокористувацького режиму. Він керує станом мережі проєкту, обробляючи з'єднання та конфігурації мережі. У компоненті NetworkManager необхідно налаштувати рівень мережного транспорту як UnityTransport. Логіка гравця включає комбінацію MonoBehaviours для безпосередньої ігрової механіки та NetworkBehaviours для керування станами мережі. Немережні компоненти, такі як контролери персонажів та аніматори, функціонують на локальному екземплярі кожного гравця.

Пр20 "Створення багатокористувацької гри (продовження)" (денна) Створіть імітацію простого 3D-геймплея з гуманоїдним персонажем за допомогою Universal Render Pipeline. Проєкт включає невелику сцену для тестування ігрового майданчика та налаштовуваний контролер від третьої особи. Мета проєкту полягає в тому, щоб запустити кілька копій цієї програми, а потім забезпечити взаємодію різних клієнтів в одному середовищі. Студенти використовуватимуть стартовий пакет Starter Assets – ThirdPerson з Unity Asset Store. Його потрібно імпортувати за допомогою Package Manager. Для розробки багатокористувацької гри портібно встановити та використати такі пакети Unity: Netcode for GameObjects (додає можливості багатокористувацького режиму до існуючого робочого процесу GameObject/MonoBehaviour), Multiplayer Tools Window (покращує робочі процеси для розробки багатокористувацького режиму), Multiplayer Play Mode (дозволяє тестувати функціональність багатокористувацької гри). Студенти використовуватимуть компонент NetworkManager для підтримки мережного багатокористувацького режиму. Він керує станом мережі проєкту, обробляючи з'єднання та конфігурації мережі. У компоненті NetworkManager необхідно налаштувати рівень мережного транспорту як UnityTransport. Логіка гравця включає комбінацію MonoBehaviours для безпосередньої ігрової механіки та NetworkBehaviours для керування станами мережі. Немережні компоненти, такі як контролери персонажів та аніматори, функціонують на локальному екземплярі кожного гравця.
Тема 8. Тестування та налагодження мережевих ігор
Лк15 "Тестування мережевих ігор" (денна) Локальне тестування. Збірки гравців. Режим багатокористувацької гри. Імітація мережевих умов. Інструмент Network Simulator. Тестування клієнтських підключень. Методи налагодження багатокористувацьких ігор. Налагодження малювання. Рендерер ліній. Текстове ведення журналу. Записи екрану. Збільшення фіксованого часового кроку. Використання точок зупинки. Створення помічника командного рядка для тестування.
Лк16 "Налагодження мережевих ігор" (денна) Сервіси Unity для створення багатокористувацьких ігор. Сервіс Matchmaker для зв'язування гравців з іншими гравцями або ігровими сесіями на основі таких критеріїв, як рівень майстерності чи географічне розташування, для забезпечення збалансованого ігрового досвіду. Визначення критеріїв підбору партнерів. Профілі та рейтинги гравців. Запити на підбір партнерів. Динамічні коригування. Використання бібліотек Lobby (передігрова зона, де гравці можуть зібратися, налаштувати параметри гри та підготуватися до початку гри) та Relay (забезпечення простого та безпечного з'єднання між клієнтами) для створення способу знаходити спільну мову між гравцями та взаємодіяти один з одним в різних багатокористувацьких ігрових сценаріях. Багатокористувацький хостинг. Спілкування гравців у чаті Vivox.

9. Стратегія викладання та навчання

9.1 Методи викладання та навчання

Дисципліна передбачає навчання через:

МН1	Лекційне навчання
МН2	Практикоорієнтоване навчання

Лекції надають студентам матеріали з основ клієнт-серверної взаємодії, серіалізації та

десеріалізації, мережевого програмування, використання API веб-сервісів для отримання даних, використання класів Unity для роботи з UDP та TCP-з'єднаннями, виявлення та усунення проблем мережевих з'єднань, використання серверів у ігровій індустрії, програмних рішень Unity для розробки багатокористувацьких ігор та моделювання мережевої взаємодії, тестування та налагодження мережевих ігор (РН1, РН2). Лекції доповнюються практичними заняттями, що надають студентам можливість застосовувати теоретичні знання на практичних прикладах (РН1, РН2, РН3, РН4). Зміст практичних робіт спрямований на практикоорієнтоване навчання, що передбачає засвоєння студентами клієнт-серверних технологій в ігровій індустрії, розвиток вмінь студентів працювати з сучасним інструментарієм розробки комп'ютерних ігор (РН1, РН2, РН3, РН4).

9.2 Види навчальної діяльності

НД1	Підготовка до опитування
НД2	Виконання та презентація результатів практичної роботи
НД3	Захист звіту про виконання практичної роботи
НД4	Самостійне виконання індивідуального завдання (контрольної роботи)

10. Методи та критерії оцінювання

10.1. Критерії оцінювання

Визначення	Чотирибальна національна шкала оцінювання	Рейтингова бальна шкала оцінювання
Відмінне виконання лише з незначною кількістю помилок	5 (відмінно)	$90 \leq RD \leq 100$
Вище середнього рівня з кількома помилками	4 (добре)	$82 \leq RD < 89$
Загалом правильна робота з певною кількістю помилок	4 (добре)	$74 \leq RD < 81$
Непогано, але зі значною кількістю недоліків	3 (задовільно)	$64 \leq RD < 73$
Виконання задовольняє мінімальним критеріям	3 (задовільно)	$60 \leq RD < 63$
Можливе повторне складання	2 (незадовільно)	$21 \leq RD < 59$
Можливе одноразове повторне складання	2 (незадовільно)	$0 \leq RD < 20$

