УДК 004.414.3:004.514
DOI: 10.31673/2412-9070.2025.022617
А. О. ПЕРЕПЕЛИЦЯ, магістр;
ORCID: 0009-0000-5223-3728
О. А. ДІБРІВНИЙ, доктор філософії, доцент,
ORCID: 0009-0007-8881-359X
Державний університет інформаційно-комунікаційних технологій, Київ

АНАЛІЗ ПІДХОДІВ ДО АДАПТИВНОГО КОМПОНУВАННЯ ТА ІДЕНТИФІКУВАННЯ ЕЛЕМЕНТІВ В IMMEDIATE MODE GUI ДЛЯ ЇХ ВИКОРИСТАННЯ В UNITY

Immediate Mode GUI (IMGUI) активно використовується для створення графічних інтерфейсів у програмуванні. Багато сучасних бібліотек реалізують цю парадигму для забезпечення динамічного відображення інтерфейсу у режимі реального часу. Використання IMGUI дає можливість ефективно відтворювати елементи на екрані кожного кадру та обробляти взаємодію користувача безпосередньо через виклики функцій.

Однак IMGUI має виклики, пов'язані з адаптивним компонуванням елементів та їхньою ідентифікацією між кадрами. У статті розглянуто дві ключові проблеми IMGUI: адаптивне компонування та ідентифікацію елементів. Проаналізовано різні підходи до вирішення проблеми адаптивного компонування, зокрема методи, що вимагають повторного виконання коду або затримують реакцію програми на один кадр. Визначено, що підхід з відтермінуванням вводу є більш перспективним. Окремо розглянуто механізми ідентифікації елементів інтерфейсу між кадрами, включаючи автоматичну та ручну генерацію ідентифікаторів. На основі аналізу зроблено висновок, що в середовищі Unity найоптимальнішим способом ідентифікації елементів є хешування текстових міток, завдяки чому забезпечується баланс між зручністю реалізації та продуктивністю системи.

Отримані результати можуть бути використані при розробці графічних інтерфейсів на основі IMGUI, зокрема у високопродуктивних застосунках, де критичною є швидкість оновлення кадрів та ефективність обробки подій.

Ключові слова: IMGUI, GUI, адаптивне компонування, бібліотека, парадигма, архітектура, підхід, ідентифікація.

Постановка проблеми

У сучасному програмуванні графічних інтерфейсів користувача (GUI) використовуються різні підходи до їхньої побудови, серед яких виділяється парадигма Immediate Mode GUI (IMGUI). Вона ґрунтується на тому, що інтерфейс генерується кожного кадру безпосередньо під час виконання програми, а взаємодія з користувачем визначається через виклики відповідних функцій [5]. Такий підхід відрізняється від традиційних (Retained Mode GUI), де структура інтерфейсу зберігається у вигляді окремих об'єктів і оновлюється лише за необхідності. Завдяки простоті реалізації та ефективному використанню ресурсів, парадигма Immediate Mode GUI (IMGUI) активно використовується у розробці графічних інтерфейсів для ігрових рушіїв, налагоджувальних інструментів та інтерактивних застосунків. Однак її застосування пов'язане з двома ключовими проблемними напрямами: адаптивним компонуванням інтерфейсу та ідентифікацією елементів.

Проблематика адаптивного компонування полягає в тому, що під час побудови інтерфейсу в IMGUI неможливо одразу визначити розміри та розташування кожного елемента без попереднього збору повної інформації про всі інші елементи. Це суперечить принципу негайної обробки подій, закладеному в архітектуру IMGUI.

Ідентифікація елементів є другою важливою складовою, оскільки забезпечує збереження стану та коректну обробку взаємодії між кадрами. Існуючі підходи або генерують ідентифіка-

тори автоматично, що обмежує їх застосування у складних динамічних структурах, або вимагають ручного втручання розробника, ускладнюючи процес розробки.

Виходячи з цього, проблема полягає у необхідності обґрунтувати та проаналізувати підходи до побудови адаптивного інтерфейсу та ідентифікації елементів у IMGUI, які б забезпечили високу продуктивність системи, точність взаємодії та зручність для розробника.

Ця проблема має безпосередній зв'язок із важливими науковими та практичними завданнями у галузі комп'ютерної графіки, розробки ігрових рушіїв та інтерактивних систем. Від ефективності її вирішення залежить здатність сучасних інтерфейсів працювати у режимі реального часу з великою кількістю елементів, забезпечуючи при цьому високу якість користувацького досвіду та мінімальне споживання ресурсів системи.

