


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ЗАКЛАД «ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА»
Навчально-науковий інститут математики та інформаційних технологій

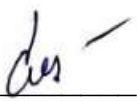
Кафедра фізико-технічних систем та інформатики

Іванов Олександр Олександрович

**Використання сучасних інструментів при розробці ігрового
проєкту (на основі Unity3D)**

Магістерська робота
за спеціальністю 122 «Комп'ютерні науки»

Особистий підпис – 

Науковий керівник –  к.п.н., доцент, О.О. Смагіна
(підпис) (науковий ступінь, наукове звання, ініціали, прізвище)

Зав. кафедри – _____ д.т.н., проф., Ю.Г. Козуб
(підпис) (науковий ступінь, наукове звання, ініціали, прізвище)

АНОТАЦІЯ

Іванов О.О.

Тема: Використання сучасних інструментів при розробці ігрового проєкту (на основі Unity3D)

Спеціальність: 122 «Комп'ютерні науки»

Установа: ДЗ «ЛНУ імені Т.Шевченка», 2024.

Магістерська робота містить: 72 с., 44 рис., 8 табл., 10 додат., 32 джерела.

Об'єкт дослідження – процес створення ігрового проєкту - симулятора на основі кросплатформеного рушія Unity для використання в освітньому процесі закладу П(ПТ)О при підготовці кваліфікованих робітників.

Предмет дослідження – методи та засоби створення симулятора – додатка навчального призначення для використання на ПК під управлінням ОС Win x64.

Мета роботи – розробка функціональної моделі симулятора навчального призначення, його програмна реалізація та апробація в освітньому процесі.

Результати роботи. Досліджено парадигму навчання засновану на використанні ігрових проєктів навчального призначення, геймдизайн та моделі розробки ігрових проєктів. Розроблено функціональну модель програми-симулятора навчального призначення. За допомогою методу оціночних матриць обрано обрано та застосовано гібридну методологію розробки. В середовищі розробки Unity із тривимірних моделей створених у Blender 3D та із застосуванням мови програмування C# виконано розробку симулятора. Наведено техніко-економічне обґрунтування проєкту.

Висновок. Розроблено симулятор збирання системного блоку для використання в освітньому закладу П(ПТ)О.

Ключові слова. ГЕЙМІФІКАЦІЯ, СИМУЛЯТОР, ФУНКЦІОНАЛЬНА МОДЕЛЬ, UNITY, ІГРОВИЙ ПРОЄКТ.

ABSTRACT

Ivanov O.

Subject: Using modern tools when developing a game project (based on Unity3D)

Specialty: 122 "Computer Science"

Institution: SI "LNU named after T. Shevchenko", 2024.

The master's thesis contains: 72 pages, 44 figures, 8 tables, 10 appendices, 32 sources.

The object of research is the process of creating a game project - a simulator based on the cross-platform Unity engine for use in the educational process of the professional education institution in the training of qualified workers.

The subject of research - is methods and means of creating a simulator - an educational application for use on a PC running the Win x64 OS.

The main purpose of the article is to develop a functional model of an educational simulator, its software implementation and testing in the educational process.

Work results. This paper has clearly shown that the learning paradigm based on the use of game projects for educational purposes, the game design and the model of game development of game projects was studied. A functional model of an educational simulator program has been developed. Using the method of evaluation matrices, a hybrid development methodology was selected and applied. A simulator was developed in the Unity development environment from three-dimensional models created in Blender 3D and using the C# programming language. The technical and economic justification of the project is presented.

Conclusion. A system unit assembly simulator has been developed for use in an educational institution professional education.

Key words. GAMIFICATION, SIMULATOR, FUNCTIONAL MODEL, UNITY, GAME PROJECT.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

| | |
|--------|--|
| ВПУ | - вище професійне училище |
| МОНУ | - міністерство освіти і науки України |
| ОС | - операційна система |
| П(ПТ)О | - професійна(професійно-технічна) освіта |
| ПЗ | - програмне забезпечення |
| ПК | - персональний комп'ютер |
| ПУ | - програмне управління |
| ЦП | - центральний процесор |
| DPE | - design, play and experience |
| GBL | - game based learning |
| MDA | - mechanics, dynamics and aesthetics |

ЗМІСТ

| | |
|---|----|
| ВСТУП..... | 6 |
| РОЗДІЛ 1. Теоретичні основи ігрових ком'ютерних проєктів навчального призначення..... | 9 |
| 1.1 Парадигма навчання заснована на використанні ігрових проєктів навчального призначення..... | 9 |
| 1.2 Дизайн та моделі розробки ігрових проєктів..... | 13 |
| 1.3 Програми-симулятори навчального призначення як різновид ігрового програмного забезпечення. Їх місце та роль в освіті України..... | 19 |
| 1.4 Висновки до розділу 1..... | 22 |
| РОЗДІЛ 2 Аналіз методологій та засобів розробки..... | 24 |
| 2.1 Огляд існуючих на ринку аналогічних програмних продуктів.... | 24 |
| 2.2 Розробка функціональної моделі програми-симулятора навчального призначення..... | 25 |
| 2.3 Вибір методології розробки та моделі життєвого циклу..... | 32 |
| 2.4 Аналіз та вибір оптимальних інструментальних засобів планування та розробки..... | 41 |
| 2.5 Висновки до розділу 2..... | 46 |
| РОЗДІЛ 3 Практична реалізація програми-симулятора на рушії Unity 3D..... | 47 |
| 3.1 Розробка ескізів вікон програми..... | 47 |
| 3.2 Створення моделей елементів корпусу та комплектуючих..... | 50 |
| 3.3 Механізм реалізації логіки симулятора..... | 53 |
| 3.4 Програмування, збірка та тестування симулятора..... | 58 |
| 3.5 Техніко-економічне обґрунтування розробки симулятора..... | 66 |
| 3.6 Висновки до розділу 3..... | 69 |
| ЗАГАЛЬНІ ВИСНОВКИ..... | 70 |

| | |
|---|-----|
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... | 74 |
| Додаток А. Діграма Ганта..... | 79 |
| Додаток Б. Лістинг програмного коду MainMenu.cs..... | 80 |
| Додаток В. Лістинг програмного коду SpawnObject.cs..... | 81 |
| Додаток Г. Лістинг програмного коду AnimationControl.cs..... | 85 |
| Додаток Д. Лістинг програмного коду Inventory.cs..... | 86 |
| Додаток Е. Лістинг програмного коду ActiveStatus.cs..... | 88 |
| Додаток Ж. Лістинг програмного коду DeleteObject.cs..... | 89 |
| Додаток З. Лістинг програмного коду CheckBuild.cs..... | 91 |
| Додаток И. Лістинг програмного коду PauseMenu.cs..... | 98 |
| Додаток К. Результат розрахунку трудомісткості проєкту в онлайн-калькуляторі СОСОМО 2..... | 100 |

ВСТУП

В умовах тривалої війни та карантинних заходів, в яких Україна перебувала, різко збільшилося використання інформаційних технологій в освіті та зросло їх значення для належного забезпечення освітнього процесу. Значного поширення набули системи управління навчанням, зокрема Moodle, Google Workspace, Microsoft Teams, різноманітні системи контролю знань, в основному тестові, наприклад Kahoot, На урок, Hot Potatoes.

Великий вибір програмних засобів та платформ переважно хмарного типу певним чином допоміг забезпечити реалізацію освітнього процесу шляхом використання технологій віддаленого (дистанційного) навчання[1]. Однак, в кожній галузі така екстремальна цифровізації виявила ряд проблем та суперечностей, зокрема проблемною виявилась реалізація практичних видів занять. Якщо заклади загальної середньої освіти більш-менш пристосувались до нових реалій, то для закладів системи професійної (професійно-технічної) освіти та фахової передвищої освіти саме реалізація практичних видів підготовки стала викликом та справжньою проблемою.

Виконання освітніх та освітньо-професійних програм підготовки кваліфікованих робітників та фахових молодших бакалаврів передбачає виконання різномовних видів практичної підготовки таких, як виробниче навчання (в майстернях та на виробництві), виробничої, навчальної, технологічної, переддипломної практики тощо. В системі професійної (професійно-технічної) освіти кількість практичних робіт навіть з предметів професійно-теоретичної підготовки може становити до 50-70% від загальної кількості годин на предмет. Тобто підготовка кваліфікованих робітників є практико-орієнтованою, оскільки забезпечує здобуття професійних кваліфікацій, що переважно відповідають 3-4 рівням національної рамки кваліфікацій і передбачають виконання трудових функцій з використанням інструментів, механізмів, пристосувань, обслуговування техніки та агрегатів,

тобто тих, де велике значення мають моторні дії, формування вмінь та навичок.

Якщо для ряду професій, наприклад класифікаційного угруповання 41 «Службовці пов'язані з інформацією», можна використовувати певні програмні засоби, зокрема хмарні, для освітнього процесу і робити це доволі успішно, хоча і без повноцінного забезпечення професійно-практичної складової підготовки, то для більшості інших професій переважно відсутні програмні засоби, які б могли забезпечити формування професійних компетентностей, передбачених державними освітніми стандартами професійно-технічної освіти.

Починаючи з 2019 року із введення карантинних заходів автором як членом педагогічного колективу ВПУ № 7 м.Кременчука, спільно із колегами здійснювався пошук програмних засобів, які могли бути використані під час дистанційного навчання для проведення уроків виробничого навчання та практичних робіт.

Найбільш вдалимими б для цього могли стати симулятори – програмні засоби, що реалізують імітацію справжніх трудових дій, технологічних процесів, обслуговування та керування устаткуванням, машинами та механізмами. Найбільш затребуваними виявились симулятори верстатного та зварювального устаткування, електромонтажних робіт, опоряджувальних, столярних та інших видів будівельних робіт, які могли б бути встановленими на ПК або мобільні пристрої (смартфони, планшети) або виявились би хмарними сервісами. На жаль, попри велику кількість наукових публікацій, постійне проведення конференцій та вебінарів щодо цифровізації освіти, зокрема професійної (професійно-технічної), реальні програмні розробки, що відповідали б рівню здобувачів освіти, начальним програмам, були б доступні закладам П(ПТ)О майже відсутні.

