Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
(заочнодистанційного навчання) Форма навчання денна (заочна) Кафедра
комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХІВСЬКА
(підпис)
«___»_____202_р

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА РЕАЛІСТИЧНОГО ІГРОВОГО СИМУЛЯТОРА КАСОВОГО АПАРАТУ»

зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр

Виконавець роботи Голишко Єгор Олександрович
_____ «___»_____202_р
(підпис)
Науковий керівник Ольховська О.В., зав. каф., к.ф.-м.н _____
_____ «__»_____202_р
(підпис)

Рецензент

ПОЛТАВА 2025

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК

ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка реалістичного ігрового симулятора касового апарату»

зі спеціальності 122 Комп'ютерні науки»

освітня програма «Комп'ютерні науки»

ступеня бакалавр

Прізвище, ім'я, по батькові Голишко Єгор Олександрович

Затверджена наказом ректора № ____-Н від « » _____ 202_ р.

Термін подання студентом роботи «__ » _____ 2025 р.

Вихідні дані до кваліфікаційно роботи: публікації з теми. Курсовий проєкт з фаху на тему, проєктування дизайну та інтерфейсу для реалістичного симулятора касового апарату».

Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ЗМІСТ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ, ТЕРМІНІВ.

Перелік графічного матеріалу: 3-4 аркуші графічного матеріалу, інші необхідні ілюстрації.

Консультанти розділів кваліфікаційної роботи.

Розділ	ПІП, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постанова задачі			
Інформаційний огляд			
Теоретична частина			
Практична реалізація			

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3. Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6. Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доробка (за необхідністю), рецензування		

Дата видачі завдання « __» _____ 2025 р.

Здобувач вищої освіти Голишко Єгор Олександрович

Науковий керівник _____.

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____

(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.

Секретар ЕК_____ _____

(підпис) (ініціал та прізвище)

Затверджую Погоджено

Зав. кафедрою _____ Науковий керівник_____

к.ф.-м.н. Олена ОЛЬХОВСЬКА _____

«__» _____ 2025 р. «__» _____ 2025 р.

Завідувачу кафедри КНІТ
Олені ОЛЬХОВСЬКІЙ
здобувача вищої освіти 4 курсу, групи КН 41 б
освітньо-професійної програми «Комп'ютерні науки»
зі спеціальності 122 «Комп'ютерні науки»
Навчально-наукового інституту денної освіти (заочно-дистанційного навчання)
Мельниченка Ігоря Олександровича
моб. тел. 0507742578

ЗАЯВА

Прошу дозволити мені виконувати кваліфікаційну роботу на тему „ Розробка реалістичного ігрового симулятора касового апарату “

Здобувач вищої освіти _____ Єгор ГОЛИШКО

Науковий керівник _____

РЕФЕРАТ

Записка: … 60с., … 10 рис., … 1 таблиця, … 30 джерел.

КАСОВИЙ СИМУЛЯТОР, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, АДАПТИВНИЙ ДИЗАЙН, UNITY.

. **Об’єкт розробки** — програмний додаток-комплекс «Реалістичний симулятор касового апарату», розроблений використовуючи ігровий рушій Unity , який буде симулювати роботу двох касових систем , а саме Shoop Keep та MINI-T400 , та відтворювання основного функціоналу цих апаратів . Також опрацювання основних етапів створення UI , робота з динамічним масштабуванням інтерфейсу , адаптування дизайну під різні розміри дисплею .Робота з опрацюванням та аналізом справжніх систем , розробка state-based систем.

Мета роботи – розробити симулятор двох касових систем MINI T-400 та Shoop Keep , який буде за дизайном та особливостями використання схожий до реальних умов а також оглянути можливості адаптації цього симулятора під різні платформи зі збереженням коректного розташування елементів UI для підтримки адаптивності застосунку. Розробити інтерактивні підказки , щоб навчання роботи з цими системами проходило якомога простіше. Використати State-based підхід для регулювання сторінок касового апарату та їх коректної роботи.

Методи дослідження – В процесі розробки було застосовано такі методи:

Теоритичні методи : було здійснено аналіз наявних систем у галузі симуляторів для подібних систем , також було проведено огляд необхідної технічної літератури , яка б дала краще розуміння в якому напрямку я маю рухатись на початку розробки.

Емпіричні методи : практична перевірка розробленого проекту в середовищі розробки UNITY , тестування коректної роботи бази даних з усіма необхідними системами , аналіз ефективності state-based менеджменту у регулюванні станами симулятору.

ЗМІСТ

ВСТУП

На сьогоднішній день ми чітко бачимо як бізнес та технології стрімко йдуть вперед та найм більшої кількості співробітників для опрацювання касових-фінансових операції стає все більше необхідним , незважаючи на те , що процес навчання не є складним , але все ж підготовка нових працівників займає дорогоцінний для великих та малих компаній час. В цій дипломній роботі було поставлено задачу створити навчальний додаток-симулятор для роботи з деякими видами касових апаратів , щоб навчання робочого персоналу було скорішим , зручнішим та безпечнішим для всього бізнесового сектору в нашій країні та за кордоном .Тому випливає висновок ,що впровадження подібного тренувального додатку для оптимізації притоку робочогу персоналу є не тільки необхідним , а й розумним у фінансовому розумінні рішенням.

Для створення цього додатку було оглянуто багато різних “рушіїв” та середовищ розробки і їх мов програмування , але з-поміж всіх найяскравіше виділився **Unity.** Далі в роботі буде обґрунтовано чому було обрано саме Unity , а не будь – яке інше середовище розробки .Під час розробки цього додатку вдалося опрацювати та дослідити всі основні етапи створення симуляторів : налаштування та адаптація UI , написання коду , робота з звуковими файлами та ефектами . Такод вдалося розробити state-based систему ефективного контролю над “сторнінками” та станами кожної з двох обраних систем , а саме T-400 MINI , українська система ,яка зазвичай використовується на автовокзалах та магазинах та американська POS система Shoop-Keep. Було досліджено основні етапи роботи з цими касовими системами та створено приближені до реалності тренувальні операції з якими може працювати потенційний працівник.

Неможна не оцінити наскільки серйозний вплив оптимізація та підвищення кваліфікації працівників дають на удосконалення витрат на ті чи інші елементи компаній , створення подібного симулятору дає можливість інтегрувати ці всі процеси в загальний робочий вокрфлоу.

Під час виконання роботи використовувавсь науковий підхід та орієнтуватися на комплес методів наукової літератури , також використовувась порівняльний аналіз

існуючих рішень , системне моделювання та експериментальну перевірку. Впроваджував використання state-based логіки для найзручнішого регулювання операціями касових систем . Також було використано систему для роботи з базами даних , а саме Excel та формат файлу .csv для обліку наявних тестових товарів для симулятору з якими може працювати користувач .Було оглянуто реальний UI та створено аналог інтерфейсу , щоб збільшити рентабельність та реалістичність до підходу навчання.

Розглянувши наявні системи касових апаратів, було обрано по одній з популярних POS-систем в Україні та в США — зокрема, системи MINI T-400 та ShopKeep. Такий вибір було зроблено для того, щоб потенційний додаток міг розвиватися не лише в Україні, а й за кордоном, тим самим підвищуючи зацікавленість потенційних користувачів. Також це дає змогу оцінити різні підходи до обробки касових операцій у різних системах та країнах.

Оцінивши все перелічене вище, було усвідомлено, що робота над цим додатком стане цікавим та захопливим процесом вивчення функціонування касових систем. Також я зрозумів, що подібний додаток є необхідністю в теперішніх кризових умовах на світовому ринку праці, а ще він дозволить оптимізувати деякі ключові процеси виробництва та підготовки персоналу. Уже очевидно, що для покращення досвіду потенційних користувачів має бути враховано кросплатформенність майбутнього продукту, тому необхідно подбати про адаптацію інтерфейсу до різних розмірів екранів, тощо.

Тепер, коли всі основні питання стали зрозумілими, настав час визначити ключові етапи розробки технологічного продукту. Розглянувши основні касові операції на реальних системах, можна дійти висновку, що під час роботи важливо зберігати баланс між технологічною складністю та простотою у використанні. Створення подібних реалістичних симуляторів є серйозним кроком до підвищення продуктивності підприємств. Важливо також оцінити, що саме необхідно для практичної реалізації, які підводні камені можуть виникнути під час розробки, а також порівняти Unity з популярними альтернативами для обґрунтування вибору цього рушія. Під час створення цього програмного додатку було поставлено за мету

навчитись практичним навикам розробки сучасних інноваційних додатків , дослідити майбутні необхідні кроки для впровадження цієї системи у потенційне підприємство. За результатами роботи буде отримано додаток , що спростить навчання та роботу для багатьох людей .Тож давайте окреслимо основні етапи та методии , які буде необхідно пройти для розробки цього продукту.

1. Визначити мету та завдання розробки тренувального симулятора касових апаратів.
2. Окреслити об’єкт та предмет дослідження.
3. Провести огляд сучасних симуляторів касових апаратів та інструментів для їх розробки.
4. Розглянути психолого-педагогічні аспекти використання симуляторів у навчанні.
5. Обґрунтувати вибір Unity як основного середовища розробки.
6. Описати використання Microsoft Visual Studio для написання коду та мови програмування C#.
7. Визначити роль Excel для обробки баз даних та створення CSV таблиць.
8. Створити інтуїтивно зрозумілий та адаптивний дизайн UI для симулятора.
9. Впровадити інтерактивні фідбеки для кнопок (аудіо та візуальні ефекти, підсвічування при наведенні).
10. Реалізувати state-based управління для моделювання різних режимів роботи касових апаратів (Т-400 та ShoopKeep beta).
11. Інтегрувати функціонал роботи з базами даних (.csv).
12. Створити компонент SFX Manager для централізованого управління звуковими ефектами через Audio Source.
13. Провести юзабіліті-тестування симулятора на різних пристроях.
14. Провести оптимізацію системи, враховуючи продуктивність та стабільність роботи.
15. Дослідити вплив інтерактивного симулятора на процес навчання та мотивацію користувачів.
16. Провести аналіз отриманих результатів з погляду педагогічних аспектів.

17. Підготувати звіт з описом розробки, методології дослідження, отриманих результатів та висновків.

18. Оформити результати роботи відповідно до вимог до бакалаврських робіт.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Ідея розробки реалістичного симулятора касового апарату полягає в тому , щоб створити динамічне тренувальне середовище в якому нові потенційні співробітники , або персонал , який бажає підвищити свою кваліфікацію , могли без труднощів підготувати себе до нового інструменту роботи . Основний акцент програми покладено на представлення якомога реалістичного симулювання роботи реальних систем , а саме касових апаратів Т-400 та ShoopKeep . Завдяки цьому додатку ознайомлення з базовим функціоналом для потенційних користувачів буде простим і дасть розуміння основних операцій, які необхідні в робочому процесі, виключаючи будь-які втрати для виробництва.

1.1 **Основні аспекти проєктування:**

Уявіть програму, завдяки якій у вас з’явиться повноцінний тренувальний майданчик, де ви зможете проаналізувати та опрацювати кожну операцію, доступну у системах Т-400 і ShopKeep.Для створення такої симуляції нам важливо проаналізувати та зрозуміти всі аспекти й складові, які необхідні для навчання роботі, а також врахувати всі нюанси, щоб практичний досвід був максимально наближеним до реального.

Саме такі етапи роботи ми ставимо собі за мету. Ми всі знаємо, яким іноді складним буває навчання, тож ідея створення спрощеного тренувального симулятора має чітку та зрозумілу мету.

Основна концепція цього програмного продукту полягатиме в тому , щоб створити безпечне середовище роботи для навчання початківців та досвідчених користувачів роботі нових та незрозумілих для них касових систем.Адже виконання роботи без практики , навіть якщо у вас вже є досвід роботи зі схожими системами , може бути вкрай складним , почуття невпевненості та розгубленості може принести збитки підприємству та порушувати локальні ланцюжки роботи , що може принести дискомфорт у колективі. Саме тому інтерактивні тренування , де кожну операцію

можна спробувати опрацювати власноруч , допоможуть підготувати нових та готових до роботи спеціальістів.