Аналіз останніх досліджень

Концепція Immediate Mode GUI (IMGUI) виникла як продовження ідей Immediate Mode API у комп'ютерній графіці, де найбільш відомим прикладом є OpenGL. У таких системах набір графічних примітивів зберігається на стороні клієнта, а виклики функцій безпосередньо призводять до візуалізації примітивів без створення окремих об'єктів або структур даних для подальших змін [7]. Найбільш відомою IMGUI-бібліотекою сьогодні є Dear ImGui, що широко застосовується у розробці ігор та налагоджувальних інструментів. Водночас парадигма Retained Mode GUI (RMGUI) є більш поширеною та зрозумілою серед розробників [9]. У RMGUI елементи інтерфейсу створюються як об'єкти із збереженням стану, які можна змінювати після створення, наприклад, викликом методів hide() для приховування або setColor() для зміни кольору. Прикладами реалізації цієї парадигми є HTML та бібліотека React.

На сьогодні кількість наукових публікацій, присвячених безпосередньо концепції та особливостям IMGUI, залишається вкрай обмеженою. Більшість матеріалів представлено технічними описами бібліотек, статтями розробників або блогами, що ускладнює системний науковий аналіз.

Однією з небагатьох наукових робіт, які досліджують IMGUI, є праця С. Л. ван Бернема [2]. У цій роботі автор демонструє ефективність застосування IMGUI у традиційних настільних застосунках та описує реалізацію прототипної однозаголовкової бібліотеки NBUI. Важливо, що ця бібліотека за продуктивністю не поступається Retained Mode рішенням, водночас забезпечуючи більшу гнучкість у поведінці віджетів та підтримуючи «гаряче» перезавантаження макетів інтерфейсу для декларативного проєктування.

Додатковий внесок у вивчення IMGUI зробили Ф. Брендель та С. Лідтке [1], які дослідили можливість використання цієї парадигми для побудови інтерфейсів у додатках з доповненою реальністю. Вони підкреслили, що через постійні зміни розташування елементів у тривимірному просторі та залежність вигляду інтерфейсу від відстані до об'єктів, IMGUI є більш гнучким та менш схильним до помилок у порівнянні з RMGUI, оскільки не потребує постійної синхронізації моделі та вигляду інтерфейсу.

Отже, аналіз літературних джерел дозволяє зробити висновок, що ступінь наукового дослідження тематики IMGUI залишається низькою, незважаючи на актуальність проблематики для сучасних застосунків у галузях комп'ютерної графіки, ігрової індустрії, доповненої та віртуальної реальностей. Подальші дослідження є необхідними для системного аналізу існуючих рішень та розробки ефективних підходів до адаптивного компонування та ідентифікації елементів в межах цієї парадигми.

Метою статті є визначення відмінних особливостей існуючих бібліотек Immediate Mode GUI, які забезпечують їхню зручність використання та високу продуктивність. Особлива увага буде приділена аналізу підходів до компонування та ідентифікації елементів, що дозволить визначити рішення, найбільш придатні для застосування у середовищі Unity, завдяки чому буде створене підгрунтя для подільшої розробки IMGUI-бібліотеки, здатної забезпечити створення не лише налагоджувальних інтерфейсів, а й повноцінних внутрішньо-програмних графічних інтерфейсів для широкого кола завдань.

Основна частина

На сьогодні існує значна кількість бібліотек, що реалізують парадигму Immediate Mode GUI (IMGUI), серед яких найбільш відомими є Dear ImGui, egui, nuklear та vui. У багатьох випадках модуль графічного інтерфейсу, побудований за принципами IMGUI, не виокремлюється як самостійна бібліотека, а функціонує як інтегрована частина прикладного програмного забезпечення. Такий підхід, зокрема, застосовано у RAD Debugger. Крім того, низка рішень на базі IMGUI є закритою за своїм вихідним кодом, що обмежує можливість детального аналізу їхньої внутрішньої структури. Відомості про особливості таких систем переважно отримуються зі слів авторів або офіційних матеріалів. Прикладами подібних реалізацій є програма Disk Voyager та ігровий рушій компанії Keen Games.