Отже, можна констатувати гострий брак симуляторів навчального призначення в освітньому середовищі України та потребу у їх розробці. З 2022 року було розпочато розробку симулятора на основі ігрового рушія для

збирання системного блоку ПК, який був би придатний для використання як на уроках професійно-теоретичної так і загальноосвітньої підготовки для широкого кола професій, що і було покладено в основу представленого дослідження.

Отже, об'єктом дослідження є процес створення ігрового проєкту - симулятора на основі кросплатформеного рушія Unity для використання в освітньому процесі закладу професійної(професійно-технічної) освіти при підготовці кваліфікованих робітників.

Предметом дослідження є методи та засоби створення симулятора – додатка навчального призначення для використання на ПК під управлінням ОС Win x64.

Метою роботи є розробка функціональної моделі симулятора навчального призначення, його програмна реалізація та апробація в освітньому процесі.

При проведенні дослідження використано такий загальнонауковий метод як функціональний аналіз та імперичні методи спостереження, експеримент, аналіз, порівняння.

Апробацію розробки виконано у 2022-2023 та 2023-2024 навчальних роках у Вищому професійному училищі № 7 м.Кременчука Полтавської області на уроках професійно-теоретичної та загальноосвітньої підготовки учнів за професіями загальними для усіх галузей економіки, будівельного та електротехнічного профілю.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ІГРОВИХ КОМ'ЮТЕРНИХ ПРОЄКТІВ НАВЧАЛЬНОГО ПРИЗНАЧЕННЯ

1.1 Парадигма навчання заснована на використанні ігрових проєктів навчального призначення

Швидка інформатизація освіти призвела до переосмислення ролі як освіти в цілому, так і процесу навчання зокрема. Старі форми і методи, що забезпечували досягнення результатів навчання, сьогодні стають все менше ефективними.

У книзі Елфіна Тофлера «Шок майбутнього» (1970 рік) система освіти описана як така, що здійснює «штампування людей, придатних для виживання у системі, яка помре раніше, ніж вони самі», а знання порівнюється з продуктом нетривалого зберігання [2]. Такі футуристичні думки могли б шокувати наших вітчизняних педагогів періоду 70-90 років, але сьогодні варто визнати, що без докорінної зміни у підходах до функціонування системи освіти, методів та засобів які вона використовує, ми не зможемо готувати конкурентно-здатних фахівців, не залежно від рівня та галузі освіти.

У своїй роботі «Після трьох уже пізно» Масару Ібука зазначає доцільність та перспективність надання здобувачам досвіду отримання нових знань через ігрові форми [3]. Необхідно зауважити, що це стосується не тільки дітей та підлітків, які здобувають формальну інституційну освіту, а й навчання дорослого населення, у т.ч. підготовки кадрів для підприємств та організацій.

Ще одна вимога до сучасного навчання та освіти в цілому описується в книзі Ікуджіра Нонака та Хіротакі Такеучі «Компанія - творець знання. Зародження і розвиток інновацій в японських фірмах». Одна з думок, що описується у книзі полягає у тому, що створення нового знання вже не можливе лише шляхом механічного опрацювання інформації, що створення знання нової якості залежить від ряду характеристик (явних і не явних) тих, хто це знання створює. Фактично автори описують творчу, новаторську

діяльність людини або креативність[4]. Не зважаючи на те, що мова у книзі йде про роботу у компаніях, ці ідеї є цілком придатними до застосування в освіті.

Доцентка Національного університету «Одеська політехніка» Тетяна Лугова зазначає, що «на перший план у смисловому полі «знання» виходять «досвід» та «креатив», що є центральними в ідеології гейміфікації»[5].

Відомий теоретик гейміфікації навчання Джеймс Пол Гі вважає, що сьогодні дотримання правил логіки і обчислень не є важливими для функціонування мозку людини. Сучасна теорія пізнання припускає, що люди навчаються через досвід. Мозок людини зберігає інформацію про все те, що людина пережила і це має прямий вплив на процес пізнання. Якщо виходити із тверджень Джеймса Пола Гі, то для створення сприятливих умов навчання необхідно створити хороші умови отримання досвіду[6].

Під гейміфікацією розуміють «використання ігрових практик та механізмів у неігровому контексті для залучення кінцевих користувачів до розв'язання проблем»[7].

Гейміфікація в навчанні - це застосування елементів гри в освітньому процесі для стимулювання мотивації, підвищення зацікавленості та полегшення засвоєння знань учнями. Цей підхід використовує механізми та прийоми, які характерні для ігор, у навчальних цілях.

Основні аспекти гейміфікації включають:

- а) використання точних правил, завдань, рівнів складності, нагород та інших елементів, характерних для ігор, у навчальних процесах;
- б) створення ситуацій, де учні отримують задоволення від навчання через гру, що мотивує їх активніше займатися освітою;
- в) використання бейджів, рівнів, очок або віртуальних винагород для підвищення зацікавленості та стимулювання досягнень;
- г) використання можливостей для співпраці між учнями (командні завдання) або створення змагань, щоб заохотити до активності;
- д) створення індивідуальних шляхів розвитку для кожного учня, враховуючи їхні особисті потреби та можливості.

Отже, освітню гейміфікацію можна визначити, як систему, що складається із ряду елементів, структура якої продемонстрована на рис.1.1.

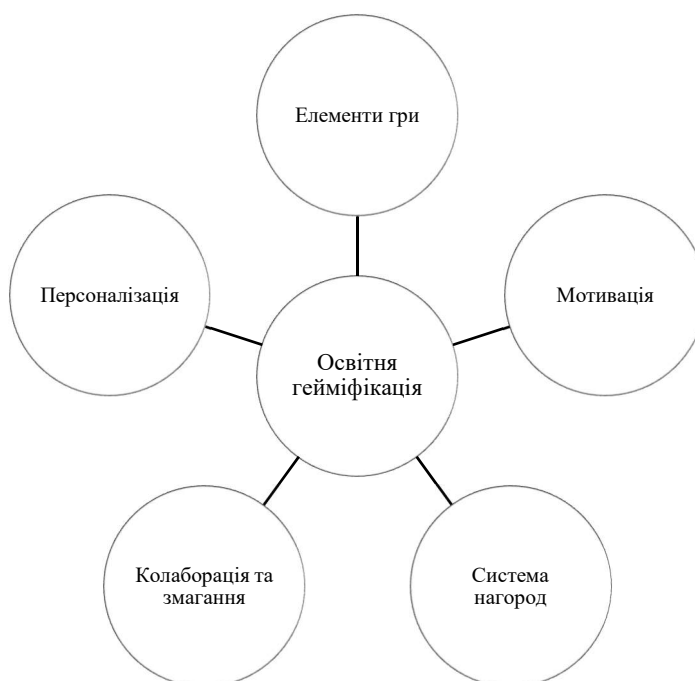


Рис1.1. Складові освітньої гейміфікації

Гейміфікація може бути застосована в різних форматах: від використання спеціальних навчальних платформ з елементами гри до розробки власних ігорних завдань для навчального процесу. Головна ідея - зробити навчання цікавим, захоплюючим та стимулюючим, адаптуючи його під потреби та інтереси здобувачів освіти.

Грунтуючись на вищезазначеному можна сформулювати парадигму навчання через ігрові проєкти як підхід до освіти, коли навчання відбувається через створення і використання ігрових сценаріїв, завдань або проєктів, та передбачає активне залучення здобувачів до процесу навчання шляхом інтерактивних ігор, завдань та проєктів.

Ця парадигма має декілька ключових переваг. По-перше, це підвищення ступеня залученості до процесу навчання. Ігрові проєкти привертають увагу здобувачів і спонукають їх активно взаємодіяти з матеріалом навчання. По-друге, практична спрямованість або навчання через досвід. Ігрові сценарії

дають можливість здобувачам застосовувати отримані знання на практиці шляхом вирішення завдань або проблем в ігровій формі. По-третє, колективна робота, яка сприяє співпраці та взаємодії між здобувачами, адже багато ігрових проєктів передбачають командну роботу. По-четверте, мотивація до навчання, оскільки одночасний процес гри та навчання може стимулювати бажання засвоювати нові знання через цікавий та залучний спосіб. По-п'яте, розвиток навичок, оскільки ігрові проєкти сприяють розвитку широкого спектру навичок, таких як критичне мислення, співпраця, творчість та розв'язання проблем.

Ігрові проєкти можуть мати різні форми - це може бути створення ігрових програм, рольових ігор, віртуальних середовищ або навіть фізичних ігор, спрямованих на досягнення навчальних цілей.

Важлива роль серед інших видів ігрових проєктів відводиться саме комп'ютерним проєктам (програмам).

Один з ключових аспектів досліджень Джеймса Пола Гі - це погляд на ігри та програми як тьюторів. Гі вважає, що ігри та програми можуть виконувати роль ефективних тьюторів, навчаючи людей різноманітним аспектам мови, культури та навіть складних навичок[8]. Він виділяє кілька ключових принципів, які роблять програми та ігри ефективними тьюторами:

- а) програми можуть створювати ситуації, що дозволяють отримувати знання в різних контекстах, що сприяє кращому засвоєнню матеріалу;
- б) використання програм, які можуть враховувати поточний рівень знань учня та адаптувати навчальний матеріал для оптимального навчання;
- в) програми можуть надавати детальний зворотний зв'язок, допомагаючи здобувачам розуміти свої помилки та покращувати навички;
- г) ігрові та програмні середовища дозволяють здобувачам активно взаємодіяти з матеріалом, надаючи їм можливість самостійно досліджувати та вирішувати завдання.