Цей проєкт має бути не просто віртуальним тренажером , а й відтворювати реалістичну послідовність операцій під час умовної роботи на справжній системі. Невід'ємним є і те , щоб були наявні основні типові операції , такі як: облік товарів , застосування спеціальних знижок , вибір способу оплати , оформлення чеку. Чи замислювались ви , скільки насправді є нюансів , які має врахувати касир під час опрацьовуючи операції? Саме ці нюанси і стануть основним аспектом розробки системи. З огляду на те , що навіть у здавалось би простих задачах можуть критись дуже багато підводних каменів , що також може бути сказано і про роботу з касовими системами.

1.2 Інтеграція комплексних функцій та технологічних рішень

Також важливим аспектом є користувацький інтерфейс. Важливо, щоб його вигляд напряму передавав те, як виглядає реальна система, з якою, можливо, в майбутньому буде працювати користувач, передаючи як візуальний контекст, так і практичний. Також важливо продумати, як інтерфейс буде змінюватися відповідно до розміру екрану користувача. Це вкрай необхідно для майбутньої реалізації кросплатформенності, та й загалом адаптивність покращить користувацький досвід на будь-якій платформі.

Незважаючи на те, що зовнішньому вигляду застосунку було приділено значну увагу, основним у ньому, все ж, залишається програмний функціонал. Симулятор підтримує досить широкий спектр операцій, які може відтворювати, що дозволяє опанувати як базові дії, так і глибші аспекти систем. Зважаючи на те, що головною ідеєю продукту є навчання роботі з певними системами, я прагнутиму відтворити функціонал настільки реалістично, наскільки дозволяють мої аналітичні навички.

- Автоматизацію процесів продажу та оформлення операцій.
- Роботу з базами даних для збереження та обробки інформації про товари.

- State-based управління, що дозволяє точно відтворити логіку переходів між робочими режимами апарату.
- Інтерактивні підказки, що сприяють навчанню користувачів та полегшують освоєння системи.
- Панель товарів для додавання позицій у чек.
- Поле для введення кількості товару.
- Пошук товарів з бази даних у форматі .csv
- Отримання чеку після оформлення покупки
- Сторінка перегляду всіх наявних товарів в базі даних
- Додавання товару у кошик , вибір кількості доданого товару
- Додавання скидки до товари за потребою
- Поле для вибору способу оплати.(готівка/карта)
- Кнопка підтвердження транзакції.

Розробка симулятора касових апаратів Т-400 та ShoopKeep beta має на меті не лише технічне відтворення функціоналу, але й створення навчального майданчика для підготовки майбутніх спеціалістів. Реалізація проєкту дозволить забезпечити комплексний підхід до тренування персоналу, що в подальшому сприятиме підвищенню ефективності роботи закладів торгівлі та обслуговування

1.3 Вибір основних інструментів для створення застосунку

Основною задачею цієї дипломної роботи є створення інтерактивного додаток , тому у цьому розділі буде оглянуто наявні засоби для створення програмного забезпечення та зробити намітки того , що нам необхідно зробити в рамках плану розробки, також донесу свою думку , що-до вибору тих чи інших середовищ розробки. Базою для створення додатку заплановано обрати систему Unity , що також тягне за собою використання мови програмування C# , для обробки коду - середовище Microsoft Visual Studio , для обробки баз даних та створення таблиць було використано Microsoft Excel , щоб створювати .csv таблиці підтримувані нашим обраним двигуном для розробки.

Один з запропонованих рушіїв розробки є Unity. Unity – це популярне середовище для розробки комп'ютерних ігор , також може бути використаний для створення реалістичних двовимірних , або тривимірних научних симуляцій та багато чого іншого .

Завдяки популярності у Unity є величезна спільнота розробників , яка дає змогу отримати практичні та теоритичні приклади для релізації тих чи інших ідей. Unity дає величезний інструментарій для створення інтерфейсів та реалізацію практичних систем, що є ключовою причиною чому було обрано саме його.

Ще одною причиною через яку заплановано обрати Unity є те , що інтерфейс цієї системи є зручним та інтуїтивно зрозумілим , що дозволяє швидко прототипувати ідеї. Для більшості інших рушіїв інформація досягається лише з технічної документації , але завдяки великій спільноті розробників є змога опанувати та вивчити основні аспекти роботи набагато простіше, що дозволяє ефективно втілювати ідеї в життя.

Тож, з огляду на те, що може запропонувати Unity, зрозуміло, що це, якщо не найкраще, то найближче до ідеалу середовище розробки. Величезна спільнота розробників, зрозумілий інтерфейс, можливість адаптації застосунків під різні платформи, швидке прототипування та багато іншого. Цього більш ніж достатньо для реалізації запропонованої ідеї тренувального симулятора, але все ж потрібно проаналізувати переваги та недоліки й розглянути інші рушії для розробки.

Отже, перед нами стоїть непроста задача: вирішити, яке середовище розробки ми будемо використовувати. Нам потрібно проаналізувати всі наявні аналоги Unity та з'ясувати , чи справді Unity є найкращим вибором для створення подібних симуляторів.

Розробка застосунку передбачає використання мови програмування. Зручним для себе IDE я обрав Microsoft Visual Studio — доволі зручний інструмент із власними перевагами та недоліками, про які розповім далі. Однією з головних переваг серед інших IDE є можливість встановлювати доповнення, що спрощують редагування й

написання коду. Одне з таких — IntelliSense, яке дозволяє середовищу передбачати ваші наміри та підказувати методи й властивості, які ви хочете набрати.

Незалежно від обраного в майбутньому середовища розробки, заплановано використовувати саме це рішення, адже воно підтримує будь-яку мову програмування.

Візьмемо для прикладу ефективність IDE на модулі Unity, який відповідає за обчислення фізики об'єктів: клас PhysicsEngine та інтерфейс ICollisionDetector, що залежить від сторонньої бібліотеки MathNet.Numerics. Лише завдяки гнучкості C# усе це легко поєднується. Хіба не чудово, що можна без зайвих зусиль додавати пакети та відразу викликати їхні функції?

Ключ до успішної побудови продукту стоїть у правильній організації робочого процесу, тому було проаналізовано необхідні задачі та створено базові UML-діаграми, для того щоб не було непорозумінь у власних цілях під час самої розробки, саме це також варто зробити під час подальшого планування розробки. Завдяки такому підходу, проект буде триматися в чистоті та порядку.

Також під час створення цього програмного продукту , в будь-якому випадку доведеться працювати з базами даних, так як створення симулятора касового апарату передбачає облік товарів, цін, скидок і т.д. Тому одне з питань, яке доведеться вирішити — це як ми будемо обробляти ці дані. Найяскравішим варіантом, поки що, для мене є Excel, так як це одне з найпопулярніших рішень у своєму роді, що дозволяє опрацьовувати дані в багатьох форматах, що може знадобитись, так як рушії для розробки додатку можуть потребувати незвичайні формати, такі як .xls.

Варто відзначити і кілька нюансів:

Переваги: простота створення Pivot-зведень для агрегації даних, швидкий доступ до Power Pivot для побудови складних моделей, можливість використовувати DAX-функції для аналітики.

Недоліки: великі об'єми (понад 100 000 рядків) можуть сповільнювати роботу, а надмірні залежності від макросів ускладнюють підтримку.

Отже, можна стверджувати , що саме баланс між гнучкістю Excel та строгістю дозволить уникнути зайвих витрат часу на налаштування імпорту в рушій для розробки. А якщо виникне потреба у швидких правках — наприклад, зміна цін чи

оновлення категорій —буде використаний Excel , що дасть змогу вносити корективи одним кліком та генерувати оновлений CSV.

1.4 Основні етапи розробки

В першу чергу , для створення детальної симуляції також варто дослідити всі основні характеристики касових апаратів та як вони можуть використовуватись у реальній роботі . Під час дослідження функціональних можливостей та специфікацій цих пристроїв було проаналізовано основні операції , а саме обробку транзакцій, ведення товарного обліку, застосування знижок та обчислення фінансових показників.

Зокрема , важливо усвідомити , як кожна з моделей обробляє ті чи інші операції , як касова система опрацьовує друк чеків та співпрацює з прикріпленою базою даних. Це має допомогти в розумінні того , як правильно спланувати етапи розробки логіки симулятору. Важливо , щоб у симуляторі також збереглися всі основні деталі та нюанси які є на справжніх системах , так як це і є головною ціллю застосунку.

Автор вважає , що такий підхід дозволить не лише забезпечити відповідність симулятора вимогам сучасних бізнес-процесів, а й створить умови для ефективного навчання нових співробітників, які зможуть на практиці опанувати основні операції і зменшити ризик помилок під час роботи з реальним обладнанням.

Після аналізу усіх головних операцій та процесів до наших обраних касових систем , що формуватимуть базовий функціонал , треба оглянути які елементи додатку будуть відповідати за навчальний аспект . Також варто дослідити , як найзручніше передати користувачеві навчальний матеріал .

Особливу увагу має бути приділено механізму застосування знижок – було розролено сценарій, який дозволяє користувачу легко вибирати та застосовувати знижки до окремих товарів , або до всього чеку, що є надзвичайно важливим для оптимізації робочих процесів.

Ще один важливий аспект – вибір способу оплати. Треба створити сценарій, де користувач може обирати між різними методами оплати, такими як готівка чи банківська карта. Це дозволяє симулятору відображати реальні умови роботи, де оператор повинен знати, як працювати з різними типами транзакцій.

Не менш важливою є робота з базою даних товарів. Треба також оглянути як база даних взаємодіятиме з системами симулятору , розробити основний функціонал , а саме : пошук товарів , видалення товарів , збереження списку обраних товарів та накладення на них скидок.

Таким чином, комплексне опрацювання цих сценаріїв дозволить користувачам симулятора отримати реалістичний досвід роботи з касовими апаратами, що сприятиме їхній швидкій адаптації до реальних умов роботи та зниженню кількості помилок під час експлуатації обладнання.

Також важлиивим аспектом є практична сторона проекту. Важливо щоб продукт зберігав чисту структуру коду та був зрозумілим для аналізу. Тож за ідею заплановано взати state-based підхід , або простіше кажучи, реалізація скінченних автоматів (finite state machines), де кожен стан — від «Ініціалізація» до «Фіналізації транзакції» — чітко описаний і легко переключається. Це нагадує систему світлофорів на перехресті: якщо щось піде не так із логікою перемикань, потік «транзакцій» просто зупиниться!

З оглядку на вище зазначені данні випливає , що чітке розділення на стани (Idle, ScanItem, ApplyDiscount, PrintReceipt тощо) дозволить уникнути хаосу в бізнес-логіці. Кожен перехід обробляється за допомогою event-driven архітектури: натиснув кнопку — спрацював івент, змінився стан. Це не просто абстракція — це гарантія того, що симулятор буде поводитись так само, як реальний POS-термінал, навіть якщо йдеться про екстрену ситуацію (наприклад, збій живлення під час друку чека).

Розробляючи логіку касира , було створено набір сценаріїв (use cases), що відтворюють:

Оформлення покупки (CRUD-операції з товарною базою)

Застосування знижок (promotion engine із підтримкою складних правил)

Повернення товару (reverse transactions із логом audit trail)

Цей підхід базується на принципах Domain-Driven Design: кожен агрегат (наприклад, SaleTransaction) відповідає за свій контекст, що полегшує масштабування та підтримку коду.

У результаті буде відтворено систему, яка не лише демонструє архітектурні патерни (State Machine, Event Sourcing, ETL), а й максимально наближена до

щоденних операцій касира. Завдяки такому комплексному підходу користувачі симулятора отримують живий досвід роботи з POS-терміналом, а навчання стає швидшим і безпечнішим.

Після того, як буде закінчено проєктування, обов'язково треба провести тестування застосунку з різними розмірами екрана, адже я впевнений, що лише через ретельну перевірку можна забезпечити бездоганну роботу симулятора в реальних умовах. Обов'язково треба провести серію тестів, використовуючи пристрої з різними розмірами екранів – від компактних мобільних телефонів до великих моніторів, щоб оцінити ефективність роботи адаптивних механізмів. Це дозволить провести оцінку флюїдності додатку та його готовність до переносу на інші платформи. Тестування має включати перевірку масштабування, адаптацію розміщення елементів у співвідношенні одне з одним. Тож при виборі двигуна для розробки треба також взяти до уваги те, що у нас є необхідність подібного функціоналу.

Одне з фундаментальних тверджень у галузі тестування програмного забезпечення сформульоване безпосередньо в роботі Канера:

«Тестування - це процес виконання програми з метою пошуку помилок». [22]
Таким чином, сутність тестування полягає не в доведенні відсутності помилок, а саме в активному пошуку і виявленні дефектів у програмному продукті. Це означає, що тестувальник має створювати такі умови й сценарії, які максимально підвищують шанси "зловити" помилку в роботі системи.