Варто підкреслити, що різні IMGUI-бібліотеки демонструють суттєві відмінності у підходах до вирішення окремих функціональних завдань. Найбільш значущими серед них є адаптивне компонування елементів інтерфейсу та ідентифікування елементів, від ефективності реалізації яких залежить зручність використання бібліотеки та її продуктивність.

Адаптивне компонування

Найбільш простою формою організації компонування у бібліотеках IMGUI є ручне компонування, при якому кожен елемент інтерфейсу має заздалегідь визначені розміри та координати розташування. Такий підхід забезпечує мінімальні обчислювальні витрати та спрощує процес рендерингу, оскільки всі необхідні параметри елемента відомі на момент його виклику. Проте ручне компонування є недостатнім для більшості сучасних бібліотек, особливо тих, що орієнтовані на створення користувацьких інтерфейсів загального призначення.

У разі використання адаптивного компонування процес побудови інтерфейсу у системах IMGUI значно ускладнюється. Саме на цьому етапі спостерігаються принципові відмінності у підходах, застосованих авторами різних бібліотек до реалізації цієї функціональності.

Один із поширених підходів передбачає виконання подвійного проходу по коду протягом одного кадру. На першому етапі система здійснює виклик усіх функцій побудови інтерфейсу без урахування дій користувача у поточному кадрі. Наприклад, навіть у разі натискання користувачем на кнопку під час цього проходу, функція DoButton не поверне позитивного результату взаємодії. Головна мета першого проходу — зібрати повну інформацію про всі елементи, які повинні бути відображені, а також про їхні бажані розміри та розташування.

Після першого проходу у системи вже є вся інформація про дерево елементів у цьому кадрі і вона може застосувати функції автоматичного компонування так само, як це робиться з Retained Mode GUI. Можна як написати свій алгоритм розташування елементів, так і застосувати відомі алгоритми вирішення обмежень, такі як Cassowary та Flexbox.

Під час другого проходу система реагує на введення інформації від користувача і повертає змінені значення з функцій. Оскільки система вже визначила розташування і розмір елементів, у неї є можливість правильно інтерпретувати дії користувача з курсором. Рендеринг елементів може відбуватися одразу при виклику функції елементу або в кінці кадру, залежно від потреб програми.

Найбільш відомим прикладом системи з двома проходами за кадр є IMGUI бібліотека Unity, яка використовувалась як для створення інтерфейсу рушія, так і для ігор, які створювалися за допомогою цього рушія [11]. Хоча останні роки Unity переводять рушій на нову RMGUI систему, велика частина вікон досі працює на їх імплементації IMGUI.

Кількість проходів у системах, що реалізують адаптивне компонування в IMGUI, може бути збільшено залежно від архітектурних рішень конкретної бібліотеки або програмного комплексу. Зокрема, у закритому ігровому рушії компанії Keen Games застосовується підхід, який передбачає три проходи за один кадр: окремо виконуються етапи компонування, обробки введення користувача та рендерингу графічного інтерфейсу [12].

Очевидним недоліком такого підходу є більший час виконання програми на процесорі через потребу виконувати однакові дії декілька разів. Другий недолік полягає в тому, що поруч з кодом для UI не можна виконувати певну бізнес-логіку: наприклад, обчислювати значення, які

залежать від частоти кадру. Це може бути однією з причин, чому IMGUI не закріпилась як бібліотека для створення інтерфейсів в Unity. Однак, існують бібліотеки, які і досі є популярними і використовують цей підхід, як-от Nuklear [4].