Головна ідея полягає в тому, що ігри та програми можуть бути більш ефективними та привабливими для навчання, забезпечуючи здобувачам

можливість отримати навички та знання через активну участь у відтворюваних ситуаціях та інтерактивних завданнях.

Саме Джеймс Пол Гі заклав основи розвитку нового виду освітніх технологій - Game based learning (GBL), а саме навчання, заснованого на принципах комп'ютерних ігор.

GBL передбачає, що здобувач в процесі гри не вивчає певний предмет/дисципліну, а вчиться виконувати певні професійні дії, наприклад бути зварником, а не вивчати технологію та обладнання зварювального виробництва[9]. Цей вид освітніх технологій дає можливість формувати професійні компетентності в умовах, що максимально наближені до умов реального життя (виробництва, сфери обслуговування тощо), а отже є перспективним при застосуванні в П(ПТ)О.

1.2. Дизайн та моделі розробки ігрових проєктів

Розробка ігрових навчальних проєктів підпорядковується усім правилам створення програмного забезпечення прикладного призначення.

Проектуванням ігрових проєктів займається геймдизайн - «створення правил і цілей гри, проектування ігрового контенту і геймплею»[10].

Проектування комп'ютерних ігрових проєктів фактично не відрізняється від проектування інших типів ігр.

Загальну структуру геймдизайну можна описати схемою поданою на рис.1.2. Вона відповідає стандарту ISO/IEC/IEEE 15288:2023 Systems and software engineering - System life cycle processes [11], тобто задовольняє загальноприйнятим вимогам щодо відповідності основним етапам життєвого циклу розробки програмного забезпечення.

Над теорією геймдизайну активно працює американська дизайнерка Джейн Макгонігал, яка визначила наступні її елементи, а саме[12]: готовність до гри, мету гри, правила, простір та зворотний зв'язок.

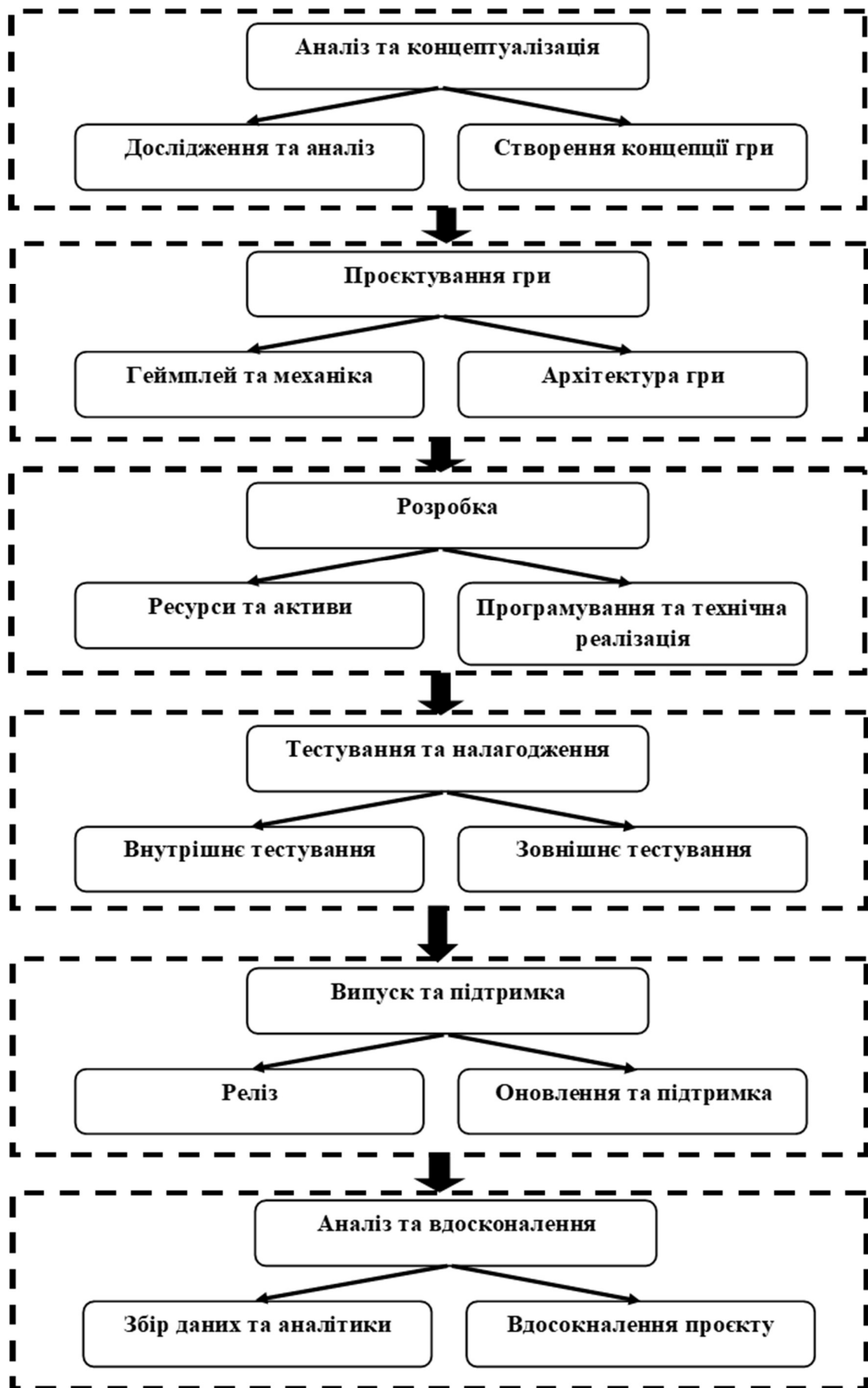


Рис. 1.2. Загальна структура геймдизайну

Джейн Макгонігал під готовністю до гри розуміє добровільність участі, оскільки в її розуміння гра через примус вже є роботою. Щодо ігрових проєктів навчального призначення, які можуть бути обов'язковим компонентом навчально-методичного забезпечення певного предмету, то у цьому випадку можна прийняти той факт, що особа, яка погодилась навчатися за певною професійною програмою вже під час вступу надала свою згоду на використання запропонованих їй методів та засобів навчання.

При створенні цифрової (комп'ютерної) гри Джейн Макгонігал запропонувала свою власну концепцію, що включає наступні етапи:

- а) "Фантазування" (Будування місії та визначення цілей) - визначення теми та місії гри: Що гра буде робити для гравців? Які цілі вона ставить?
- б) "Приготування" (Створення правил і гравців) - розробка правил гри: Які основні механіки та правила будуть у грі?; визначення гравців: Хто буде грати в гру? Які їх потреби та очікування?
- в) "Мобілізація" (Створення контенту та механік) - створення ігрових механік: Які можливості в грі? Які завдання будуть?; створення контенту: Які рівні, завдання, персонажі, світи?
- г) "Розвиток" (Прототипування та тестування) - створення прототипів для перевірки механік та ігрових елементів, перевірка гри на ефективність та задоволення гравців;
- д) "Спільнота" (Залучення та взаємодія гравців) - створення спільноти: Як можна залучити гравців до співпраці або взаємодії?; створення соціальних механік: Як гра може стимулювати спілкування та взаємодію гравців?
- е) "Співтворчість" (Розвиток та поширення гри) - розвиток гри в зв'язку з відгуками: Як можна вдосконалити гру з урахуванням фідбеку гравців?; поширення гри: Як залучити більше гравців та зробити гру більш популярною?

Її ідеї відповідають сучасним поглядам на методологію розробки програмних продуктів, доповнюючи її особливостями властивими для ігрових проєктів.

Геймдизайн, як будь-яка проєктна діяльність, має свої обмеження та виклики, які можуть впливати на процес створення ігрового проєкту. Визначемо найбільш вагомі з них. Так недостатність фінансів для розробки та випуску гри може призвести до обмеження функціональності, якості графіки та загального вибору можливостей. Часові рамки можуть бути обмеженими, особливо якщо необхідно випустити гру в певний строк, що може вплинути на якість розробки та тестування. Обмеження технічних можливостей платформи чи обладнання, на якому гра буде запускатися, може обмежити функціональність або графічне оформлення гри. Необхідність працювати з обмеженим числом членів команди, які можуть мати різні рівні експертизи, може вплинути на обсяг та якість роботи. Іноді в геймдизайні може бути обмежений креативний процес через певні вимоги від видавців чи ринку, що може обмежувати оригінальність та інновації. Забезпечення відповідності очікуванням та потребам цільової аудиторії також може обмежувати креативний процес та можливості геймдизайну.

Зазначені обмеження можуть бути суттєвими при створенні ігрових проєктів персоналом закладів освіти або на їх замовлення.

В розробці ігор в якості методологічної основи використовується модель MDA (Mechanics, Dynamics and Aesthetics) рис. 1.3 [13].

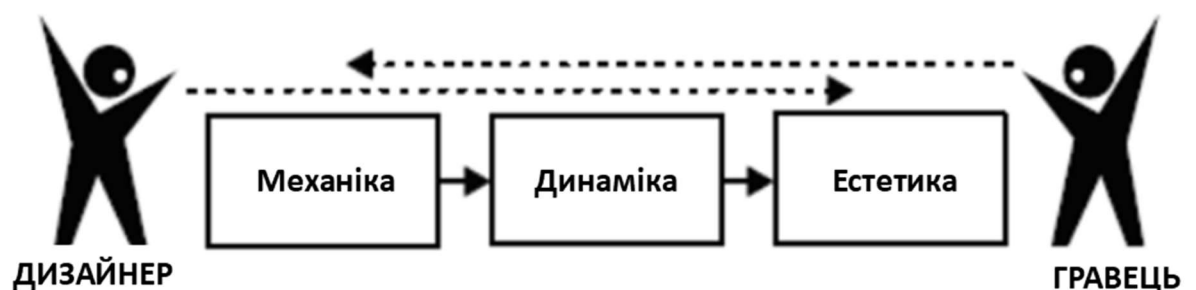


Рис.1.3. Модель MDA

Модель MDA (Mechanics, Dynamics, Aesthetics) - це підхід до аналізу та розуміння ігрового досвіду, який був розроблений Робертом Функе, Крістофером Шеллі та Джейсоном Шоком. Ця модель розглядає гру з трьох основних поглядів: механіка, динаміка та естетика.