На завершальному етапі розробки відбувається впровадження інтерактивних елементів у середовище Unity. Це включає додавання функцій для взаємодії користувача з інтерфейсом: додавання товарів у чек, вибір способу оплати, UI для майбутніх операцій взаємодії між користувачем та інтерфейсом.

Також обов'язково необхідно дослідити навчальні сценарії та впровадити інтерактивні підказки, щоб реалізувати свого роду тренувальний аспект у симулятор.

РОЗДІЛ 2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Огляд наявних програм фінансового обліку

Для побудови якісної симуляції важливо дослідити та проаналізувати аналоги до обраних систем для проєктування. У сучасних програмах використовується величезна кількість аспектів, які забезпечують комфортний та ефективний робочий досвід. Це дасть змогу поглибитися у розуміння того, як працюють технології в цих системах, які буде необхідно впровадити в реалізовану програму симулятора.

Як приклад, я візьму системи фінансового обліку, такі як QuickBooks, SAP або Oracle Financials. Вони застосовують модульну структуру, яка дозволяє розділити функціональні блоки (бухгалтерський облік, управління фінансами тощо). Завдяки побудові такого типу люди, які користуються цими системами, мають змогу виконувати необхідні операції за мінімально можливу кількість кроків. Також це знижує час, необхідний для адаптації.

Сучасні касові системи підтримують адаптацію до використання у різних умовах залежно від обраного комп’ютера чи розміру екрана. Це особливо актуально для нових касових систем, які вже мають можливості для використання навіть через мобільний застосунок.

Під час аналізу важливо звернути увагу на те, як елементи інтерфейсу поєднані одне з одним та як саме вони створюють синергію. Також варто оглянути, як інтерфейс взаємодіє з користувачем та як працюють усі базові операції, такі як пошук товарів, транзакцій, автозаповнення даних тощо. Це вкрай необхідно для створення реалістичного користувацького досвіду для мого продукту.

Також варто звернути увагу на те, що в сучасних касових системах є можливості для підвищення продуктивності під час роботи завдяки персоналізації необхідних елементів інтерфейсу під потреби користувача. Завдяки подібному підходу підвищується не лише позитивний досвід роботи для конкретної людини, а й загальна ефективність під час взаємодії з такими системами.

Оскільки ми маємо справу з програмами для фінансового обліку, потреба в захисті даних у подібних системах є особливо важливою. Сучасні касові системи

застосовують цілу низку багаторівневих механізмів захисту, таких як аутентифікація та шифрування даних. Ці функції вкрай необхідні, адже робота з обліком фінансів для будь-якого підприємства є надважливою.

Аналіз відповідних рішень для касових систем дає розуміння того, що без інтеграції сучасних інструментів для персоналізації та адаптації вже не обійтися. Адаптивний дизайн та персоналізація користувацького досвіду вже закоренилися в індустрії як необхідний стандарт. Отриману інформацію обов'язково треба взяти до уваги та використати під час створення симулятора.

2.2 Обґрунтування вибору ShopKeep та MINI T400

Голвною метою було створення правдоподібного тренажера-симулятора для касових систем, які широко використовуються як в Україні, так і закордоном. Оглянувши список систем, що є фаворитами, було зазначено, що американська система ShopKeep та українська MINI T400 ME є найкращим вибором для мого застосунку. Далі буде обґрунтвано, чому були обрані саме ці системи.

Рисунок 2.1 – Зображення обраних систем для симулятору в реальному житті

Проаналізувавши наявні аналоги, стало зрозуміло, що інтерфейси ShopKeep та MINI T400 найбільше підходять як зразок інтуїтивності та адаптивності. Порівнюючи ці системи з іншими, можна сказати, що інтерфейси даних продуктів зрозумілі з першого погляду. Жодних елементів, які перевантажують UI немає: мінімалістичний та зрозумілий дизайн. Проте, не дивлячись на зручність інтерфейсу в обох системах, порядок дій під час виконання окремих операцій не завжди є очевидним.

MINI T400 та ShopKeep також обрано тому, що вони включають усі базові операції, необхідні при роботі з будь-якою POS-системою. Завдяки тому, що ці системи мають основний набір необхідних операцій, якщо користувач навчиться працювати з ними, у майбутньому адаптація до подібних систем не викличе проблем, оскільки інші системи можуть мати аналогічний функціонал.

Ці моделі мають «золоту середину» між простотою та функціональністю — достатньо складні, щоб навчити всіх нюансів, але не настільки, щоб ускладнювати процес освоєння. Поєднання ShopKeep і MINI T400 ME дозволяє охопити весь цикл операцій: від швидкого введення товарів через штрихкод-сканер до коригування знижок і друку фіскального чека.

Використання поєднання цих двох систем для створення нашого симулятора дає змогу охопити найважливіший цикл операцій при використанні POS-систем. У цих системах є такі операції: коригування знижок, пошук необхідних товарів, покупка товарів, реєстрація касира тощо, тому це майже ідеальний варіант для створення навчального середовища. Давайте розглянемо основні переваги використання симулятора для цих систем:

Use case сценарії: оформлення продажу, оформлення чеку, обробка повернень, робота з акціями та бонусами;

Audit Trail: кожна дія записується в лог, що дає змогу аналізувати помилки та вдосконалювати навички;

Role playing: завдяки цьому додатку стажер зможе відчути себе справжнім касиром у «бойових» умовах без ризику втратити реальні кошти.

2.3 Вибір середовища для розробки

Для створення будь-якого додатку треба середовище розробки , нам важливо оцінити перваги найпопулярніших з них, та обрати найкращий варіант , який би відповідав всім вимогам , які необхідні для створення додатку-симулятора. Найяскравішими в цьому полі є такі системи як: Unreal Engine , Godot , Unity , WPF (.NET) , Phaser (HTML5/JS). В подальшому огляді важливо оцінити переваги кожного з них , та обрати який найбільше підходить.

У кожної з цих систем абсолютно свій підхід до універсальності та крос-платформеності , Unreal Engine одразу ж дає можливість до експорту проекту під основні нині існуючі платформи , але якщо порівнювати цей двигун з Unity , то він одразу ж сильно програє в перегруженості та запуск навіть простих додатків може потребувати немалих ресурсів , на противагу в Unity пустий проект може працювати навіть на вкрай простому залізі. . WPF (.NET) сконцентрований на Windows-десктопі й не має рідної підтримки мобільних чи веб-платформ (хіба що через Xamarin чи Electron-обгортки), однак його розширена система стилів, шаблонів і векторної графіки дозволяє створювати адаптивні інтерфейси високої деталізації для різних розмірів вікон. Phaser (HTML5/JS) принципово орієнтований на браузер, тож працює скрізь, де є сучасний WebGL-движок (десктоп, мобільні браузери), і підтримує респонсивну розмітку Canvas або CSS-обгортки; це робить його легким для швидкого прототипування та розгортання, але обмеженим у доступі до «рідних» API та більш просунутих можливостей GPU.

Проаналізувавши наявні рішення , наразі найбільш перспективно виглядає саме рушій Unity , його потужності в кросплатформенності та адаптивності дають неабиякі можливості для творчості та проектування , а це саме те що нам необхідно для створення симулятору. Давайте розглянемо основний функціонал Unity та чи вистачить цього для створення додатку.

Давайте спробуємо оглянути та порівняти всі наявні двигуни на предмет наявності всіх необхідних елементів необхідних для створення додатку.

Та зробимо висновок , який саме рушій буде найкращим для застосунку.В таблиці наведено порівняння популярних рушіїв для розробки програмних додатків , де перечислено їх переваги та недоліки.

Критерій	Unreal Engine	Godot	Unity	WPF (.NET)	Phaser (HTML5/JS)
Призначення	Переважно 3D ігри, AAA-рівень	Легкий, універсальний для 2D/3D	Потужний для 2D/3D	Бізнес-додатки, десктоп UI	Браузерні 2D ігри

Критерій	Unreal Engine	Godot	Unity	WPF (.NET)	Phaser (HTML5/JS)
Підтримка 2D	Обмежена, не основний фокус	Відмінна, оптимізована	Дуже хороша, зручна система Sprite	Призначена для форм і UI елементів	Орієнтована на 2D, зручно для піксель-арту
UI/Інтерфейс	Складна, через UMG	Простий, але базовий	Гнучкий (Canvas, UI Toolkit)	Найкраща для UI, DataBinding	Проста, але кастомна
Простота для початківця	Важкий поріг входу	Дуже зручний для навчання	Середній — багато матеріалів	Дуже простий для C#/.NET користувачів	Простий для JS/HTML знайомих
Кросплатформеність	Так, але великі збірки	Так, легкі збірки	Так, широке охоплення	Тільки Windows	Працює в браузері, майже скрізь
Продуктивність для 2D	Надлишкова потужність	Оптимізований для 2D	Висока, але може бути overkill	Висока, але тільки для Win	Оптимально для простих 2D
Інструменти для симуляторів	Не спеціалізовані	Легко реалізується через ноди/сцену	Prefab + Script підходить ідеально	Binding, MVVM ідеальні для логіки	Ручна реалізація, але можливо
Підтримка спільноти/матеріалів	Висока, але більше по 3D	Активна, зростає	Масивна кількість туторіалів	Міцна серед .NET спільноти	Дуже багато прикладів і статей
Ліцензія / Вартість	Безкоштовно до $1M доходу	Повністю безкоштовно (MIT)	Безкоштовно, Pro-ліцензія для бізнесу	Безкоштовно з .NET SDK	Повністю безкоштовно
Рекомендованість для проекту	(надто важкий)	(ідеально для 2D-симулятора)	(універсальний вибір)	(для win-only UI симулятора)	(підходить для веб-версії)

Таблиця 2.1 – Порівняння рушіїв для створення додатків

Чітко проаналізувавши аналоги ,зрозуміло що Unity є найкращим вибором на данний момент , активна підтримка спільноти та легкість у прототипуванні дають величезні переваги над конкурентами.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Огляд всіх етапів прототипування

Для створення симулятору , інтерфейс є одним з ключових елемнтів , який створорює основний досвід для користувача. Зважаючи на те , що поставлено за мету створення навчального-симулятору , необхідно створювати інтерфейс , який буде максимально приближеним до реальної системи. Тож перед початком створення кодової частини , вкрай необхідно провести візуальний аналіз наших POS систем , а саме ShopKeep та MINI-T400 та зрозуміти які етапи розробки треба пройти для створення симуляції

Перед тим як створити інтерфейс , необхідно зануритись в контекст людей які працюють з даними системами. Під час дослідження матеріалів , було знайдено декілька відео людей , які працюють з ними , та зміг проаналізувати те , як ці користувачі працюють з інтерфейсом ,зрозумів що є головним а що є другорядним. Також важливо зрозуміти з якими труднощами , під час користування інтерфейсом може зіткнутися касир.

Також , для розуміння контексту , вкрай необхідно поглянути на інтерфейси інших касових систем , проаналізувати їх переваги та недоліки над системами які ми обрали ,проблема може бути у складній ієрархії меню або у занадто дрібному шрифті , саме на ці деталі необхідно звернути увагу. Такий підхід може дозволити оминути проблеми з якими стикаються юзери і обрати найкращий підхід до розробки.

Для початку прототипування інтефейсу , буде використано прості кнопки , без жодного дизайну чи особливого розташування . Це треба для загального зпрощення прототипування інтефейсу та скоротиь час необхідний на розробку.Після того , як я зроблю каркас системи , з усім базовим практичним функціоналом ,його буде взято за основу для створення вже фінального дизайну.

Кольори, іконки, типографіка — усе має надавати якийсь загальний досвід , створючи синергію. Також важливо розуміти , що всі елементи дизайну мають адаптуватись до різних розмірів екрану , тому можливо деякі деталі необхідно буде ападтувати під цю

необхідність , при цьому важливо не втратити основну картину , щоб зберегти баланс між адаптивністю та схожістю на систему до якої я роблю симуляцію.

Немає нічого страшнішого за користувача, який губиться в меню. В інтернеті вдалося знайти матеріали де було показано як користувачі працювали з цими системами , що дало можливість проаналізувати в яких місцях користувачі губляться найчастіше. Далі — швидкі ітерації: змінюю відступи, збільшую touch-target до 48×48 px, оптимізую порядок табуляції.