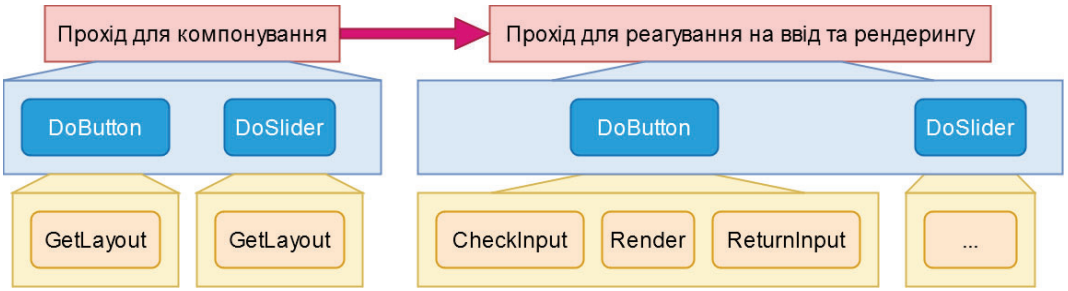

Рис. 1. Схема роботи INGUI з двома проходами

Альтернативним підходом, що з'явився на більш пізньому етапі розвитку бібліотек IMGUI, є метод одноразового проходу за кадр із відтермінуванням обробки введення на один кадр. У цьому випадку виклики функцій побудови елементів інтерфейсу здійснюються лише один раз у межах кожного кадру. Така організація дозволяє системі зібрати повну інформацію про розташування та параметри елементів, після чого виконати їх рендеринг наприкінці поточного кадру. На початку наступного кадру з'являється інформація про введення користувача. Дерево елементів і їх розташування зберігається з попереднього кадру і використовується, щоб визначити, з яким елементом користувач хоче взаємодіяти. При проході по функціям у наступному кадрі передається інформація про введення користувача, який він ініціював у попередньому кадрі. Це може викликати потенційну проблему: що, якщо користувач натиснув на кнопку, а вона пропала у наступному кадрі. У такому випадку взаємодія не відбудеться. На початку кадру програма визначить, що взаємодія хоче відбутися з певною кнопкою. Функція цієї кнопки не викличеться у новому кадрі, а отже бібліотека не може повернути результат взаємодії. В реальній ситуації, користувачі не помічають цієї затримки. У сучасних програмах мінімальна кількість кадрів — 30 кадрів в секунду. Стандартом стає 60 кадрів за секунду і користувачі все частіше хочуть, щоб програми працювали з ще більшою кількістю кадрів за секунду, бо це робить анімації плавнішими. При 30 кадрах за секунду час одного кадру становить приблизно 33 мс, у той час, як час реакції людини є в середньому 250 мс. Це робить затримку непомітною для переважної більшості користувачів. На основі цього підходу були побудовані програми як RADDebugger, а також до розробників не надходило скарг користувачів від затримки введення [10].

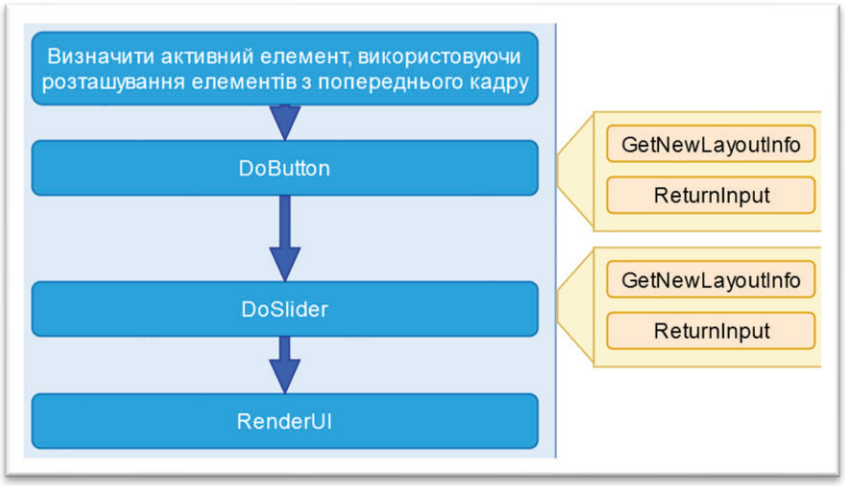

Рис. 2. Схема роботи IMGUI з адаптивним компонуванням і одним проходом за кадр

Ідентифікування елементів

Унікальна ідентифікація елементів є другою принциповою проблемою в архітектурі IMGUI, вирішення якої залежить від конкретних підходів, реалізованих у різних бібліотеках. Необхідність забезпечення унікальної ідентифікації обумовлена потребою зберігати інформацію про те, з яким саме елементом інтерфейсу взаємодіє користувач, упродовж кількох кадрів.

Особливість парадигми IMGUI полягає в тому, що ззовні API має виглядати як система, що не зберігає стан між кадрами. Наприклад, програміст не має змоги напряму отримати через API інформацію про позицію конкретної кнопки або актуальне значення повзунка шляхом звернення до окремого об'єкта. Цей підхід спонукає розробника самостійно управляти станом динамічних елементів, таких як значення повзунків або вибрані опції.