Механіка (Mechanics) - це базові правила та системи, які визначають внутрішній функціонал ігрового середовища.

Механіка включає в себе всі конкретні правила, взаємодію гравця з грою, системи руху, спеціальні ефекти, системи балансування, умови перемоги чи поразки тощо.

Динаміка (Dynamics) - це взаємодія гравця з механікою гри, що включає в себе рух, зміну стану, емоційні реакції, стратегічні та тактичні вибори тощо.

Динаміка описує, які враження викликає гра у гравців, які відчуття вона пробуджує та як вона еволюціонує від початку до кінця.

Естетика (Aesthetics) - це емоційний стан, який викликається в гравцях під час взаємодії з механікою та динамікою гри.

Естетика визначає, чи відчуває гравець задоволення від гри, чи вона цікава, чи викликає вона почуття досягнення, влади, співчуття, експлорації тощо.

Модель MDA допомагає розробникам та дизайнерам гір краще розуміти, як механіка перетворюється на динаміку, а динаміка - на естетику, забезпечуючи гравцеві глибокий та задовольючий ігровий досвід. Ця модель допомагає краще аналізувати, проєктувати та оцінювати якість ігрового процесу та відчуттів, які він викликає.

Не менш важливою в контексті освітнього процесу є модель DPE (Design, Play and Experience) рис.1.4 [14].

Модель DPE (Design, Play and Experience) - це підхід до навчання, який акцентує на дизайні навчального процесу, активній участі здобувачів у грі або ігрових вправах та відчуттях, які вони отримують від навчання. Ця модель використовує елементи гейміфікації та геймдизайну для поліпшення процесу

навчання. Розглянемо основні складові цієї моделі: дизайн, гру та експеримент.



Рис. 1.4. Модель DPE

Дизайн у моделі DPE стосується розробки навчальних матеріалів та програми, спрямованих на досягнення певних навчальних цілей. Це включає в себе створення уроків, завдань, методів оцінювання, які відповідають певним вимогам навчання.

У моделі DPE акцент робиться на ігровому підході до навчання. Це включає в себе використання ігор, ігрових елементів або симуляцій для активного залучення учнів до навчального процесу. Гра використовується для стимулювання інтересу та активізації навчального досвіду.

Експеримент - це фаза, де учні отримують навчальний досвід через активну участь у грі або інших інтерактивних заняттях. Вони виконують завдання, вирішують проблеми, набувають знання та навички через практичний досвід.

Модель DPE намагається створити більш захоплююче та ефективне навчальне середовище, в якому здобувачі активно залучаються та здобувають знання через гру та експеримент. Цей підхід стимулює більшу увагу, зацікавленість та взаємодію в процесі навчання.

1.3. Програми- симулятори навчального призначення як різновид ігрового програмного забезпечення. Їх місце та роль в освіті України

Програма-симулятор навчального призначення - це вид ігрового програмного забезпечення, який спрямований на навчання та набуття навичок через імітацію реальних сценаріїв або ситуацій.

Програми-симулятори вважаються видом ігрового програмного забезпечення через їхні основні характеристики та способи використання. Вони моделюють реальні або імітують абстрактні сценарії та ситуації, дозволяючи користувачеві взаємодіяти з ними. Це подібно до основної функції багатьох ігор - створення віртуальних світів або ситуацій для гравців. Можуть включати в себе різноманітні інтерактивні елементи, завдання, вправи та ігрові механіки, які заохочують активну участь користувача. Це подібно до того, як у багатьох іграх гравець взаємодіє з ігровим світом. Програми-симулятори часто розробляються з певною ціллю, яка може бути пов'язана з навчанням, тренуванням або емуляцією певних аспектів реальності. Це подібно до завдань чи мети, які є у багатьох іграх.

Симулятори як правило реалізують принцип активного навчання або навчання заснованого на дії.

Програми-симулятори в освіті мають великий потенціал для формування професійних компетенцій здобувачів через відтворення реальних або симульованих сценаріїв та ситуацій. Симулятори дозволяють створювати реалістичні сценарії без реальних ризиків для здобувачів освіти. Наприклад, симулятори верстатів дозволяють майбутнім верстатникам широкого профілю або операторам верстатів з числовим програмним управлінням вправлятися

без реального ризику отримати травму на малознайомому верстаті або призвести до виникненні аварійної ситуації на вартісному устаткуванні. Вдалим прикладом такого симулятора є Swansoft CNC SIMULATOR рис. 1.5., який дозволяю відпрацьовувати навички програмування та механічної обробки деталей для усіх груп верстатів з найбільш популярними у світі системами програмного управління.



Рисунок 1.5. Приклад симулятора роботи на верстатах з ПУ

Завдяки симуляції реальних робочих ситуацій, здобувачі освіти можуть отримати практичний досвід, який відображає справжні виробничі умови, наприклад збирання електричного ланцюга системи освітлення, монтажу устаткування у шафу електромонтером з ремонту та обслуговування електроустаткування, встановлення у стійку на налагодження мережевого устаткування оператором з обробки інформації та програмного забезпечення тощо. Симулятори можуть створювати сценарії, де здобувачі освіти повинні приймати рішення в реальних або подібних реальним ситуаціях, що сприяє розвитку їхніх аналітичних та рішення-орієнтованих навичок[15].

Програми-симулятори можуть адаптуватися до рівня знань кожного здобувача освіти та надавати індивідуальний підхід. Крім того, їхній контент можна використовувати для багаторазового навчання. Завдяки можливості відстеження результатів та дії фідбеку, програми-симулятори можуть адаптуватися для поліпшення навчального процесу. Навчання за допомогою симуляторів дозволяє здобувачам освіти працювати зі складними сценаріями, які можуть бути важко відтворити у реальних умовах.

Програми-симулятори відіграють важливу роль у дистанційному навчанні. Для певних професій або навичок, які вимагають практичних навичок, програми-симулятори надають можливість здобувачам віртуальної практики та експериментування з безпекою та ефективністю.

Програми-симулятори можуть бути налаштовані для відповіді на конкретні навчальні потреби, рівень знань та індивідуальні особливості здобувачів. Вони дозволяють здобувачам навчатися у своєму власному темпі та в будь-якому місці, де є доступ до інтернету та програми. Симулятори можуть забезпечувати можливість більш ефективного використання часу та ресурсів, оскільки вони можуть надати доступ до практичного досвіду без необхідності реального обладнання або приміщень.

В Україні симуляційне навчання почало активно використовуватися при підготовці фахівців медичного профілю, пілотів для керування повітряними суднами, водіїв автомобілів.

На жаль, точної кількості програм-симуляторів для навчання кваліфікованих робітників, розроблених в Україні, встановити не можливо, оскільки не існує універсальної бази даних або зведеної статистики. Не має рекомендованих або схвалених МОНУ або хоча б повноцінно апробованих програм-симуляторів, що можуть бути використані для створення освітнього середовища закладу освіти.

Причини такого явища є декілька. Насамперед, брак доступу до сучасного обладнання та технологій ускладнює створення ефективних симуляційних програм. Матеріально-технічна база закладів професійної

(професійно-технічної) освіти та інфраструктура є застарілою [16]. Хоча в останні роки можна спостерігати збільшення інвестицій в заклади П(ПТ)О, створення навчально-практичних центрів, зокрема за професіями державного значення, ситуація із доступом до новітніх технологій, стажуванням викладачів та майстрів виробничого навчання в умовах реального виробництва та швидкої зміни технологій залишається складною.

Також майже відсутнє фінансування розвитку технологічних інфраструктур для симуляційного навчання. Через брак програм-симуляторів відсутній попит на підвищення кваліфікації педагогічних працівників щодо формування педагогічних компетентностей в галузі симуляційного навчання. Відсутність чітких методичних рекомендацій та стандартів для симуляційного навчання також гальмує процес розробки та впровадження програм-симуляторів. На додачу до вищезазначеного існує брак кваліфікованих фахівців з розробки симуляторів та спеціалістів із знанням освітніх потреб для ефективного впровадження цих програм у навчальний процес.

1.4 Висновки до розділу 1

В першому розділі описані сучасні парадигми навчання, що передбачають використання комп'ютерних ігрових проєктів навчального призначення. Проаналізовано роботи закордонних та вітчизняних науковців та практиків із зазначеної тематики. Висвітлено основні аспекти гейміфікації та навчання, заснованого на принципах комп'ютерних ігор.

Наведено загальну структуру геймдизайну та доведено її відповідність міжнародним стандартам із розробки програмного забезпечення. Описано концепцію створення комп'ютерної гри за Джейн Макгонікал. Розглянуто моделі геймдевелопмента MDA та DPE. Описані їх складові та наведено графічне представлення сутності моделей.

У розділі показано, що програми-симулятори навчального призначення є видом ігрового програмного забезпечення, описано їх потенціал та роль у вітчизняній освіті.

Зазначені проблеми та кризові явища, що пов'язані із розробкою вітчизняних ігрових проєктів – симуляторів для системи професійної (професійно-технічної) освіти, вказано на їх гострий брак та відсутність ефективних нормативно-методичних інструментів щодо їх розробки.