Після запуску симулятора збираю метрики: Time-on-Task, кількість помилок, час на навчання. Ці дані стають відправною точкою для оновлень: можливо, варто додати темну тему або змінити розташування панелі статусу принтера. UI — це живий організм, який росте разом із потребами користувачів.

Після того як підготовка каркасу для симулятору буде виконана , можна буде почати працювати з кодовою частиною проекту та створювати всі ноебхідні класси для симуляції даних систем та коректної взаємодії зі створеним інтерфейсом.

3.2 Огляд використаних технологій під час розробки

Під час розробки додатку було використано цілу низку систем та програм, які надали можливість виконати поставлену задачу, а саме: Photoshop для обробки графіки, Microsoft Excel для обробки баз даних, Microsoft Visual Studio для редагування коду та ігровий рушій Unity. Давайте розглянемо ці програми та з’ясуємо, яким чином вони будуть використані під час розробки.

Щоб надихнутися оригінальною системою, важливо також намагатися створювати такі графічні ассети, які будуть подібними до оригіналу. Для такої задачі найкраще підійде Photoshop. Ця програма має весь необхідний функціонал для створення UI та будь-якої графіки. Серед основних інструментів — "Move Tool" для переміщення об’єктів, "Marquee" та "Lasso" для виділення ділянок зображення, "Brush" і "Eraser" для малювання та стирання, "Clone Stamp" для копіювання фрагментів, а також "Healing Brush" для ретуші. Інструмент "Gradient" дозволяє створювати плавні переходи кольорів, а "Text Tool" — додавати й редагувати текст. Крім того, Photoshop пропонує шари (layers), маски, фільтри, регулювання кольору і

безліч інших можливостей, які роблять його універсальним засобом для обробки графічних елементів.

Unity буде ядром нашого додатку, який поєднуватиме всі частини одна з одною та створюватиме цілісний продукт. Важливо, що Unity дає можливості працювати з усіма необхідними елементами, такими як обробка баз даних та створення User Interface за допомогою C# для обробки логіки. Завдяки такому потужному функціоналу можна зрозуміти, що в ньому є все необхідне для того, щоб розробити прототип, який пізніше буде перероблений у готовий продукт. Оскільки Unity має величезну спільноту розробників, буде змога легко дізнатися відповіді на складні питання, що зробить процес розробки чітким та зрозумілим.

Для написання коду була використана мова програмування C#, яка застосовується для написання скриптів у рушії Unity. Потрібно також взяти до уваги, що C# є об’єктно-орієнтованою мовою програмування з чіткою структурою коду, що полегшує розробку складних проєктів. Вона має зрозумілий синтаксис, розвинену систему типів і підтримку сучасних концепцій, таких як асинхронне програмування. Завдяки глибокій інтеграції з Unity API, C# дозволяє швидко створювати функціональні прототипи та розширювати їх до повноцінних проєктів.

Для обробки баз даних із товарами, які є в симуляторі, ми будемо використовувати Excel та файли .csv, які підтримуються Unity. Excel є потужним інструментом для обробки баз даних, тому експорт створеної бази даних у форматі .csv не є проблемою. Це дозволяє спростити процес оновлення інформації про товари — у разі змін достатньо лише змінити дані в Excel та повторно зберегти файл у форматі .csv. Такий підхід є ефективним, масштабованим та зручним для тестування, особливо на етапах розробки й налаштування інтерфейсу касового апарату.

3.3 Основні патерни коду використані під час розробки

Під час розробки тренувального симулятора я зіштовхнувся з низкою викликів, що потребували глибокого аналізу та творчих підходів до організації коду. Хороша архітектура додатку – це не просто набір інструкцій, а справжня система, де кожен компонент працює у гармонії з іншими, сприяючи загальній стабільності,

масштабованості та ефективності. Аналізуючи ключові принципи, я дійшов висновку, що інтеграція перевірених паттернів проектування є фундаментальною для створення надійного продукту. Дозвольте поділитися думками, як це було реалізовано на практиці.

Уявіть, що замість постійного створення нових об’єктів, система має надійний «банк» готових до використання ресурсів у кеші програми. Паттерн Object Pool дозволить використовувати одні й тіж обєкти раз за разом ,наприклад як панелі товарів при пошуку, при цьому не збільшуючи нагрузку на систему , що робить її значно плавнішою навіть при інтенсивному навантаженні. Як казав один з класиків у галузі, «ефективність – це мистецтво використання ресурсів у найкращий спосіб» .

Singleton є одним з найвживанішим мною паттернів в обєктно орієнотованому програмуванні? Цей паттерн гарантує, що певний компонент, такий як SFX Manager , або PageManager, буде доступний по всій програмі через єдиний екземпляр. Такий підхід дає можливість серйозно спростити складність програми , дає можливість створення простої та чіткої структури проекту. Розуміючи, як важливо підтримувати баланс, я неодноразово переконувався: створення одного центрального модуля значно знижує ризики перевантаження системи.

Основи ООП – інкапсуляція, наслідування та поліморфізм – є еталонними алгоритмами у розробці будь-якого додатку. Завдяки використанню цих принципів код залишаєшться організованим , незважаючи на складність програми:Завдяки цим принципам функціональні частини залишаються ізольованими , що значно покращує надійність, а використання наслідування допомагає створити модульні, розширювані компоненти. Наприклад, базовий клас для управління транзакціями дав змогу адаптувати систему під різні типи касових апаратів. Чи не захоплює вас можливість створювати щось універсальне, що легко підлаштовується під будь-які вимоги?

Одним із ключових аспектів написання якісного коду є збереження його простоти. Як наголошують автори посібника,

"Про простоту часто говорять при розробці програмного забезпечення, і вона є передумовою надійності. Чи може ваш дизайн програмного забезпечення впоратися зі змінами у виробництві? Чи можете ви розширювати і підтримувати свій додаток з

часом? Хоча ви можете створити ту саму функціональність без патернів (і часто швидше), щось швидке і легке не обов'язково призводить до чогось простого". [27]

Таким чином, навіть якщо реалізувати функціональність можна швидко й без застосування патернів, це не гарантує простоти структури програми. Використання перевірених шаблонів (наприклад, SOLID-принципів) допомагає зробити код гнучким, масштабованим та зручним для подальших змін.

Ще одним незамінним алгоритмом є концепція паттерну Observer. Він дозволяє різним частинам додатку сповіщати одна одну про зміни – наприклад, при натисканні кнопки користувача або зміні стану об’єкта. Завдяки цьому, інтерфейс працює в реальному часі, що надає додатку необхідну гнучкість і жвавий відгук.

Застосування паттерну Factory дозволило централізовано створювати об’єкти різного типу без прямого виклику їх конструкторів з різних частин коду. Це призвело до значного зниження зв’язності між компонентами системи та полегшило розширення функціональності. Додавання нового типу об’єкта обмежувалося лише змінами у фабричному класі – просто і елегантно!

Для реалізації варіативних алгоритмів поведінки було використано **Strategy Pattern**. Цей підхід дає змогу визначати різні стратегії обробки транзакцій або управління інтерфейсом в залежності від конкретного сценарію. Зі свого досвіду, я впевнений, що можливість динамічного зміни алгоритмів без перезапуску програми робить систему надзвичайно адаптивною до нових викликів та умов.

Ще одним кроком до підвищення якості коду стала імплементація Dependency Injection (DI). За допомогою цієї техніки я зміг знизити зв’язок між компонентами, що значно полегшує їх заміну і тестування. Наприклад, компоненти, такі як SFX Manager чи модулі для роботи з базами даних, можна інтегрувати в ізольованому середовищі без страху порушити загальну структуру системи.

Насамкінець, принцип Separation of Concerns допоміг розділити логіку роботи системи на окремі частини: інтерфейс користувача, обробка даних, аудіо- та візуальні ефекти. Такий підхід сприяє не лише чистоті коду, але й забезпечує зручність підтримки. Пам’ятаю, як під час першого великого проекту вивчав цей принцип, і

наскільки полегшилось життя, коли кожен модуль займався своєю задачею, немов добре налаштований оркестр.

Використання паттернів таких як Factory чи Strategy – це абсолютно необхідне рішення яке має сенс майже в будь-якому проекті, такий підхід дозволяє вибудувати справжню стратегічну поведінка в розробці програмного забезпечення, яку підтримують провідні експерти галузі , системний підхід до кодування дозволяє створити гнучкі та масштабовані рішення, що відповідають динамічним вимогам сучасних IT-проектів.

Особисто я неодноразово стикався з проблемою заплутаності проектів і зрозумів, що лише ретельно спланована архітектура може запобігти серйозним збоям у системі. Тому використання цих архітектурних рішень у нашому проекті стовідсотково виправданим.

На завершення, інтеграція вищезгаданих паттернів не лише надала системі модульну архітектуру, але й відкрила нові можливості для її подальшого розвитку. Цей підхід несе глибокий вплив на продуктивність, стабільність та адаптивність симулятора, що особливо важливо в умовах постійно змінних вимог сучасного навчального процесу.

3.4 Особливості створення інтерфейсів у Unity

Unity має досить великий перелік інструментів які дають можливість проектування комплексних інтерфейсів для додатків будь-якої складності, я пропоную нам розглянути ці інструменти та зрозуміти де і як їх можна використати під час створення симулятору. Нижче наведено основні компоненти Unity, що допоможуть створити дійсно ефективний та адаптивний інтерфейс, а також мої роздуми щодо їх використання.

Canvas – це фундамент, на якому будується весь інтерфейс. Він не лише організовує розміщення таких компонентів, як кнопки, текстові поля чи панелі, але й підтримує масштабування для різних роздільних здатностей екрана. Завдяки Canvas’у забезпечується стабільність і візуальна узгодженість UI, що дозволяє зберегти

зручність та гармонійний вигляд навіть на пристроях з різними характеристиками дисплеїв.

Компоненти Horizontal Layout Group і Vertical Layout Group є незамінними для створення чіткої структури розташування елементів. Вони дозволяють автоматично розподіляти об’єкти по горизонталі або вертикалі, це може бути наприклад використано для структурування розташування кнопок циферблату .

Для впорядкування великої кількості однотипних елементів немає нічого простішого, ніж Grid Layout Group. Цей компонент дозволяє створити сітку з рівномірними відступами, що особливо корисно при розміщенні, наприклад, списків товарів або цифрових панелей. Такий підхід не тільки забезпечує компактність, але й робить інтерфейс легким для сприйняття, адже кожен елемент займає своє законне місце в "маленькому світі" UI.

Ще одним чудовим засобом для динамічної адаптації є Content Size Fitter. Цей інструмент дозволяє елементам змінювати свої розміри залежно від вмісту, що є справжнім порятунком при роботі з текстом або іншими динамічними об’єктами. Наприклад, коли обсяг тексту змінюється, Content Size Fitter миттєво підлаштовує розміри панелі, зберігаючи її естетичний вигляд і логічну цілісність інтерфейсу.

Canvas Scaler — це незамінний інструмент для масштабування UI під різноманітні розміри екранів. Він дозволяє автоматично підігнати інтерфейс під будь-який пристрій завдяки режимам Scale with Screen Size та Constant Pixel Size. Цей компонент зберігає пропорції елементів, не дозволяючи їм втратити якість навіть при зміні роздільності, що є надзвичайно важливим для симуляторів, котрі використовуються як на великих моніторах, так і на мобільних пристроях.

Щоб елементи залишалися на своїх місцях незалежно від змін розміру екрана, використовуються Anchor Points та Pivot Points. Перші закріплюють положення елементів відносно країв, а останні забезпечують правильний центр обертання й масштабування. Ці засоби гарантують, що навіть при значних змінах роздільної здатності, інтерфейс збереже свою структуру та правильну композицію.

Одне з ключових понять у UX-дизайні формулюється так:

„Користувацький досвід — це результат сприйняття, що формується під час взаємодії людини з продуктом. Він охоплює емоції, переконання, уподобання, фізичні й психічні реакції, поведінку та досягнення цілей.“ [29]

Таким чином, проектування інтерфейсу має враховувати не лише візуальні чи функціональні властивості, але й те, як саме користувач відчуває та оцінює кожен етап взаємодії з продуктом.