Проте зазначене обмеження стосується виключно зовнішнього інтерфейсу бібліотеки. Водночас усередині самої системи зберігання певного мінімального стану є допустимим і навіть необхідним — зокрема для коректної обробки введення користувача. Головною умовою залишається те, що збережений стан не повинен бути доступним або змінюватися ззовні через API бібліотеки.

Ідентифікування елементів використовується, щоб зберігати інформацію, з яким елементом відбувається взаємодія. Якщо курсор виходить за межі елемента, інформація про нього не зникає і взаємодія продовжується. Зазначимо, що потрібно ідентифікувати тільки елементи, з якими можлива взаємодія. Тексти, зображення та інші статичні елементи можуть перебудовувати повністю кожний кадр.

Перший спосіб ідентифікувати елементи — порядковий зростаючий номер. Коли викликається функція для показу елементу, береться наступне незайняте число і присвоюється елементу. Перевагою є те, що користувач може не передавати жодний параметр у функцію, бібліотека бере відповідальність за присвоєння ідентифікатора на себе. Проблема полягає в тому, що якщо елемент посередині ієрархії ховається, всі елементи за ним змінять свій номер на менший. Якщо один з таких елементів був активним, система помилково визначить активним інший елемент. Вирішенням такої проблеми є виставляння «контрольних точок» - фіксованих номерів, які не змінюються протягом життєвого циклу. Стандартною практикою є виставляння фіксованого номеру кожному вікну, щоб вікна не впливали одне на одне. Також можна виставляти контрольні точки там, де очікується приховування секцій вікна. Це робить цю практику не такою зручною для розробника. До того ж, такий підхід схильний до людської помилки — можна легко забути виставити контрольну точку у потрібному місці і для бібліотеки є складною задачею помітити це і попередити програміста про неправильне користування бібліотекою. Цей підхід використовується Unity IMGUI [13].

Наступний спосіб — хешування даних, переданих користувачем бібліотеки. Часто назва елементу може слугувати унікальним ідентифікатором сама по собі. Якщо кнопці потрібний текст для відображення всередині і кнопка з такою назвою єдина у вікні, то розробнику не потрібно більше нічого передавати у функцію, щоб бібліотека могла унікально ідентифікувати цю кнопку. Бібліотеці достатньо провести хешування тексту і тримати хеш як унікальний ідентифікатор кнопки. Якщо на екрані є одночасно декілька вікон і кожне з них має кнопку з такою назвою, існує можливість додати контекст до вікна. Зазвичай, це функції по типу PushContext(string id) та PopContext(). Ідентифікатор, який переданий у контекст, стає другим параметром у функцію хешування і тільки кнопки всередині одного вікна мають бути унікально ідентифіковані. Якщо все ж таки у вікні існують дві кнопки і обидві потрібно назвати однаково, можна передавати додатковий ідентифікатор, який не буде показаний на екрані, але прийматиме участь у генерації ідентифікатора. Бібліотека DearImgui, яка використовує метод, що описується, робить це наступним чином: "OK##ID1", "OK##ID2" [6]. Текст після ## не показується всередині кнопки. Для програміста може бути незручно передавати ідентифікатор у кожний елемент, з яким можна взаємодіяти. До того ж, не всі елементи мають назви, що показуються на екрані. Наприклад, деякі кнопки можуть містити лише зображення. Задача слідкувати за унікальністю ідентифікатора перекладається на програміста, але, з іншого боку, бібліотека легко може визначити неунікальні ідентифікатори і попередити програміста на етапі розробки. Ще однією перевагою є те, що на основі переданих назв елементів можна побудувати ієрархію інтерфейсу з метою налагодження програмістом, схожу на консоль налагодження у браузері. Коли кожний елемент має зрозумілу людині назву замість цифр, легше знайти елемент, який цікавить і подивитись інформацію про нього.

Ще одним підходом до унікальної ідентифікації елементів у IMGUI є використання інформації про місце виклику функції у програмному коді, зокрема передавання назви файлу та номера рядка виклику. У мовах програмування C та C++ такі дані можуть бути отримані за допомогою макросів FILE та LINE.

На відміну від методу послідовної нумерації, зміна кількості або порядку елементів у такій системі не впливає на сформований ідентифікатор, що забезпечує стабільність його значення між кадрами. Водночас цей підхід усуває потребу покладатися на розробника при генерації ідентифікаторів, оскільки вони формуються автоматично на основі місця виклику.