РОЗДІЛ 2 АНАЛІЗ МЕТОДОЛОГІЙ ТА ЗАСОБІВ РОЗРОБКИ

2.1 Огляд існуючих на ринку аналогічних програмних продуктів

Кількість розповсюджених симуляторів збирання комп'ютерів на ринку не є дуже великою. Можна віділити декілька найбільш популярних продуктів, зокрема PC Building Simulator, 3DMark, PC Architect, Build Your Own PC Simulator. Їх порівняння подано у вигляді табл. 2.1.

Таблиця 2.1

Порівняння програм-симуляторів збирання ПК

| Критерій | PC Building Simulator | 3DMark (окремий модуль) | PC Architect | Build Your Own PC Simulator |
|--------------------------------------|-----------------------|------------------------------------|--|-----------------------------|
| Вартість, долларів США | 10-30 | 30-50 | Умовно безкоштовний продукт (є система внутрішніх покупок) або покупка з певними функціями | 5-15 |
| Можливість використання для навчання | так | ні | умовно | умовно |
| Реалістичність | Відмінна | Фокус на тестуванні продуктивності | Добра | Середня |

| Критерій | PC Building Simulator | 3DMark (окремий модуль) | PC Architect | Build Your Own PC Simulator |
|---------------------------------|-----------------------|--|---------------|-------------------------------------|
| Різноманітність компонентів | Широкий вибір | Обмежений вибір, більше для тестування | Широкий вибір | Обмежений вибір, базовий функціонал |
| Складність роботи з інтерфейсом | Інтуїтивний інтерфейс | Середня | Низька | Низька |

Як можна побачити, найбільш придатним є симулятор PC Building Simulator. Він має належний функціонал, дає можливість не тільки збирати, але й тестувати зібраний комп'ютер. Однак його суттєвим обмеженням є його вартість. Для виконання однієї чи декількох робіт здобувачі освіти в переважній більшості відмовляться його придбати. На момент написання роботи вартість симулятора для індивідуального використання складала 415 грн. Заклад освіти також у свою чергу малоспроможний придбати ігровий симулятор для здобувачів.

Отже, не дивлячись на наявність на ринку схожих за своїм функціоналом програмних продуктів, необхідність розробки власного симулятора збирання системного блоку комп'ютера залишається актуальною. Тим не менше знайомство із принципом роботи існуючих продуктів та їх інтерфейсом дає змогу більш виважено підійти до розробки власного продукту.

2.2 Розробка функціональної моделі програми-симулятора навчального призначення

«Функціональна модель програми-симулятора навчального призначення - це опис того, як програма повинна працювати з точки зору

навчання, навчальних процесів та досягнення освітніх цілей. Ця модель визначає функції, функціональні можливості та способи взаємодії з користувачем з урахуванням освітніх потреб»[17].

Основні складові функціональної моделі програми-симулятора навчального призначення зображені на рис.2.1.

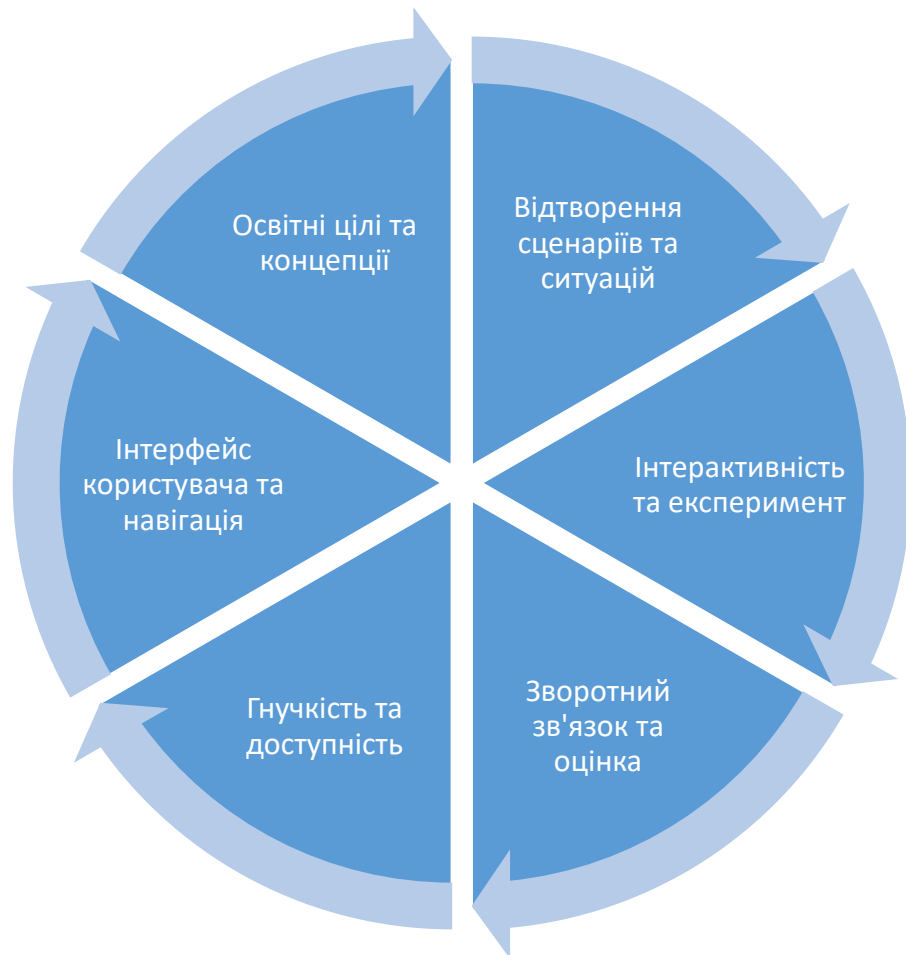


Рис.2.1. Складові функціональної моделі програми-симулятора навчального призначення

Під освітніми цілями та концепціями розуміється опис основних цілей навчання, на досягнення яких спрямована, а також ключові концепції, які повинні бути передані користувачам.

Відтворення сценаріїв та ситуацій – це здатність програми відтворювати реальні чи абстрактні сценарії, які допомагають учням краще зрозуміти матеріал.

Інтерактивність та експеримент передбачає, що програма повинна надавати можливості для взаємодії користувача з моделями та дозволяти їм проводити експерименти, щоб спостерігати результати в реальному часі.

Зворотний зв'язок та оцінка - це механізми для оцінки прогресу користувачів, отримання зворотного зв'язку та адаптації програми для кращого викладання.

Гнучкість та доступність передбачає, що програма повинна бути гнучкою, здатною адаптуватися до потреб різних користувачів та навчальних сценаріїв, а також ступінь її інклюзивності.

Інтерфейс користувача та навігація допомагають користувачам ефективно взаємодіяти з програмою та мати доступ до необхідної інформації.

Для розробки функціональної моделі програми-симулятора не першому етапі було проведено визначення вимог-очікувань від розробки серед її потенційних користувачів – здобувачів освіти та педагогічних працівників. Із здобувачами освіти робота була побудована у вигляді проведення фокус-груп (4 групи: 12, 30, 29, 30 здобувачів, загальна кількість - 101), а з викладачами був застосований метод інтерв'ю (3 викладача, 2 майстра виробничого навчання, всього 5 осіб). Результати проведеної роботи відображені у табл.2.2. Можна побачити, що вимоги потенційних замовників (здобувачів та педагогів) мали суттєві відмінності, тому у кінцевому варіанті були враховані побажання і тих і інших, але не задоволені повністю.

Враховуючи визначені початкові умови розробки стало можливим розробити структурно-функціональну схему симулятора.

Результати визначення вимог-очікувань до програми-симулятори

| Критерій | Здобувачі освіти | Педагоги |
|------------------------------------|--|---|
| Освітні цілі та концепції | Під час будь-яких занять | Для проведення практичних робіт, підбір компонентів та перевірка їх сумісності |
| Відтворення сценаріїв та ситуацій | Візуалізація лабораторного столу, корпусу ПК та комплектуючих, без додавання текстових документів - інструкцій, звітів | Візуалізація майстерні/лабораторії, наявність детальних інструкцій для практичних |
| Інтерактивність та експеримент | Можливість обирати деталі із переліку самостійно | Можливість визначати перелік деталей, який може використовувати здобувач освіти |
| Зворотний зв'язок та оцінка | Оцінка виставляється програмою | Оцінка виставляється викладачем |
| Гнучкість та доступність | Можна додавати нові комплектуючі, що з'являються на ринку | Достатньо обмеженого набору комплектуючих |
| Інтерфейс користувача та навігація | Ігровий, англomовний (за англomовну версію висловилося 87% здобувачів), навігація спрощена | Віконний, подібний до офісного пакету MS Office, україномовний |

Враховуючи наявні ресурси (людські, фінансові, часові) та кінцеву мету розробки – створення симулятора навчального призначення для проведення практичної роботи з предмету «Архітектура та обслуговування комп'ютерів» та демонстрацій на уроках з предметів «Інформатика» та «Інформаційні технології в галузях металообробки», було вирішено розробити продукт максимально простий у використанні, який можна легко розповсюдити, встановити та максимально просто отримати результат роботи.

Було вирішено відмовитися від ідеї створення програми-симулятора для виконання великої кількості практичних робіт на користь створення невеликих програм для кожної практичної роботи. На переконання автора це дасть змогу більш гнучко реагувати на зміни і навчальній програмі, за потреби змінювати окремі види робіт, а не переробляти великий проєкт за найменшої необхідності.

Просте пояснення роботи системи дає діаграма використання зображена на рис.2.2: учень підбирає компоненти, перевіряє сумісність, отримує результат та/або демонструє його викладачу.