Кожен із зазначених компонентів – від Canvas до Pivot Points – грає свою унікальну роль у формуванні адаптивних інтерфейсів. Unity надає розробникам потужні інструменти, які, у поєднанні з творчим підходом, дозволяють створити UI, що чудово реагує на зміни умов та пристроїв. Чи не є захоплюючою можливість створювати справді "розумний" інтерфейс, який не просто виглядає гармонійно, а й відчувається інтуїтивно природним? Сам я завжди із задоволенням відзначаю, що такі рішення сприяють оптимізації робочих процесів і підвищенню зручності використання додатків.

У процесі розробки інтерфейсу POS-систем ключовим є забезпечити інтуїтивність та швидкий доступ до найчастіше використовуваних функцій. Як зазначає Keenan,

"Частиною того, що робить дизайн POS UI ефективним, є інтуїтивно зрозумілий макет. Персонал повинен мати можливість знаходити інформацію та виконувати найпоширеніші дії за якомога меншу кількість кліків". [18]

Отже, проектування панелі оператора має базуватися на принципі мінімізації кроків для виконання основних завдань: пошуку товару, оформлення продажу та друку чеку. Це дозволяє значно скоротити час обслуговування клієнтів і зменшити навантаження на персонал.

3.5 Опрацювання взаємодії між симулятором та користувачем

Сучасні інтерактивні додатки, зокрема симулятори, де створення максимально реалістичного досвіду є першорядним завданням, отримали новий імпульс завдяки ретельно продуманим інтерактивним фідбекам для кнопок. Для мене особисто інтеграція аудіо- та візуальних сигналів — це не просто технічна задача, а спосіб

забезпечити миттєве підтвердження дій користувача та підвищити інтуїтивність взаємодії.

При натисканні кожної кнопки симулятора спрацьовує звуковий сигнал (SFX), що одразу інформує користувача про виконання дії. Це своєрідний «мовний сигнал» системи, який допомагає знизити невпевненість під час роботи. Разом із звуковим супроводом, під час наведенні курсора миші активується візуальне підсвічування, що створює додатковий рівень взаємодії. Такий підхід нагадує гру світла та звуку, де кожен елемент робить свій внесок у створення зручного і динамічного інтерфейсу.

Щоб оптимізувати аудіовідтворення, у симуляторі реалізовано спеціалізований компонент — SFX Manager. Він забезпечує централізоване керування аудіо, програючи всі звукові ефекти через один Audio Source. Така організація не лише зменшує навантаження на систему, а й спрощує налаштування гучності, затримок та інших параметрів. Стандартний компонент Unity — Audio Source — відповідає за відтворення аудіокліпів в сцені, надаючи можливості для регулювання якості звуку, його тону і просторового розміщення.Також. Злиття цих підходів дозволяє уникнути дублювання коду та гарантує стабільну роботу додатка.Для своєї роботи , цей компонент використовує паттерн Singleton , що дає можливість отримати доступ до нього з будь – якого скрипту , на цій UML – діаграммі можна поглянути як цей алгоритм працює в середовищі Unity.

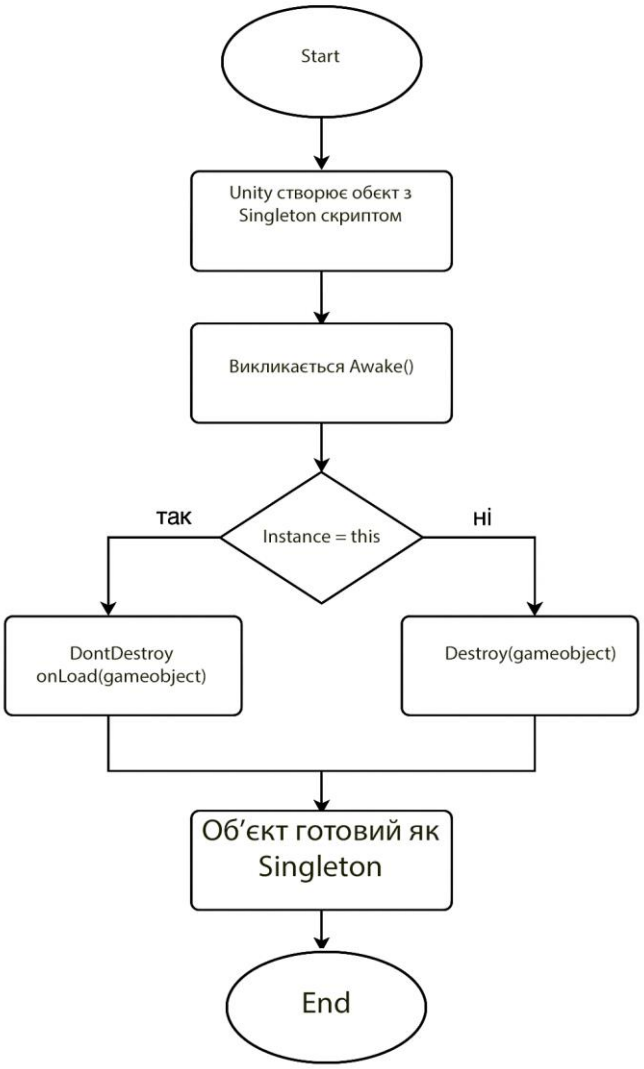

Рисунок 3.1 – Зображення алгоритму роботу паттерну Singleton

Інтерактивні фідбеки значно покращують зручність використання інтерфейсу. Візуальні підсвічування не лише підвищують естетичну складову, а й надають додаткову інформацію про доступність функцій, що зменшує ймовірність помилок під час роботи. Це особливо важливо для новачків, які проходять навчання перед тим, як працювати з реальним обладнанням. Як результат, інтеграція аудіо- та візуальних сигналів перетворює симулятор на ефективний інструмент для швидкої адаптації співробітників до реальних умов роботи з касовими апаратами.

3.6 Психолого-педагогічні аспекти використання симуляторів для навчання

Інтерактивні симулятори відкривають перед сучасною освітою неймовірні перспективи, адже вони сприяють набуттю практичних навичок і глибшому

засвоєнню матеріалу, що позитивно впливає на підготовку майбутніх професіоналів. Якщо задуматися, чи не є досвід, отриманий через пряме взаємодію з симульованими середовищами, найкращим інструментом для навчання? Саме завдяки такому підходу, який віддзеркалює принципи експериментального навчання, інформація закріплюється через власний досвід, що значно полегшує розуміння і запам'ятовування навіть найскладніших концепцій.

Однією з ключових переваг інтерактивних симуляторів є можливість миттєвого отримання зворотного зв'язку. Коли користувач бачить наслідки своїх дій буквально в реальному часі, це стимулює його вчитися на власних помилках і відточувати правильні алгоритми дій. На власному прикладі я переконався: коли результати видно одразу, знижується рівень когнітивного навантаження, а складні теоретичні поняття стають набагато доступнішими для розуміння.

Інтегруючи симулятори у навчальний процес, організації можуть сформувати реалістичну, але контрольовану ситуацію, де майбутній оператор має можливість робити помилки без наслідків для реальної роботи. Уявіть собі ситуацію, коли помилка не призводить до серйозних наслідків – чи не є це найбільшим полегшенням для тих, хто тільки починає свій професійний шлях? Таке навчання сприяє набуттю впевненості, оскільки дозволяє відточувати навички, не відчуваючи страху перед можливими помилками.

Інтерактивний характер симуляторів значно підвищує мотивацію учасників. Гейміфікація, елементи змагання і системи досягнень перетворюють навчання з монотонного процесу в захопливу подорож. Особисто я неодноразово відчував, як гра підштовхує до активної участі та самовдосконалення. Впровадження різноманітних сценаріїв, що симулюють реальні ситуації на робочому місці, дозволяє не лише «втягнути» співробітників у процес, але й сформувати у них позитивне ставлення до постійного професійного розвитку.

Не можна не відзначити і розвиток критичного мислення – завдяки постійній взаємодії з симулятором, кожен користувач вчиться швидко аналізувати ситуації, обирати оптимальні стратегії вирішення завдань і враховувати можливі ризики. У світі, де швидкість прийняття рішень часто є вирішальним фактором успіху, такий

досвід набуває особливої ваги. Чи не варто нам постійно шукати шляхи для вдосконалення саме цього аспекту?

Таким чином, інтеграція інтерактивних симуляторів у навчальний процес – це багатогранний підхід, що не лише оптимізує засвоєння матеріалу, але й створює умови для безпечних експериментів, стимулює мотивацію та розвиває критичне мислення. Як зазначає класик педагогіки Ерік Еріксон, «навчання – це не просто накопичення знань, а трансформація досвіду», що ілюструє важливість практичного застосування отриманих навичок у симульованому середовищі. Особисто я впевнений: саме таке навчання дозволяє підготувати кваліфікованих співробітників, які зможуть втриматися в конкурентному середовищі ринку праці, адже вони встигають адаптуватися до змін та вчитися у реальному часі . Цей підхід, без сумніву, стає важливим інструментом сучасної освіти, перетворюючи традиційний процес навчання на динамічну та інтерактивну подорож до здобуття знань і навичок.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1 Створення основних сторінок для симулятору Shoop Keep.

До основних сторінок симулятора належать такі сторінки:

1. Сторінка вводу коду працівника – відображена на рисунку 4.1

Рисунок 4.1 - Сторінка вводу коду працівника

На малюнку 4.1 зображено початкову сторінку індетифікації кассиру , для симулятору Shop Keep. Завдяки реалізованому коду система дає змогу опрацювати процесс аунтифікації кассира . Після виконання цієї операції програма дасть вам працювати з усими основними функціями системи, та дозволить перейти до основного інтерфейсу касового апарату. Для тестування використовується код «1234»: введення цього значення та натиснення кнопки “Login” перенесе вас до основої сторінки цієї системи. Крім того, на сторінці передбачена кнопка «Clear», яка дозволяє очистити поле вводу для зручного повторного введення коду.

2. Головна сторінка - відображена на рисунку 4.2

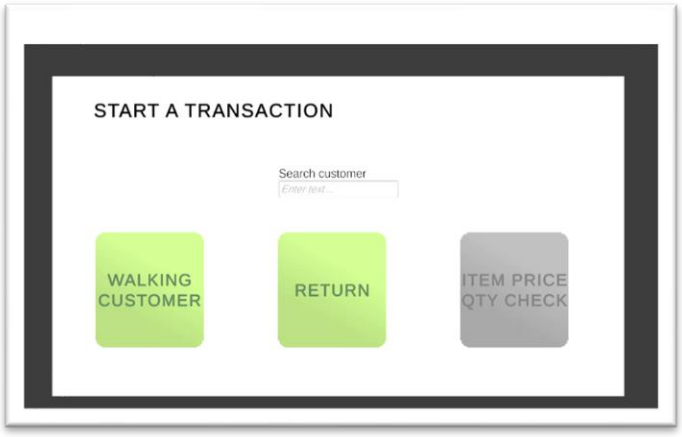

Рисунок 4.2 – Головна сторінка симулятору

На цьому зображенні представлена головна сторінка касового апарату. Після введення правильного коду система перенаправить вас сюди, де ви зможете обрати подальші дії. З головної сторінки можна або повернутися назад, або перейти до сторінки обслуговування клієнта, натиснувши кнопку «WALKING CUSTOMER».

3. Сторінка формування замовлення. - відображена на рисунку 4.3

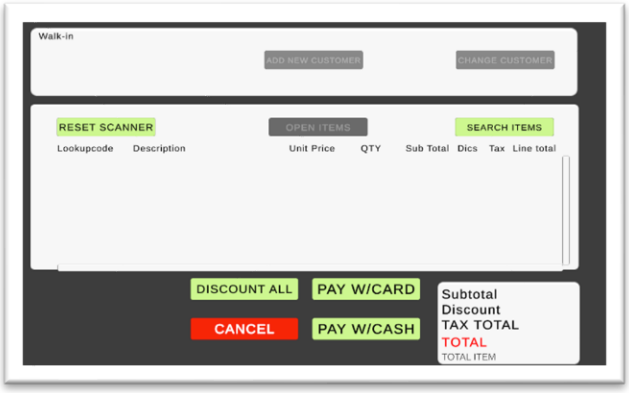

Рисунок 4.3 – Сторінка формування замовлення

Це сторінка на якій реалізована функцію формуваня замовлення та вибору оплати.З реалізованого функціоналу ми можемо перейти на сторінку “Search Items” і оглянути наявні товарі в базі даних. Ця сторінка – копія того як це виглядає в ShopKeep(beta) .