Однак, зазначений метод має певні обмеження. Зокрема, виникають складнощі у випадках, коли декілька елементів створюються в одному рядку коду, наприклад, у циклі. У таких ситуаціях додатково необхідно передавати параметри, які забезпечують розрізнення окремих елементів, щоб уникнути конфлікту ідентифікаторів. У такому випадку можна передавати додатковий параметр, який дозволить унікально ідентифікувати елементи у циклі, наприклад, порядковий номер. На відміну від першого способу, де користувач має не забувати поставити контрольні точки і бібліотека не може попереджати його, цей спосіб виглядає набагато простішим для слідування людиною і бібліотека може попереджати, якщо це все ж таки станеться. Недоліком цього методу є те, що він підходить тільки для мов програмування, в яких є можливість передати назву файлу та номер рядка. Наприклад, до них не належить C#. Прикладом бібліотеки, що використовує цей метод, є vui [8].

Останнім із відомих підходів до забезпечення коректної взаємодії з елементами інтерфейсу без прямої їх ідентифікації є збереження інформації про сам факт взаємодії користувача, а не про конкретний ідентифікатор елемента. Якщо курсор виходить за межі елемента або елемент переміщується, система перестає вважати його активним. Деякі бібліотеки вважають подібної реалізації достатньо для забезпечення необхідного рівня взаємодії з користувачем і обмежуються саме цим підходом. Зокрема, такий принцип закладено у бібліотеці Nuklear [4].

Подальшим розвитком зазначеної концепції є збереження координати точки на екрані, в якій було здійснено натискання миші. Це дозволяє кожного кадру перевіряти чи триває взаємодія з елементом на основі збігу цієї координати з поточним положенням елемента. Такий підхід дає змогу підтримувати взаємодію навіть у разі зміни структури інтерфейсу. Водночас його ефективність знижується у ситуаціях, коли елементи інтерфейсу зазнають руху, зокрема під час приховування частини вікна, анімації елементів або перетягування вузлів у графових структурах. Прикладом реалізації даного підходу є бібліотека Lobster [3].

Висновки

За результатами проведеного аналізу встановлено, що ключовими факторами, які забезпечують зручність та високу продуктивність існуючих бібліотек Immediate Mode GUI (IMGUI), є ефективна організація адаптивного компонування та надійна система ідентифікації елементів. Аналіз показав, що використання підходу одного проходу за кадр із затримкою обробки введення є найбільш перспективним для створення IMGUI-бібліотеки у середовищі Unity. Такий підхід дозволяє зменшити навантаження на процесор та уникнути дублювання коду.

З урахуванням особливостей архітектури ECS (Entity Component System), яка активно впроваджується в Unity, саме парадигма IMGUI є більш придатною для створення внутрішньо-програмних інтерфейсів у таких умовах. Вона дозволяє уникнути проблем синхронізації між станом інтерфейсу та змінами об'єктів, характерних для RMGUI.

Щодо ідентифікації елементів, найбільш доцільним для реалізації в Unity та мові програмування C# є підхід, заснований на хешуванні текстових міток. Він забезпечує баланс між зручністю використання бібліотеки та коректною роботою з динамічними елементами інтерфейсу без потреби у складних механізмах контролю з боку розробника.

Отримані результати створюють передумови для подальшого розвитку IMGUI-бібліотек у контексті сучасних вимог до продуктивності та гнучкості інтерфейсів користувачів. Застосування описаних підходів дозволить спростити розробку внутрішньо-програмних інтерфейсів у Unity, зробити їх більш інтегрованими в архітектуру ECS та зручними для розробника. Це

відкриває перспективи створення універсального інструменту для побудови повноцінних динамічних інтерфейсів у сучасних ігрових та інтерактивних застосунках.

Список літератури

1. Брендель Ф., Лідтке С. Exploring the Immediate Mode GUI Concept for Graphical User Interfaces in Mixed Reality Applications. GI VR / AR Workshop 2022, м. Гамбург, 14–16 верес. 2022 р. Гамбург, 2022. URL: https://dl.gi.de/server/api/core/bitstreams/b8c94eda-1405-4b23-bc9f-e184cad04602/content (дата звернення: 11.12.2024).

2. Ван Бернем С. Л. Nbui : магістерська робота. Аахен, 2019.