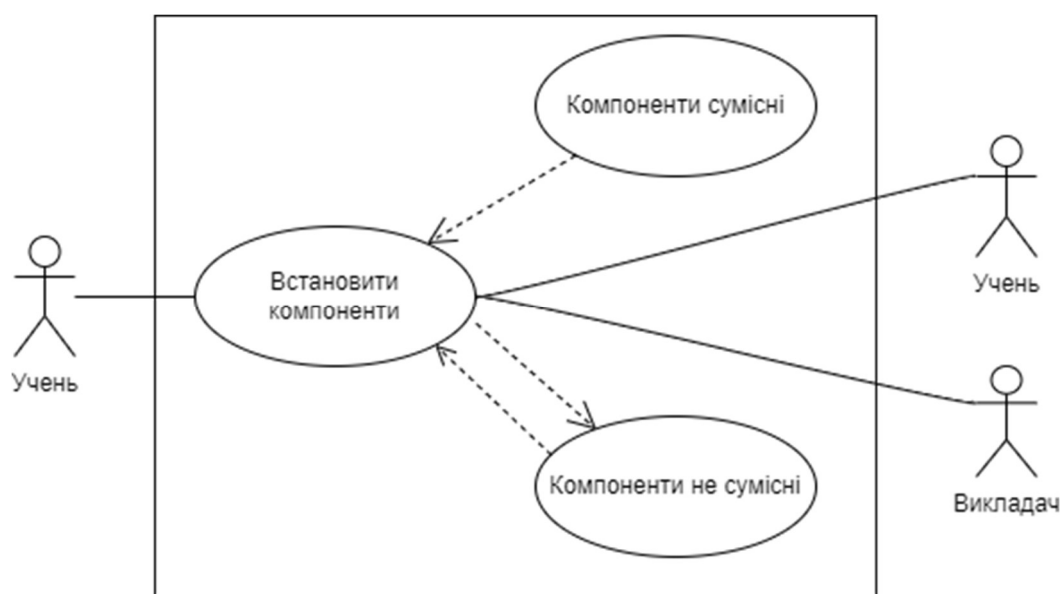


Рис 2.2. Діаграма використання програми-симулятора

Графічно функціональна схема розробки подана на рис. 2.3.

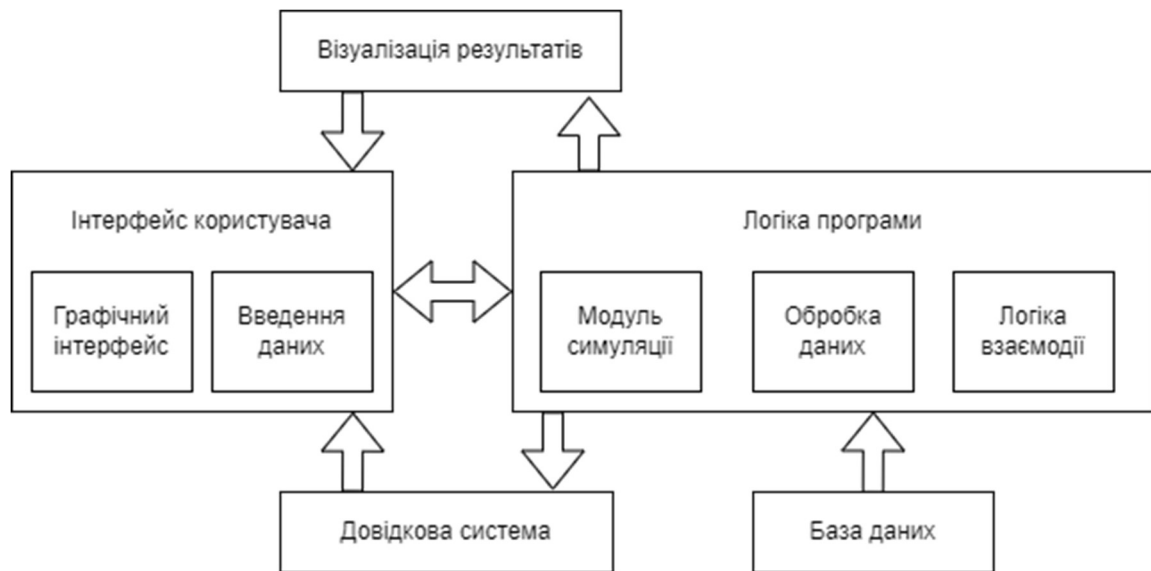


Рис 2.3. Функціональна схема розробки

Візуальна частина програми, що взаємодіє з користувачем буде містити елементи керування, вікна, кнопки, меню та інше. Механізм для введення вхідних даних передбачатиме вибір певних дій за допомогою елементів керування без ручного ведення додаткової інформації.

Модулі симуляції відповідатиме за імітацію процесів чи ситуацій, що вивчаються в навчальних цілях. Під обробкою даних розуміються алгоритми, що аналізують введені дані, виконують обчислення та обробку інформації. Логіка взаємодії у цьому випадку – це код, який визначає способи, який забезпечує взаємодію з користувачем.

Можливість відображення симульованих даних реалізується за допомогою блоку візуалізації результатів.

Під базою даних розуміється сховище для зберігання в цьому випадку моделей комплектуючих для використання їх у симуляції.

Довідкова система програми міститиме прості відомості про роботу із симулятором.

Отже, на первинному етапі розробки дано відповідь на питання «Що розробляти?».

2.3 Вибір методології розробки та моделі життєвого циклу

Методологія розробки програмного забезпечення – «це сукупність ідей, система принципів, характеристик, методів, засобів і понять які визначають стиль розробки» [18].

До основних методологій відносять:

- а) Waterfall (Каскадна) - лінійний підхід, де кожен етап розробки (аналіз, проектування, реалізація, тестування, впровадження) виконується послідовно;
- б) Agile (Гнучка) - ітеративний підхід, де розробка відбувається в короткі ітерації (спринти), що дозволяє гнучко адаптуватися до змін у вимогах та залучати замовника до процесу;
- в) Scrum - одна з методик гнучкої розробки, яка розділяє процес на короткі спринти, має чітко визначені ролі (Product Owner, Scrum Master, Development Team) та регулярні моменти огляду прогресу (Sprint Review, Sprint Retrospective);
- г) Kanban - система управління завданнями, яка базується на візуальному контролі потоку роботи через дошку з завданнями, їхніми етапами та статусами;
- д) DevOps - підхід, що поєднує розробку програмного забезпечення з інфраструктурними змінами та використанням автоматизації для швидкого впровадження змін.
- е) Lean Software Development - методологія, що використовує принципи "Lean manufacturing" для оптимізації процесу розробки ПЗ, мінімізації витрат та максимізації цінності для клієнта;
- ж) RAD (Rapid Application Development) - акуратний ітеративний підхід, спрямований на швидке створення прототипів та швидких ітераційних циклів.

- з) XP (Extreme Programming) - методологія, що акцентується на якості коду через певні практики (парне програмування, тестування, постійну змінність).

Вибір методології розробки програмного забезпечення ґрунтується на декількох ключових критеріях, що варіюються в залежності від конкретної ситуації проєкту та команди.

В першу чергу варто розглянути характер проєкту, що було зроблено у попередньому розділі, тобто врахувати розмір, складність, технічні вимоги і терміни роботи над проєктом. Наприклад, для малих, швидких проєктів більш підходить Agile, тоді як для більших і більш структурованих - Waterfall.

Якщо замовник хоче бачити проміжні результати та зміни на ранніх стадіях, гнучкі методології, такі як Agile або Scrum, є більш доречними. Якщо команда розробників є розподієною територіально, то кращим варіантом є Agile з чіткою комунікацією.

Нова команда або команда початківців може відчувати себе комфортніше з Waterfall, тоді як досвідчена може працювати з Agile або DevOps.

За частоті зміни вимог доцільною є методологія, яка дозволяє гнучкість та легко адаптується до змін, наприклад, Agile.

Деякі методології, наприклад, XP, акцентуються на якості коду та тестуванні. Якщо це є суттєвою вимогою, то можна розглядати самі ці методології.

Розглянемо переваги та недоліки кожної із зазначених методологій[19].

Waterfall передбачає, що кожен етап (вимоги, проектування, реалізація, тестування, впровадження) проводиться послідовно та структуровано. Великий акцент на документації кожного етапу дозволяє зберігати інформацію та зрозумілість у всьому процесі розробки. Через послідовний характер робіт, керування проєктом може бути досить простим.

Методологія слабо пристосовується до змін у вимогах або потребах користувачів після того, як проєкт розпочатий. Якщо на ранніх етапах були

допущені помилки або неправильно зрозумілі вимоги, виправлення може бути складним і призвести до затримок. Результатів роботи можна не побачити до завершення всього циклу розробки.

Agile - дозволяє швидко реагувати на зміни вимог і потреби клієнта під час розробки продукту. Процес розробки поділений на короткі ітерації (спринти), що дозволяє швидко створювати функціональні частини продукту для оцінки. За допомогою регулярних демонстрацій функціональності замовник може активно взаємодіяти та надавати фідбек, що допомагає уникнути непорозумінь. Спрямована на швидке виявлення та вирішення проблем, що дозволяє управляти ризиками раніше та ефективніше. Через постійний цикл оцінки та вдосконалення, сприяє покращенню якості продукту. Завдяки акценту на співпраці та комунікації, допомагає створити ефективні команди.

Методологія Agile може бути складною у впровадженні для деяких видів проектів, особливо якщо вимоги чітко не визначені або стабільність змінюється дуже часто. Не завжди можна гарантувати постійну доступність замовника для взаємодії з командою. Ітеративність може привести до перерозподілу уваги та відволікання від головних цілей проекту. Прогнозування часу на кожен спринт може бути складним через невизначеність у вимогах на початку проекту. Брак ефективної комунікації в команді може призвести до непорозумінь та затримок у розробці. Для успішної роботи за методологією Agile потрібні висока кваліфікація та досвід учасників команди.

Методологія Scrum - одна з найпопулярніших підтипів Agile, і вона має свої переваги та недоліки.

Scrum дозволяє швидко адаптуватися до змін вимог та ринкових умов завдяки коротким ітераціям (спринтам). Через регулярні спринти і планування, Scrum сприяє постійному прогресу у розробці.