4.Сторінка пошуку , та додачі товару до списку покупок. - відображена на рисунку

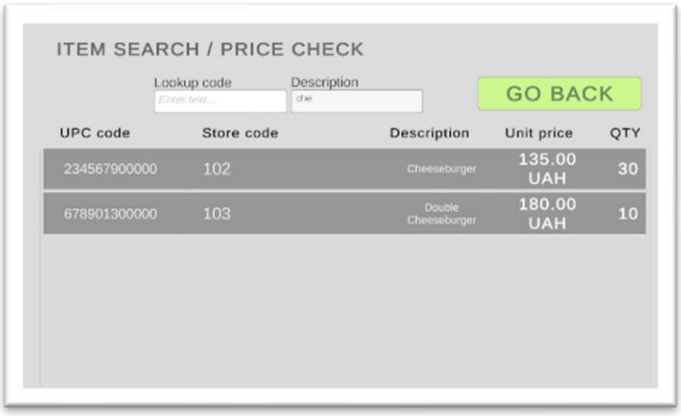

Рисунок 4.4 – Сторінка пошуку , та додачі товару до списку покупок

На цій сторінці ви маєте змогу подивитись наявний товар в базі даних використовуючи пошук за “Lookup code” або “Description” .

4.2 Реалізація пошуку товару за допомогою бази даних для системи Shoop Keep

У симуляторі реалізована функція пошуку товарів у базі даних, яка дозволяє касирам швидко знаходити необхідні позиції, вводячи відповідні параметри або назву товару. Коли користувач вводить запит, система перевіряє базу даних та отримує відповідні результати, які відображаються на UI.

Для оптимізації відображення результатів використовується підхід із застосуванням префабів (Prefab), що дозволяє створювати однотипні панелі з інформацією про товар. Кожен раз, коли система знаходить товар, для нього завантажується відповідний префаб, на якому відображаються всі дані, такі як назва товару, його ціна, опис та інші важливі характеристики. Щоб зробити цей процес більш ефективним, префаби беруться з об'єктного пулу (Object Pool) — це дозволяє уникнути частих операцій створення й знищення об'єктів, знижуючи навантаження на систему.

Об'єктний пул (Object Pool) — це підхід, який передбачає створення певної кількості об'єктів наперед і зберігання їх у спеціальному контейнері. Коли виникає

потреба у відображенні нових товарів, система не створює нові екземпляри префабу, а бере їх із пулу. Якщо якийсь із об'єктів більше не потрібен, він повертається до пулу, де його можна буде використати повторно. Це значно покращує продуктивність, оскільки зменшує кількість операцій створення та видалення об’єктів.

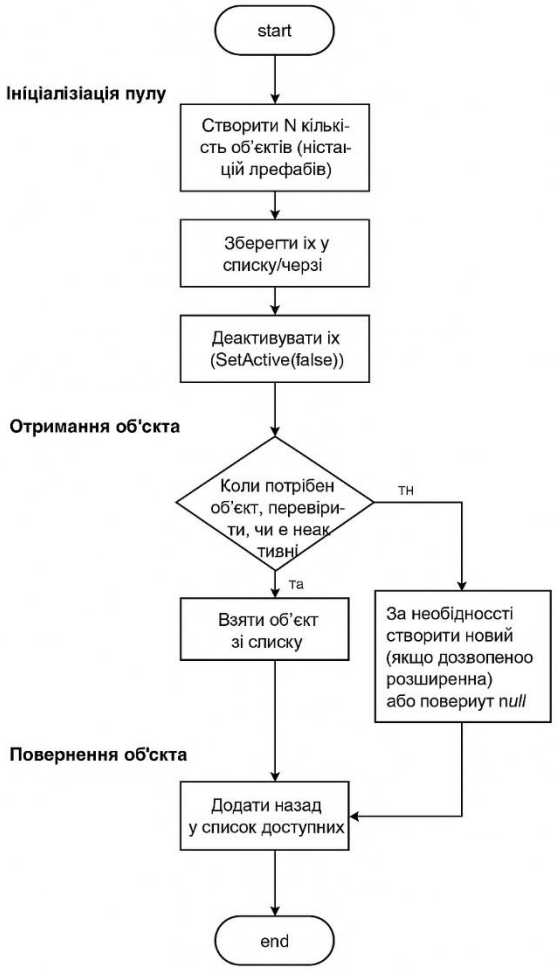

Рисунок 4.5 – Зображення роботи алгоритму Object Pool

Цей код демонструє практичну реалізацію паттерну Object Pool

```
using System.Collections.Generic;
using UnityEngine;

public class ResourceManager : MonoBehaviour
{
   [System.Serializable]
   public class Reservoir
   {
      public string identifier;
      public GameObject blueprint;
      public int capacity;
   }

      public static ResourceManager Central;
      public List<Reservoir> reservoirs;
      public Dictionary<string, Queue<GameObject>>
stockpile;

      private void Awake()
      {
         Central = this;
         stockpile = new Dictionary<string,
Queue<GameObject>>();
         foreach (Reservoir reservoir in reservoirs)
```

```
            {
                Queue<GameObject> entityCache = new Queue<GameObject>();
                for (int i = 0; i < reservoir.capacity; i++)
                {
                    GameObject entity = Instantiate(reservoir.blueprint);
                    entity.SetActive(false);
                    entityCache.Enqueue(entity);
                }

                stockpile.Add(reservoir.identifier, entityCache);
            }
        }

        public GameObject FetchBlueprintByIdentifier(string identifier)
        {
            foreach (Reservoir reservoir in reservoirs)
            {
                if (reservoir.identifier == identifier)
                {
                    return reservoir.blueprint;
                }
            }
            return null;
        }

        public void RestoreToCache(string identifier, GameObject entityToRestore)
        {
            if (!stockpile.ContainsKey(identifier))
            {
                return;
            }
            entityToRestore.transform.parent = null;
            entityToRestore.transform.position = Vector3.zero;
            entityToRestore.transform.rotation = Quaternion.identity;
            entityToRestore.SetActive(false);
            stockpile[identifier].Enqueue(entityToRestore);
        }

        public GameObject DeployFromCache(string identifier, Vector3 coordinates, Quaternion orientation, Transform ancestry = null)
        {
            if (!stockpile.ContainsKey(identifier))
            {
                Debug.Log(identifier + "_не_існує");
                return null;
            }

            foreach (GameObject entity in stockpile[identifier])
            {
                if (!entity.activeInHierarchy)
                {
                    entity.transform.position = coordinates;
                    entity.transform.rotation = orientation;
                    entity.SetActive(true);

                    if (ancestry != null)
                    {
                        entity.transform.parent = ancestry;
                    }
                    else
                    {
                        entity.transform.parent = null;
                    }

                    return entity;
                }
            }

            Reservoir reservoir = reservoirs.Find(r => r.identifier == identifier);
            if (reservoir != null)
            {
                GameObject freshEntity = Instantiate(reservoir.blueprint);
                freshEntity.transform.position = coordinates;
                freshEntity.transform.rotation = orientation;
                freshEntity.SetActive(true);

                if (ancestry != null)
                {
                    freshEntity.transform.parent = ancestry;
                }

                stockpile[identifier].Enqueue(freshEntity);
                return freshEntity;
            }

            return null;
```

}

Коли об'єкт запитується з пулу, він налаштовується з урахуванням інформації про конкретний товар. На панелі відображається основна інформація, в нашому випадку це код для індентифікації , опис , ціна за одиницю продукту та кількість. Це дозволяє швидко й ефективно показати інформацію про товар без зайвих витрат ресурсів на створення об'єктів щоразу, коли користувач здійснює пошук.

Приклад панельки префабу для товару можна подивитися на рисунку 4.5

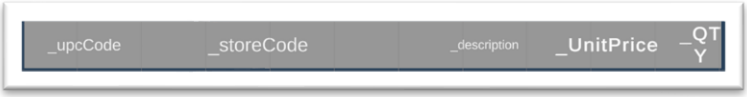

Рисунок 4.6 - Зображеня макета товару для функції пошуку

Ця панель включає кілька ключових компонентів: UPC Code , Store Code(код товару), текстовий блок з описом , ціну та кількість. Завдяки такому підходу, UI у симуляторі залишається зручним та ефективним, забезпечуючи швидкий доступ до інформації без затримок, що важливо для симуляцій, які потребують швидкої взаємодії з користувачем.

Завдяки ефективній комунікації бази даних з товарами та системи пошуку, забезпечується ефективна робота функціоналу симулятора , та дозволяє користувачам швидко знаходити потрібні позиції за допомогою унікальних ідентифікаторів товарів, таких як UPC (Universal Product Code), а також за описом товару. Ця функція значно спрощує процес взаємодії касира з системою, особливо у випадках, коли необхідно знайти товар серед великої кількості одиниць.

Однією з задач проектування було створення алгоритму пошуку за UPC кодом.UPC Code — це стандартний код, що складається з 12 цифр, який унікально ідентифікує товар. В программі-симуляторі було реалізовано сценарій коли користувач може ввести UPC код у спеціально відведене поле пошуку, після чого система перевіряє підготовлену базу даних на наявність відповідного товару. Завдяки цьому підходу, касири можуть швидко знайти потрібні товари, просто просканувавши штрих-код або

ввівши код вручну. Система моментально реагує на запит, відображаючи інформацію про товар на екрані.

Окрім пошуку за UPC,також була реалізована можливість пошуку за описом товару. Користувач може ввести будь-які ключові слова, пов'язані з товаром, наприклад, його назву або основні характеристики. Система проводить пошук по всій базі даних ,при цьому використовуючи оптимізовані алгоритми пошуку, знаходячи товари, які відповідають введеним критеріям. Цей функціонал особливо корисний у випадках, коли товар не має штрих-коду або коли касир не знає UPC, але пам'ятає, як виглядає товар чи які в нього характеристики.

При реалізації пошукової функції у симуляторі використовується ефективний механізм, що забезпечує швидкий доступ до бази даних. Коли користувач вводить UPC або опис, система виконує запит до бази даних і отримує відповідні результати.

Цей клас реалізує функціонал пошуку товарів у базі даних та їх відображення в UI. Основні моменти:

За допомогою двох полів вводу (опис та код UPC) клас відслідковує зміни через події і викликає відповідні методи (UpdateDescriptionSearch та UpdateLookupCodeSearch).

```
private void UpdateDescriptionSearch(string value)
{
    searchHandling.Clear();
    lookupCodeInput.text = "";

    SearchInDescription(value);

    UpdateDisplayed();
}

private void UpdateLookupCodeSearch(string value)
{
    searchHandling.Clear();
    descriptionInput.text = "";

    SearchInUPCCode(value);

    UpdateDisplayed();
```

- }

Методи SearchInDescription та SearchInUPCCode виконують паралельний перебір бази даних товарів (data.itemDatabase). Для кожного товару перевіряється, чи міститься введений текст у відповідному полі (опис чи UPC-код). Збіги додаються до списку searchHandling.

```
public void SearchInDescription(string input)
{
   // List<string> result = new List<string>();
   searchHandling.Clear();
   Parallel.ForEach(data.itemDatabase, (item) =>
   {
      if (item.description.Contains(input, StringComparison.OrdinalIgnoreCase))
      {
         lock (searchHandling)
         {
            searchHandling.Add(item);
         }
      }
   });
}
public void SearchInUPCCode(string input)
{
   searchHandling.Clear();
   Parallel.ForEach(data.itemDatabase, (item) =>
   {
      if (item.uPCcode.Contains(input, StringComparison.OrdinalIgnoreCase))
      {
         lock (searchHandling)
         {
            searchHandling.Add(item);
         }
      }
   });
}
```

Після виконання пошуку метод UpdateDisplayed очищує попередньо згенеровані панелі з результатами, повертаючи їх у пул об'єктів, та створює нові UI-елементи для кожного знайденого товару. Метод SetupDisplay налаштовує відображення даних товару в панелі, заповнюючи відповідні текстові поля (UPC-код, код магазину, опис, ціна та кількість).

```
private void UpdateDisplayed()
{
   foreach (GameObject pannel in spawnedPrefabs)
   {
```

```
            ObjectPool.Instance.ReturnToPool("searchPannel", pannel);
        }
        spawnedPrefabs.Clear();
        foreach (Item item in searchHandling)
        {
            SetupDisplay(item);
        }
    }
```

Метод FixScientificNotation перетворює рядок із числовим значенням, яке може бути у науковій нотації, у формат без експоненціального запису.

```
    string FixScientificNotation(string input)
    {
        //
        double number = double.Parse(input, System.Globalization.NumberStyles.Float);



        return number.ToString("0");
    }
```

Таким чином, код відповідає за швидкий та ефективний пошук товарів за заданими критеріями з подальшим оновленням UI з використанням пулу об'єктів для оптимізації продуктивності.