3. Ван Оортмерссен В. lobster [Електронний ресурс] / Воутер ван Оортмерссен // github.com. – Режим доступу: https://github.com/aardappel/lobster (дата звернення: 26.11.2024).

4. Меттке М. Nuklear [Електронний ресурс] / Міша Меттке // github.com. – Режим доступу: https://github.com/vurtun/nuklear (дата звернення: 26.11.2024).

5. Мураторі К. Immediate-Mode Graphical User Interfaces - 2005 [Електронний ресурс], 2016 / Кейсі Мураторі // youtube.com. – Режим доступу: https://youtu.be/Z1qyvQsjK5Y (дата звернення: 26.11.2024).

6. Окорню О. FAQ (Frequently Asked Questions) [Електронний ресурс] / Омар Окорню // github.com. – Режим доступу: https://github.com/ocornut/imgui/blob/master/docs/FAQ.md (дата звернення: 26.11.2024).

7. Радіч К. Retained Mode Versus Immediate Mode [Електронний ресурс] / Квінн Радіч, Майкл Сатран // learn.microsoft.com. – Режим доступу: https://learn.microsoft.com/en-us/windows/win32/learnwin32/retained-mode-versus-immediate-mode (дата звернення: 26.11.2024).

8. Роуз Г. vui [Електронний ресурс] / Генрі Роуз // github.com. – Режим доступу: https://github.com/heroseh/vui (дата звернення: 26.11.2024).

9. Фельдмаєр А. GUI Programming [Електронний ресурс] / Алекс Фельдмаєр // people.uwplatt.edu. – Режим доступу: http://people.uwplatt.edu/~yangq/CSSE411/csse411-materials/f13/feldmeiera-gui%20_programming.doc (дата звернення: 26.11.2024).

10. Фльорі Р. UI, Part 2: Every Single Frame (IMGUI) [Електронний ресурс] / Райан Фльорі // www.rfleury.com. – Режим доступу: https://www.rfleury.com/p/ui-part-2-build-it-every-frame-immediate (дата звернення: 26.11.2024).

11. Immediate Mode GUI (IMGUI) [Електронний ресурс] // docs.unity3d.com. – Режим доступу: https://docs.unity3d.com/Manual/GUIScriptingGuide.html (дата звернення: 26.11.2024).

12. Micha Mettke [Електронний ресурс] : випуск 3 / ведуч. Р. Фльорі ; гість М. Меттке // The Handmade Network Podcast. – Режим доступу: https://handmade.network/podcast/ep/c1174949-adc4-492d-89b5-ca73dea4ff16 (дата звернення: 26.11.2024).

13. MonoBehaviour.OnGUI() [Електронний ресурс] // docs.unity3d.com. – Режим доступу: https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnGUI.html (дата звернення: 26.11.2024).

A. Perepelytsia, O. Dibrivnyi

ANALYSIS OF APPROACHES TO ADAPTIVE LAYOUT AND ELEMENT IDENTIFIERS IN IMMEDIATE MODE GUI FOR ITS USE IN UNITY

Immediate Mode GUI (IMGUI) is widely used for creating graphical user interfaces in programming. Many modern libraries implement this paradigm to enable dynamic, real-time rendering of interface elements. The use of IMGUI allows for efficient redrawing of interface components every frame and direct handling of user interactions through function calls.

However, IMGUI presents challenges related to the adaptive layout of elements and their identification between frames. This paper examines two key issues of IMGUI: adaptive layout and element identification. Various approaches to solving the adaptive layout problem are analyzed, including

methods that require re-execution of code or introduce a one-frame delay in program response. It is determined that the delayed input approach is more promising.

The study also explores mechanisms for identifying interface elements between frames, including automatic and developer-defined generation of identifiers. Based on the analysis, it is concluded that in the Unity environment, the most optimal method for element identification is hashing text labels, which ensures a balance between implementation convenience and system performance.

The results obtained provide a foundation for the further development of IMGUI libraries in the context of modern requirements for the performance and flexibility of user interfaces. The application of the proposed approaches will simplify the development of in-application interfaces in Unity, making them more integrated into the ECS architecture and more convenient for developers. This opens up prospects for creating a universal tool for building fully functional dynamic interfaces in modern gaming and interactive applications.

Keywords: IMGUI, GUI, adaptive layout, library, paradigm, architecture, approach, identification.