10.2 Методи поточного формативного оцінювання

	Характеристика	**Дедлайн, тижні**	**Зворотний зв'язок**
МФО1 Опитування та усні коментарі викладача за його результатами	Опитування студентів щодо їх готовності до виконання практичної роботи. Виконання роботи відбувається студентом самостійно після отримання завдання від викладача. Викладач надає усні коментарі за результатами опитування студентів	1-16	mix.sumdu.edu.ua

МФО2 Обговорення та самокорекція виконаної роботи студентами	Обговорення ходу виконання роботи студентами. Виконання роботи відбувається студентом самостійно піля отримання завдання від викладача. Студенти здійснюють самокорекцію виконаної роботи за результатами обговорення/	1-16	mix.sumdu.edu.ua
МФО3 Настанови викладача в процесі виконання практичних робіт	Пояснення умов завдань щодо виконання практичних робіт з наведенням прикладів	1-16	mix.sumdu.edu.ua

10.3 Методи підсумкового сумативного оцінювання

	Характеристика	**Дедлайн, тижні**	**Зворотний зв'язок**
МСО1 Підсумковий контроль: екзамен			
МСО2 Звіт за результатами виконання практичних робіт	Захист роботи відбувається у форматі "презентації" та усних (письмових) відповідей на контрольні питання до теми, "на запитання викладача"	1-16	mix.sumdu.edu.ua
МСО3 Виконання індивідуального завдання (контрольної роботи)	Захист індивідуального завдання відбувається у форматі "презентації" та усних (письмових) відповідей на контрольні питання до теми, "на запитання викладача"	15	mix.sumdu.edu.ua

Контрольні заходи:

	Максимальна кількість балів	Можливість перескладання з метою підвищення оцінки
Перший семестр вивчення	**100 балів**	
МСО1. Підсумковий контроль: екзамен	**40**	
	40	Ні
МСО2. Звіт за результатами виконання практичних робіт	**50**	

	5x10	50	Ні
МСО3. Виконання індивідуального завдання (контрольної роботи)		**10**	
		10	Ні

Оцінювання знань студента під час практичних занять має на меті перевірку рівня підготовленості студента до виконання конкретної роботи, для чого студент в аудиторії проходить опитування (МФО1), обговорення та самокорекцію виконаної роботи (МФО2), контроль активності під час виконання роботи (МФО3). Оцінювання практичних робіт: 10 балів - студент дає розгорнуту відповідь у звіті до виконаної практичної роботи, робота виконана в повному обсязі; 8 балів - студент дає розгорнуту відповідь, виявляє ґрунтовні знання матеріалу, однак припускає незначних помилок, 6 балів - студент недостатньо повно відповідає на поставлені питання до захисту практичної роботи, припускає помилки; 4 бали - запропонована ідея виконання практичної роботи і зроблено деякий поступ до її реалізації, студент частково дає правильні відповіді на поставлені питання до захисту практичної роботи; 2 бали - виконано 30% завдань практичної роботи та студент надає відповіді на поставлені питання щодо виконаних завдань; 0 балів - практична робота не виконана. Оцінювання індивідуального завдання (контрольної роботи) наведено у методичних рекомендаціях до її виконання.

11. Ресурсне забезпечення навчальної дисципліни

11.1 Засоби навчання

ЗН1	Прикладне програмне забезпечення (перелік конкретизується викладачем)
ЗН2	Інформаційно-комунікаційні системи
ЗН3	Бібліотечні фонди

11.2 Інформаційне та навчально-методичне забезпечення

Основна література	
1	The ultimate guide to multiplayer networking for advanced Unity developers. Unity Technologies. URL: https://unity.com/resources/ultimate-guide-advanced-multiplayer-networking
2	Серверні веб-технології [Електронний ресурс] : навч. посіб. / уклад. О.С. Бунке. — Київ : КПІ ім. Ігоря Сікорського, 2023. — 109 с.
3	Веб-програмування та веб-дизайн [Електронний ресурс] : навч. посіб. / К. В. Двірничук, Д. О. Вацек. — Чернівці : Чернівецький нац. ун-т ім. Ю. Федьковича, 2022. — 472 с.
Допоміжна література	
1	Rodrigo Silveira. Getting Started with Multiplayer Game Programming. URL: https://www.packtpub.com/en-ua/learning/how-to-tutorials/getting-started-multiplayer-game-programming

2	A Complete Guide to Multiplayer Video Game Development. Slavna Game Studio. URL: https://www.slavnastudio.com/blog/a-complete-guide-to-multiplayer-video-game-development/
3	Joshua Glazer, Sanjay Madhav. Multiplayer game programming. Addison-Wesley. URL: https://github.com/kurong00/GameProgramBooks/tree/master/11.Multiplayer%20Game%20Programming
Інформаційні ресурси в Інтернеті	
1	Unity Manual. Multiplayer. Unity Documentation. URL: https://docs.unity3d.com/Manual/multiplayer.html
2	Learn How to Make Multiplayer Games. Code Monkey. URL: https://unitycodemonkey.com/kitchenchaosmultiplayercourse.php#intro