Кроки пошуку:

Користувач вводить UPC код або опис товару у спеціально відведене поле.

Система обробляє введені дані і формує запит до бази даних.

Якщо товар знайдено, система повертає відповідні дані, включаючи інформацію про товар, такі як назва, ціна, зображення тощо.

Результати відображаються на екрані у вигляді панелей, створених з використанням префабів, що дозволяє легко сприймати інформацію про кілька товарів одночасно.

Цей підхід забезпечує швидкий та зручний доступ до інформації про товари, дозволяючи касирам ефективно обслуговувати клієнтів і виконувати касові операції. Пошук за UPC і описом товарів є ключовими функціями, що підвищують загальну продуктивність та ефективність роботи симулятора.

У процесі розробки симулятора виникла необхідність у використанні бази даних для зберігання та обробки інформації про товари. Однак, стандартний функціонал Unity не підтримує безпосереднє зчитування даних з файлів Excel (.xlsx), що ускладнює

інтеграцію даних у проект. Вихід з цієї ситуації знайшли у використанні формату .csv (Comma-Separated Values), який є універсальним і легким для обробки.

Переваги використання формату .csv

Сумісність: Формат .csv є текстовим форматом, який може бути легко зчитаний і оброблений практично будь-якою мовою програмування, включаючи C#, що використовується в Unity. Це дозволяє без проблем інтегрувати дані з бази даних у симулятор.

Простота у використанні: Дані у .csv файлах організовані у вигляді простих текстових рядків, розділених комами, що робить їх читання та редагування зручним. Це дозволяє легко оновлювати інформацію про товари, додаючи або змінюючи рядки у файлі без необхідності користуватися складними інструментами.

Легкість інтеграції: Unity має вбудовані можливості для роботи з текстовими файлами, що дозволяє розробникам швидко реалізувати функціонал зчитування даних з .csv. За допомогою кількох рядків коду можна отримати доступ до даних, розділити їх на компоненти і зберегти у відповідних структурах даних, таких як масиви або списки.

Економія ресурсів: Обробка .csv файлів зазвичай вимагає менше системних ресурсів порівняно з файлами Excel, що особливо важливо для мобільних платформ або низькопотужних пристроїв.

4.3 State-based система для регулювання інтерфейсу Shoop-Keep

За регулювання сторінок в цій системі у мене відповідає класс public class PagesManager **:** MonoBehaviour Для реггулювання інтерфейсу я використав декілька основних моментів , почнемо по порядку

1.Паттерн Signleton **,** що дає можливість отримати доступ до компонента в Сцені з будь-якого скрипта ,

Для цього ми робимо статичний референс на самого себе у самому ж скрипті

public static PagesManager Singleton;

а потім в методі Awake() , що є основним методом Unity , який викликається , ще навіть перед тим як запустилася сцена , ми даємо референс на самого себе , і так як це

статичний референс і він має публічний подифікатор доступу , будь-який скрипт в сцені має до цього компонента доступ.

```
private void Awake()
{
   Singleton = this;
   SwitchPage(Page.code);
}
```

Також подібна ж система використовується і в системі MIHI T-400.

4.4 Реалізація зчитування даних з .csv у Unity

Для реалізації зчитування даних з .csv у Unity можна використовувати простий скрипт

```
public class CSVReader
{
   static string SPLIT_RE = @",(?=(?:[^""]*""[^""]*"")*(?![^""]*""))";
   static string LINE_SPLIT_RE = @"\r\n|\n\r|\n|\r";
   static char[] TRIM_CHARS = { '\"' };
   public static List<Dictionary<string, object>> Read(string file)
   {
      var list = new List<Dictionary<string, object>>();
      TextAsset data = Resources.Load(file) as TextAsset;
      var lines = Regex.Split(data.text, LINE_SPLIT_RE);
      if (lines.Length <= 1) return list;
      var header = Regex.Split(lines[0], SPLIT_RE);
      for (var i = 1; i < lines.Length; i++)
      {
         var values = Regex.Split(lines[i], SPLIT_RE);
         if (values.Length == 0 || values[0] == "") continue;

         var entry = new Dictionary<string, object>();
         for (var j = 0; j < header.Length && j < values.Length; j++)
         {
            string value = values[j];
            value = value.TrimStart(TRIM_CHARS).TrimEnd(TRIM_CHARS).Replace("\\", "");
            object finalvalue = value;
            int n;
            float f;
            if (int.TryParse(value, out n))
            {
               finalvalue = n;
            }
            else if (float.TryParse(value, out f))
            {
               finalvalue = f;
            }
            entry[header[j]] = finalvalue;
         }
         list.Add(entry);
      }
      return list;
   }
```

4.5 Створення UI для системи MINI-T400

Далі я хочу продемонструвати , як було відтворено дизайн цієї касової системи .

Рисунок 4.6 – Зображення UI симулятору MINI T-400

На цьому зображенні ви можете побачити реалізацію юзер-інтерфейсу для системи MINI-T400. Для цієї системи я обрав більш реалістичне візуальне оформлення, щоб дати потенційному працівнику якомога краще уявлення про те, з чим він буде працювати під час виконання справжньої роботи.

Тут ми можемо побачити екран симулятора, кнопки та панель, де буде відображено потенційний чек з операціями, які ми будемо виконувати. У кожної кнопки є основні фідбеки. Наприклад, коли користувач наводить мишку на кнопку, програється звук, а сама кнопка трішки змінює масштаб, що дає розуміння, що вона є “клікабельною”.

4.6 Створення підсказок для користування системами

Для візуального оформлення інтерактивних підсказок , необхідно чітко продумати їх візуальне оформлення. Найефективнішим варіантом буде простий та зрозумілий дизайн. На рисунку 4.7 зображено , як вони будуть виглядати.

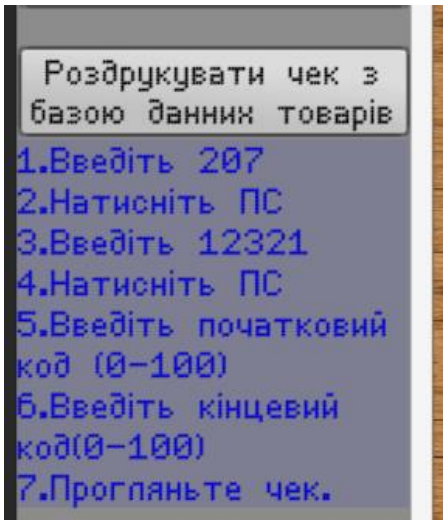

Рисунок 4.7 – Зображення візуальної реалізації підсказок

Далі давайте розглянемо код , який відповідає за демонстрацію їх у UI.

Це простий MonoBehaviour-скрипт для показу підказок. Він виконує функцію керування двома референсами кнопок (UI Button Component) та відповідно регулює вмикання й вимикання підказок для користувача.

Цей код підписується до івента OnClick UI кнопки та відповідно , або демонструє , або ховає підсказку.

```
public class ClueButton : MonoBehaviour
{
    public GameObject clueUi;
    public Button closeButton;
    public Button activateButton;

    public void OnEnable()
    {
        closeButton.onClick.AddListener(CloseClue);
        activateButton.onClick.AddListener(EnableClue);
        closeButton.gameObject.SetActive(false);
        activateButton.gameObject.SetActive(true);
        clueUi.SetActive(false);
    }
    public void OnDisable()
    {
        closeButton.onClick.RemoveListener(CloseClue);
        activateButton.onClick.RemoveListener(EnableClue);
    }
    public void CloseClue()
    {
        closeButton.gameObject.SetActive(false);
        activateButton.gameObject.SetActive(true);
        clueUi.SetActive(false);
```

```
    }
    public void EnableClue()
    {
        closeButton.gameObject.SetActive(true);
        activateButton.gameObject.SetActive(false);
        clueUi.SetActive(true);
    }
}
```

Також тут я використовую Unity-івенти, такі як OnEnable() і OnDisable(), які викликаються, коли GameObject, до якого прикріплений цей скрипт, стає активним (OnEnable()) або неактивним (OnDisable()).

4.7 Адаптування симулятору під різні конфігурації можливих систем

Для створення інтерфейсу я використав низку вбудованих в Unity інстументів для адаптування інтерфейсу під різні системи та розміри моніторів. Unity пропонує безліч вбудованих інструментів, зокрема Rect Transform, Canvas Scaler, Horizontal Layout Group, Vertical Layout Group та Grid Layout Group , що дозволяє ефективно прототипувати інтерфейс.

Основний компонент, що відповідає за позиціонування та масштабування елементів, — це Rect Transform. Він дозволяє задавати межі об’єктів інтерфейсу, прив'язуючи їх до різних частин екрану. Завдяки гнучким параметрам якорів (anchors) і прив’язок (pivot points), можна створити інтерфейс, який динамічно змінюється відповідно до розмірів екрана.

Щоб забезпечити відповідне масштабування інтерфейсу на пристроях із різною щільністю пікселів, застосовується компонент Canvas Scaler, який змінює розмір усіх UI-елементів відносно розмірів екрана. Цей компонент має різні режими роботи: Constant Pixel Size, Scale with Screen Size, та Constant Physical Size, що дозволяє адаптувати інтерфейс для мобільних пристроїв, планшетів та десктопів.

Компоненти Horizontal Layout Group та Vertical Layout Group полегшують вирівнювання елементів по горизонталі та вертикалі, дозволяючи створити динамічний інтерфейс, який автоматично налаштовує свої елементи. Grid Layout Group, у свою чергу, дозволяє організувати компоненти у вигляді сітки, що підходить для створення

інтерактивних меню або відображення великої кількості об’єктів із збереженням однакового розташування.

Комбіноване використання цих інструментів дозволяє розробити адаптивний інтерфейс, який забезпечує зручне користування незалежно від типу пристрою чи розміру екрана, гарантуючи цілісність користувацького досвіду.

Коли потрібно зберегти певне співвідношення сторін окремих елементів, компонент Aspect Ratio Fitter стає незамінним. Він дозволяє автоматично коригувати розміри елемента так, щоб зберегти задане співвідношення, що особливо важливо для відображення зображень, відео або кнопок з фіксованими пропорціями. Використовуючи цей інструмент у комбінації з Rect Transform та Canvas Scaler, можна досягти високої точності у відтворенні естетики інтерфейсу на різних пристроях.

Сучасні мобільні пристрої часто змінюють орієнтацію між портретною та ландшафтною. Для забезпечення коректного відображення UI-елементів у обох режимах можна реалізувати динамічне налаштування якорів або створити окремі макети для кожного варіанту. Це допомагає уникнути розтягування або надмірного стиснення елементів і забезпечує комфортне використання додатка незалежно від орієнтації. При створенні складних інтерфейсів, де використовується велика кількість елементів, важливо врахувати вплив анімацій та постійних змін розміру екрана на продуктивність. Оптимізація роботи UI за допомогою методів пулінгу об’єктів (object pooling) та мінімізації кількості перерахунків макету (layout recalculations) сприяє зменшенню навантаження на систему та забезпечує плавність взаємодії користувача з додатком. Оскільки не всі пристрої мають однакові характеристики, обов’язковим кроком є тестування інтерфейсу на різних екранах та роздільних здатностях. Використання емуляторів та фізичних пристроїв допомагає виявити можливі проблеми, пов’язані з невідповідністю розміру, розташуванням елементів або некоректним масштабуванням, що дозволяє своєчасно їх виправити.

Комбінування цих методів разом із стандартними компонентами Unity дає змогу створити адаптивний, зручний та естетично привабливий інтерфейс, який відповідає вимогам сучасних користувачів незалежно від платформи чи типу пристрою.

4.8 State-based система для регулювання інтерфейсу МІНІ – 400

Подібно до того, як вже було описано систему Shoop-keep, в симуляторі МІНІ–400 було також реалізовано state-based підхід для керування інтерфейсом. Принципи в обох випадках залишаються подібними, але й водночас є важливі відмінності, адже ці системи мають абсолютно різну функціональність і призначення.