Замовник активно взаємодіє з командою та регулярно отримує результати, що дозволяє виправляти та відповідати на зміни вимог. Кожен

член команди у Scrum має конкретні обов'язки та відповідальності, що сприяє ефективності роботи. Регулярні ітерації дозволяють перевіряти та оцінювати якість продукту на ранніх стадіях.

Scrum може бути неефективним у випадках, коли вимоги чітко не визначені або коли потрібна стабільність в плануванні. Успішність Scrum значно залежить від сплоченості та досвіду команди. Щоб процес розробки був ефективним, необхідно дотримуватися всіх його принципів, що може бути складно для деяких команд. Прогнозування часу та обсягів робіт може бути важким завданням через постійні зміни вимог та природу робіт у кожному спринті.

Kanban - дозволяє командам гнучко реагувати на зміни пріоритетів та вимоги, не встановлюючи жорстких часових рамок. Завдяки дошці Kanban, на якій відображені завдання та їхні стани, команда може краще контролювати процес та виявляти можливі блокування. Спрямована на постійне покращення процесів та виявлення можливостей для оптимізації шляхом усунення затримок та витрат. Методологія дозволяє команді вибирати темп роботи, розподіляти завдання та визначати власні пріоритети. Kanban надає можливість відстежувати прогрес кожного завдання та виявляти проблемні ситуації.

Однак ця методологія менш структурована порівняно з іншими методологіями. Вона не накладає жорстких правил і має менше формалізований процес порівняно з іншими методиками, що може створювати плутанину для менш досвідчених команд. Якщо не встановлені чіткі ліміти на кількість завдань, що можна розмістити на дошці, це може призвести до перевантаження команди. Команді потрібно дотримуватися правил Kanban та виявляти високу самодисципліну для успішності методології. Kanban не надає чітких механізмів для прогнозування термінів виконання завдань, що може бути проблемою у деяких сценаріях.

DevOps - дозволяє прискорити випуск програмного забезпечення через автоматизацію процесів розробки, тестування та впровадження. Завдяки

автоматизованому тестуванню та моніторингу, можна швидко виявляти та усувати помилки, що покращує якість програмного забезпечення. DevOps сприяє злиттю розробників та операторів, сприяючи зміцненню співпраці та зменшенню пробілів у комунікації. Автоматизація та оптимізація процесів дозволяє ефективніше використовувати інфраструктуру та ресурси. Завдяки постійному моніторингу, DevOps дозволяє швидко виявляти та реагувати на потенційні проблеми.

Для успішного впровадження DevOps необхідно змінити культуру, процеси та інструменти, що може бути складним у великих організаціях. Впровадження передбачає обов'язкову наявність навичок автоматизації, контролю версій, безпеки та моніторингу. Перехід до цієї методології може призвести до збільшення кількості інструментів та процесів, що потрібно керувати, що може бути складним. Завдяки автоматизації та швидкому випуску, можуть виникнути виклики щодо безпеки та контролю доступу до інформації.

Lean Software Development (LSD)- орієнтована на потреби клієнта та дозволяє створювати програмне забезпечення, яке дійсно вирішує їхні проблеми та задовольняє вимоги. Впровадження принципу «Just-in-Time» забезпечує виробництво лише тих функцій, які необхідні в даний момент, що дозволяє уникнути витрат на непотрібні функції. Lean спрямований на виявлення та усунення будь-яких зайвих витрат часу, ресурсів та енергії. Зосередження на уникненні дефектів на ранніх етапах та постійна оптимізація процесу дозволяє підтримувати високу якість продукту та дозволяє швидко реагувати на зміни та ефективно керувати проектом, підтримуючи стале покращення.

Перехід до Lean може вимагати значних змін у культурі організації та управлінських процесах, що може бути складним. Успішна реалізація передбачає активну підтримку та залучення всіх рівнів персоналу до культурних змін та покращень. Оцінка та визначення, що є цінним для клієнта, може бути складною завданням. Lean передбачає активну взаємодію та

комунікацію всіх учасників процесу, що може бути важким у великих організаціях.

RAD - дозволяє створювати прототипи та виробляти програмне забезпечення швидше, що може бути корисним для проєктів з короткими термінами. Ітеративний підхід дає змогу залучати користувачів на ранніх стадіях, що допомагає уникнути непорозумінь та відповідати їхнім потребам. Методологія спрямована на зміни вимог під час розробки, що дозволяє легше адаптуватися до змін. Ітеративний процес дозволяє оперативно вносити зміни та виправляти помилки.

Через високу швидкість розробки, документація може бути недостатньою, що може ускладнити подальшу підтримку та розвиток програми. Висока швидкість розробки може призвести до недоліків у якості, якщо процес не контролюється належним чином. Швидкість розробки та постійна адаптація вимагають досвідчених фахівців, які можуть швидко реагувати на зміни. Може бути складно передбачити всі ресурси, необхідні для успішного завершення проєкту.

XP - дозволяє швидко адаптуватися до змін вимог та ринкових умов завдяки ітеративному підходу. Великий фокус на тестуванні та рефакторингу допомагає забезпечити високу якість програмного забезпечення. Залучення замовника на ранніх стадіях дозволяє забезпечити відповідність продукту його очікуванням. Чітко визначені правила роботи, що сприяють взаєморозумінню та прозорості в команді. Постійна перевірка, тестування та швидкі виправлення помилок допомагають уникнути серйозних ризиків.

Успіх XP відчутно залежить від досвіду та сплоченості команди. Може бути складним впровадження XP в організаціях із складною ієрархією та великим обсягом проєктів. Продуктивність у XP вимагає постійної уваги до процесів розробки, що може вимагати великої віддачі з боку команди. Інколи вимоги можуть змінюватися швидко, що може ускладнити їхню чітку фіксацію.

Для вибору оптимальної методології розробки було вирішено врахувати чотири принципи описані у роботі Алієстра Коуберна [20]. Та використати метод складання оціночних матриць[21].

При використанні матриць для вибору методології розробки був обраний наступний алгоритм, зображений на рис.2.4.



Рис.2.4. Алгоритм використання матриць для вибору методології

Було вирішено поєднати матрицю Річарда Уотера з матрицею Кокса та Фіндика та ввести бальне оцінювання критеріїв, а саме: 0 балів – не відповідає, 1 – відповідає частково, 2 – відповідає. У зв'язку з тим, що у автора робота є об'єктивний брак практичного досвіду управління проєктами з розробки програмного забезпечення, з метою отримання максимально об'єктивних результатів, до оцінювання проєкту за розробленою матрицею були залучені сторонні особи, а саме: 3 фахівця в галузі розробки програмного забезпечення, 2 викладача, 8 здобувачів освіти ОПР «фаховий молодший бакалавр» за спеціальністю «Комп'ютерні науки». Таким чином загальна кількість осіб залучених до оцінювання, включно із автором роботи, становила 14 осіб. Кожний учасник оцінювання отримав бланк таблиці-матриці для заповнення, опис методологій у тому вині у якому він був поданий у роботі та джерелі [21]. Отримані оцінки були усередненні та зведені в результуючі матрицю подану у вигляді табл. 2.3.

Матриця Уотера-Кокса-Фіндика (модифікована)

| Критерій | Методології розробки | | | | | | | |
|--|----------------------|-------|-------|--------|--------|-----|-----|-----|
| | Waterfall | Agile | Scrum | Kanban | DevOop | LSD | RAD | XP |
| Розмір проєкту | 1,7 | 1,1 | 1,1 | 1,0 | 0,8 | 1,5 | 0,7 | 1,0 |
| Критичність проєкту | 1,5 | 1,4 | 1,4 | 1,4 | 0,9 | 1,5 | 0,8 | 1,1 |
| Динамічність вимог | 1,8 | 1,2 | 1,1 | 1,1 | 0,8 | 1,2 | 0,7 | 0,6 |
| Розмір команди | 1,4 | 1,1 | 1,1 | 1,2 | 0,8 | 1,8 | 0,7 | 0,8 |
| Рівень досвіду розробника (-ів) | 1,2 | 0,9 | 0,9 | 1,0 | 0,5 | 1,1 | 0,5 | 0,6 |
| Підсумкова оцінка | 7,6 | 5,7 | 5,6 | 5,7 | 3,8 | 7,1 | 3,4 | 4,1 |
| Відсоток від максимально можливої оцінки | 76 | 57 | 56 | 57 | 38 | 71 | 34 | 41 |

Як можна побачити, жодна з традиційних методологій не є такою, щоб її можна назвати найкращою для проєкту, що описується. Найбільш оптимальними можна визначити дві методології Waterfall та Lean Software Development.

Поєднанням методологій Waterfall та Lean Software Development було вирішено створити гібридну модель, яка поєднує в собі елементи структурованої послідовності Waterfall та фокус Lean на оптимізації процесів, мінімізації витрат.

Основні етапи Waterfall (аналіз, проектування, реалізація, тестування, впровадження) для створення структурованого плану розробки поєднано з

принципом Lean для зменшення зайвих операцій, максимізації ефективності та врахуванні потреб для яких створюється симулятор.

Було додані елементи контрольних точок у процесі Waterfall, що дозволило швидше переглядати та вдосконалювати процес.

Були знижені витрати (в умовах фінансування розробки тільки за рахунок власних ресурсів) на створення докладної документації.

Життєвий цикл програмного забезпечення - це процес, який включає в себе всі етапи від початку створення програмного продукту до його припинення використання.

Класична каскадна модель життєвого циклу програмного забезпечення [22] також була частково змінена з урахуванням гібридизації методології та представлена на рис. 2.5.