Було вирішено використати той самий патерн Singleton зі статичним референсом, який дозволяє зручно отримувати доступ до головного компонента інтерфейсу з будь-якої частини системи. Це суттєво спрощує архітектуру й дозволяє уникати надмірної кількості зв’язків між компонентами.

Також було взято до уваги користувачів, які вперше взаємодіють із системою. Для цього на бокових панелях екрану було розміщено підказки з доступними командами. Під час розробки було важливо, щоб інтерфейс залишався інтуїтивним, але водночас давав змогу користувачу самостійно дослідити функціонал системи. Це не тільки додає занурення у симуляцію, а й дозволяє користувачу відчути себе активним учасником, а не просто спостерігачем.

Розробляючи цю частину, було важливо зберегти баланс між функціональністю та зручністю використання, і в результаті вийшло так, що обрана структура дозволила досягти цього.

З доступних команд тут є

Одиночна покупка товару - Ця команда дозволяє змоделювати стандартний сценарій купівлі одного товару. Коли користувач вводить відповідну команду, інтерфейс перемикається в стан очікування штрих-коду товару, після чого підтверджується транзакція. Я намагався зробити цей процес максимально наближеним до реального POS-терміналу.

Z-звіт – це фінальний звіт зміни, після якого обнуляється підсумок продажів. У симуляторі я реалізував механізм генерації цього звіту лише після проходження повного циклу роботи касира. Це допомагає відчути, як працює закриття зміни в реальному середовищі.

Друк X-звіту.На відміну від Z-звіту, X-звіт дозволяє переглянути поточні підсумки продажів без їх обнулення. Для реалізації я використав окремий стан системи, що

дозволяє переглядати статистику без зміни даних. Це було важливо для демонстрації різниці між двома типами звітів.

Друк бази даних доступних товарів.Ця команда викликає виведення списку всіх товарів, які доступні в системі на поточний момент. Її було реалізовано як зручну функцію для перевірки або навчання, особливо коли користувачі тестують, як працює покупка чи множення кількості товару.

Службове внесення.Дозволяє вручну внести певну суму в касу без прив'язки до продажу. У реальному житті це часто використовується, наприклад, при старті робочого дня. У симуляторі було зроблено акцент на безпечному збереженні цієї операції в логах і в системній пам’яті.

Увімкнення касового апарату.Це одна з перших операцій, яку потрібно зробити для початку роботи та її було позначено як ключовий стартовий стан, без якого інші команди недоступні. Таким чином, користувач одразу бачить, що апарат має пройти ініціалізацію перед взаємодією.

Реєстрація касира. Після вмикання апарату, необхідно зареєструвати касира. У системі МІНІ–400 було впроваджено механізм вибору імені з умовної бази даних або створення нового, щоб показати важливість ідентифікації в системах обліку.

ВИСНОВОК

У ході виконання дипломної роботи «Реалістичний симулятор кассового апарату в Unity» було досягнуто поставлених цілей та завдань, що підтверджується як теоретичним аналізом існуючих рішень, так і практичною реалізацією симулятора. Розроблений тренувальний симулятор успішно відтворює основні операції касових апаратів, зокрема моделей Т-400 та ShoopKeep beta, що дозволяє майбутнім операторам отримати практичний досвід роботи в безпечних умовах.

Значну увагу було приділено інтеграції інноваційних технологій та паттернів програмування, таких як Object Pool, Singleton, Observer, Factory, Strategy та інші. Використання цих підходів дозволило створити гнучку та масштабовану архітектуру симулятора, що забезпечує високу продуктивність, оптимізацію ресурсів та зручність подальшої підтримки та розширення функціоналу.

Особливе місце в роботі займає розробка інтерфейсу користувача, який за допомогою інтерактивних фідбеків, аудіо та візуальних ефектів, створює максимально реалістичну модель робочого процесу. Впровадження компонентів, таких як SFX Manager з використанням Audio Source, значно підвищує інтуїтивність та зручність взаємодії з симулятором, що є важливим аспектом у навчальному процесі.

Отримані результати свідчать про практичну цінність та ефективність використання тренувального симулятора як засобу підготовки нових співробітників до роботи на реальному обладнанні. Розроблена система має великий потенціал для подальшого впровадження у навчальні процеси як у закладах освіти, так і у виробничих компаніях, що працюють у сфері торгівлі та обслуговування.

Таким чином, дипломна робота підтверджує можливість ефективного використання сучасних технологій розробки для створення інтерактивних симуляторів, що сприяють підвищенню якості навчання та швидкій адаптації персоналу до умов реальної роботи. Рекомендовано подальше вдосконалення симулятора через інтеграцію додаткових сценаріїв роботи, розширення функціональних можливостей та адаптацію до нових технологічних стандартів, що дозволить ще більше збільшити його практичну цінність. Подальші напрямки дослідження відкривають широкі можливості для

вдосконалення розробленого симулятора. Серед них можна виділити інтеграцію додаткових сценаріїв, які відображають специфіку роботи різних типів касових апаратів, а також розширення функціоналу шляхом впровадження модулів для аналізу продуктивності роботи операторів. Це дозволить не лише адаптувати симулятор до змінних умов ринку, але й забезпечити більш глибоке навчання співробітників.

Важливим є також розгляд можливості інтеграції симулятора з існуючими навчальними платформами та системами управління персоналом. Такий підхід дозволить створити єдину систему контролю за навчальним процесом, де результати роботи симулятора можуть автоматично аналізуватись і використовуватись для покращення програм якими користуються співробітники.

Отримані результати демонструють, що застосування сучасних технологій розробки, перевірених патернів програмування та інтерактивних методів навчання може значно покращити якість підготовки кадрів у сфері торгівлі та обслуговування. Розроблений симулятор не лише дозволяє швидко освоїти базові навички роботи з касовими апаратами, але й створює умови для безпечного експериментування, що особливо актуально сучасних працівників які працюють з POS-системами.

Таким чином, дипломна робота підтверджує ефективність використання інтерактивних симуляторів як засобу підготовки кваліфікованих співробітників. Рекомендовано продовжувати дослідження в цьому напрямку, розширюючи функціональні можливості симулятора та впроваджуючи новітні технології для покращення користувацького досвіду. Це забезпечить не лише високий рівень навчання, але й сприятиме сталому розвитку інноваційних рішень у сфері автоматизації бізнес-процесів

СПИСОК ЛІТЕРАТИ

1. Борромео Н.А. Розробка ігор на Unity 2020: Створення, налаштування та оптимізація професійних ігор за допомогою Unity 2020 та C#. Бірмінгем: Packt Publishing, 2020. – 580 с.
2. Гаркуша С. В., Ольховська О. В., Черненко О. О. Методичні рекомендації до виконання кваліфікаційної роботи для здобувачів вищої освіти першого (бакалаврського) рівня спеціальності 122 «Комп'ютерні науки». Полтава: ПУЕТ, 2022. 68 с.
3. Ферроне Х. Вивчаємо C# через розробку ігор на Unity / Х. Ферроне. – 5-е вид. – Київ : Діалектика, 2023. – 352 с.
4. Хокінг Дж. Єдність в дії: Мультиплатформна розробка ігор на C#. Острів притулку (NY): Manning Publications, 2022. – 416 с.
5. Барон Д. Патерни розробки ігор з Unity 2021. Бірмінгем: Packt Publishing, 2021. – 246 с.
6. Васильєв О. М. *Програмування на C# для початківців. Особливості мови.* – Київ: ТОВ «Видавництво "Альфа-Прес"», 2021. – 416 с.
7. Ферроне Х. Вивчаємо C#, розробляючи ігри з Unity 2020. 5-ий вид. Бірмінгем: Packt Publishing, 2020. – 366 с.
8. Албахарі Дж. C# 12 у двох словах: The Definitive Reference / Дж. Албахарі. - 1-е вид., перероб. і доп. - Себастополь : O'Reilly Media, 2023. - 1083 с.
9. Smith M., Ferns S., Murphy S. *Unity Cookbook: Over 160 Recipes to Craft Your Own Masterpiece in Unity 2023.* 5-те вид. Birmingham: Packt Publishing, 2023. – 780 с.
10. Godbold A. *Mastering UI Development with Unity: Develop engaging and immersive user interfaces with Unity.* 2-ге вид. Birmingham: Packt Publishing, 2024. – 638 с.
11. Мартін Р. *Чиста архітектура. Мистецтво розробки програмного забезпечення.* – Харків: Фабула, 2022. – 352 с. –

12. Ferrone H. *Learning Design Patterns with Unity: Learn the Secret of Popular Design Patterns while Building Fun, Efficient Games in Unity 2023 and C#.* Birmingham: Packt Publishing, 2024. – 676 с.
13. Кемерон С.Х. Unity 2022 на прикладі: Посібник для створення 2D та 3D ігор, вдосконалених для AR, VR та доповненої реальності. Бірмінгем: Packt Publishing, 2024. - 596 с.
14. *C# documentation.* – Microsoft Learn, 2024. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 09.05.2025).
15. Олексюк В.П., Джуга Д.Ю., Мельник П.П. Розробка гри «Симулятор студента»: Від концепції до коду. Матеріали семінару CEUR (Proc. CSSESW 2024), 2024, с. 89-109.
16. Lenard D. Designing Point-of-Sale Systems. *UXmatters*, 2018. URL: https://www.uxmatters.com/mt/archives/2018/05/designing-point-of-sale-systems.php (дата звернення: 10.05.2025).
17. Lenard D. User Experience Barriers in POS Systems. *UsabilityGeek*, 2018. URL: https://usabilitygeek.com/ux-barriers-in-point-of-sale-systems/ (дата звернення: 10.05.2025).
18. Keenan M. POS System Design Principles to Know. *Shopify Retail Blog*, 2024. URL: https://www.shopify.com/retail/pos-system-design (дата звернення: 10.05.2025).
19. Dyer N. What Is POS UI? Design Principles and Extensions. *Shopify Retail Blog (India)*, 2024. URL: https://www.shopify.com/in/retail/pos-ui (дата звернення: 10.05.2025).
20. Polack T. A Guide to Systems-Based Game Development. *Game Developer*, 2018. URL: https://www.gamedeveloper.com/design/a-guide-to-systems-based-game-development (дата звернення: 10.05.2025).
21. Rivello S.A. MVC Architecture for Unity: A Journey of Structured Game Development. *Medium*, 2024. URL: https://samuel-asher-rivello.medium.com/mvc-architecture-for-unity-a-journey-of-structured-game-development-87680aa3baca (дата звернення: 10.05.2025).

22. Канер С. Тестування програмного забезпечення / С. Канер. – 2-ге вид. – Київ : Вільямс, 2006. – 672 с.
23. Kalms M. How to Architect Code as Your Project Scales. *Unity (блог)*, 2023. URL: https://unity.com/how-to/how-architect-code-your-project-scales (дата звернення: 10.05.2025).
24. Бонд Дж. Unity і C#. Геймдев від ідеї до реалізації / Дж. Бонд. – 2-е вид. – Харків : Osmo, 2020. – 928 с. – ISBN 978-5-4461-0715-5.
25. Unity Technologies. UI systems [Електронний ресурс] : Unity Manual. – Режим доступу: https://docs.unity3d.com/Manual/UIToolkits.html (останній перегляд 09.05.2025).
26. Unity Technologies. C# Code Style Guide (Unity 6 edition) [Електронний ресурс] : Unity. – Режим доступу: https://unity.com/resources/c-sharp-style-guide-unity-6 (останній перегляд 09.05.2025).
27. **Unity Technologies.** Level up your code with design patterns and SOLID [Електронний ресурс] : електронний навчальний посібник. – Unity Technologies, 2024. – 147 с. – Режим доступу: https://unity.com/resources/design-patterns-solid-ebook?isGated=false (останній перегляд 09.05.2025).
28. Яблонський Дж. Закони UX-дизайну / Дж. Яблонський. – Себастополь : O'Reilly Media, 2020. – 160 с.
29. Унгер Р., Чендлер К. UX-дизайн. Практичне керівництво з проєктування досвіду взаємодії / Р. Унгер, К. Чендлер; пер. з англ. – СПб. : Символ-Плюс, 2011. – 336 с. – ISBN 978-5-93286-184-4. – Режим доступу: https://profibooks.org/ua/p2229074887-dizajn-prakticheskoe-rukovodstvo.html?source=merchant_center… (останній перегляд 09.05.2025).
30. Дацко М. А. Моделювання складних об'єктів. / М. А. Дацко. – М. : Максімас, 2015. – 111 с.