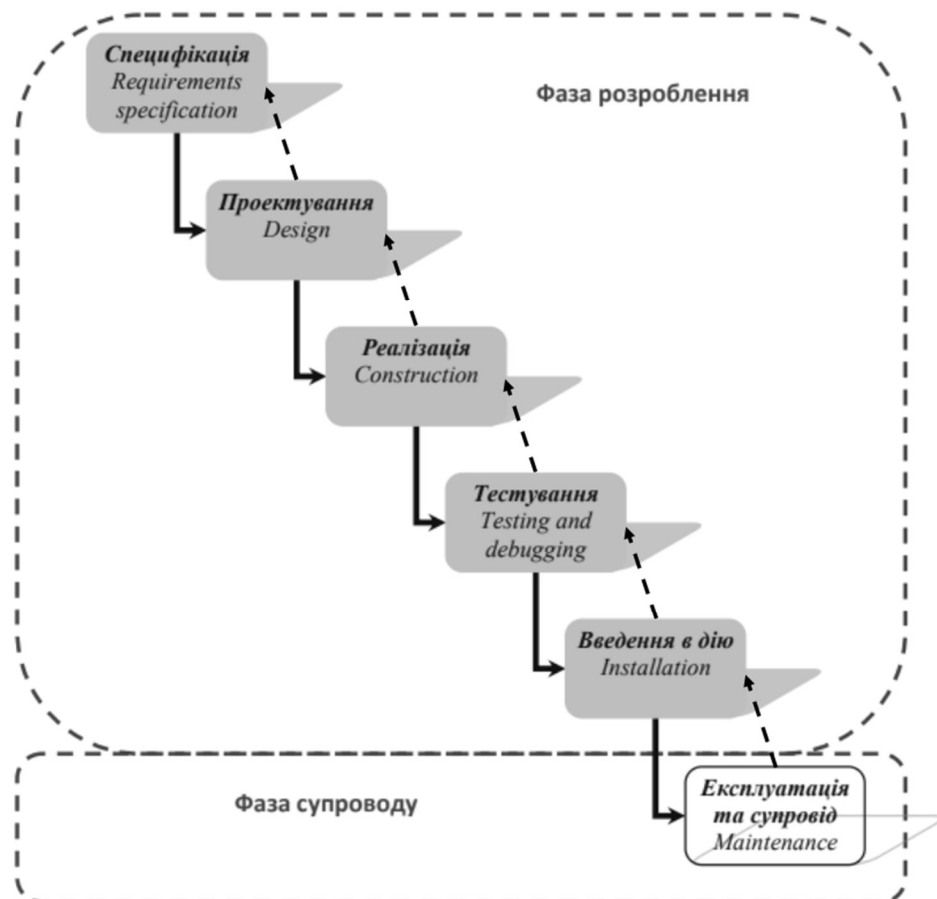


Рисунок 2.5. Модель життєвого циклу розробки програми-симулятора

2.4 Аналіз та вибір оптимальних інструментальних засобів планування та розробки

Темою роботи визначено рушій (платформу) на якому виконано розробку симулятора, а саме Unity 3D. В наявності автора перебуває Unity Pro 6.x (Educational Subscription Multi-install license) - це версія Unity Pro 6.x, яка доступна для освітніх цілей та має можливість мультиплікації на кілька пристроїв (Multi-install license). Ця версія Unity Pro має деякі обмеження, які пов'язані з освітнім статусом ліцензії, зокрема є обмеження використання лише для освітніх цілей та заборона комерційного використання розроблених продуктів. Враховуючи, що програма-симулятор розробляється для використання в освітньому процесі закладу освіти, який здійснює підготовку учнів за регіональним та державним замовленням (не за кошти фізичних та юридичних осіб), продаж розробленого продукту не передбачений, використання для розробки цієї версії Unity задовольняє ліцензійним вимогам.

Порівнюючи Unity з іншими рушіями (платформами) для створення ігрових проєктів можна скласти наступну таблицю (табл. 2.4).

Таблиця 2.4

Порівняння Unity з популярними рушіями для розробки ігор

| Критерій | Unity | Unreal Engine | CryEngine | Godot |
|-------------------------|--------------------|--------------------|-------------------|----------------|
| Кросплатформеність | Так | Так | Ні | Так |
| Мови програмування | C#, UnityScript | C++, Blueprint | C++, Lua | GScript, C# |
| Графіка | Добре | Висока | Висока | Середня |
| Ресурси | Широкий спектр | Широкий спектр | Широкий спектр | Обмежений |
| Візуальне програмування | Є (Bolt) | Так (Blueprint) | Ні | Ні |

| Критерій | Unity | Unreal Engine | CryEngine | Godot |
|-----------|------------------------|------------------------|-----------|--------------|
| Спільнота | Велика | Велика | Середня | Середня |
| Вартість | Безкоштовний - платний | Безкоштовний - платний | Ліцензія | Безкоштовний |

В цілому Unity та Unreal Engine мають схожі характеристики, однак саме Unity вважається більш простим для розробників початківців, має більшу спільноту розробників та кращу інтеграцію із зовнішніми ресурсами, підходить для малих та бюджетних проєктів [24].

Створення ігрового проєкту майже не можливе без використання графіки, яка відіграє величезну роль. Вона визначає візуальний стиль гри, створює атмосферу та перші враження від гри.

Перед створенням активів гри, художники створюють концепт-арт, який визначає вигляд персонажів, об'єктів, локацій тощо. Графіка це також і моделювання 2D або 3D моделей персонажів, об'єктів, архітектурних елементів тощо. Додавання текстур та деталей до моделей здійснюється для створення реалістичності та естетики. Рухи персонажів, об'єктів чи ефекти, які надають життя світу гри. Графіка використовується при створенні кнопок, меню, інформаційних панелей та іншої графіки для спілкування гравця з грою[25].

Напрямки використання графіки при створенні ігрових проєктів можна подати у вигляді схеми, зображеної на рис. 2.6.

Графічний редактор для розробки ігор - це програмне забезпечення, яке дозволяє створювати та редагувати графіку, відображення та візуальні ефекти для використання у ігрових проєктах.

Доцільно, щоб він мав наступні можливості:

- а) Редактори спрайтів для створення анімацій, персонажів, об'єктів та фонів гри;

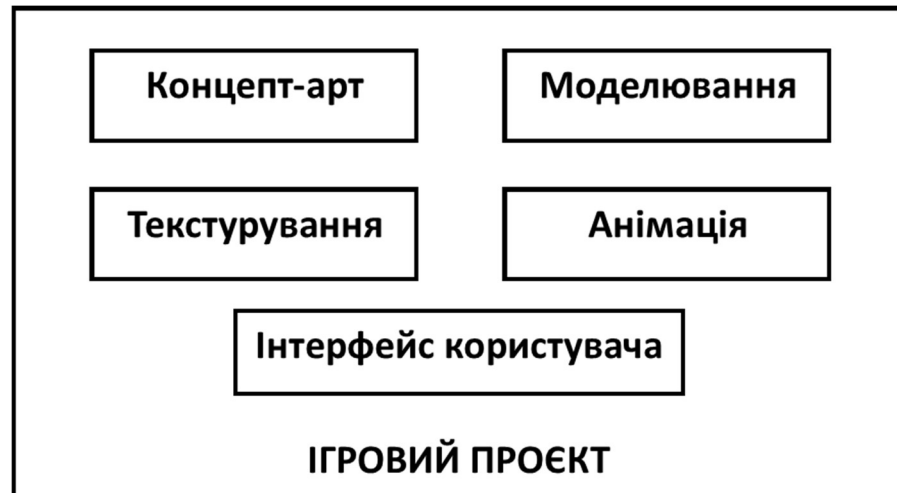


Рисунок 2.6. Застосування графіки в ігрових проєктах

- б) Інструменти 3D моделювання для створення тривимірних об'єктів, тварин, архітектурних елементів та інших деталей гри;
- в) Текстурні редактори для створення та редагування текстур, які використовуються для накладання на 3D-моделі;
- г) Редактори анімацій для створення та управління рухами та анімаціями об'єктів чи персонажів у грі;
- д) Інструменти для рендерингу для створення візуально привабливих сцен та ефектів;
- е) Редактори рівнів для створення та організації рівнів, світів та оточення у грі.

Було здійснено порівняння популярних графічних редакторів, яке подано у таблиці 2.5.

Для подальшої роботи було обрано Blender. Основним і переважаючим критерієм, що визначив його вибір – безкоштовність, оскільки це відповідає розглянутому вище принципу мінімізації видатків. Також важливим є доступність довідкових матеріалів по цьому редактору, зокрема українською мовою.

**Порівняння графічних редакторів, що можуть використовуватися
при створенні ігрових проєктів**

| Критерій | Blender 3D | Autodesk Maya | 3ds Max | Cinema 4D |
|-----------------------|---|--|--|---|
| Вартість | Безкоштов- ний | Платний | Платний | Платний |
| Інтерфейс | Різноманіт- ний, складний для початківців | Професій- ний, складний для початківців | Професійний, складний для початківців | Інтуїтивний, добрий для початківців |
| Функціональ- ність | Широкий спектр функцій для моделюван- ня, текстуруван- ня, анімації, рендерингу та багато іншого | Розширені функції для моделюван- ня, анімації, рендерингу та багато іншого | Розширені функції для моделюван- ня, анімації, рендерингу та багато іншого | Різноманітні можливості для моделюван- ня, анімації та рендерингу |
| Технічна підтримка | Велика спільнота розробників, офіційна документація та онлайн- ресурси | Офіційна та спільнотна підтримка, документація | Офіційна та спільнотна підтримка, документація | Велика спільнота, документа- ція, технічна підтримка |

| Критерій | Blender 3D | Autodesk Maya | 3ds Max | Cinema 4D |
|----------------------|---|--|---|--|
| Підтримка платформ | Windows, macOS, Linux | Windows | Windows | Windows, macOS |
| Програмування | Python, Node-based шейдери, скриптинг | Python, MEL, C++ | MaxScript, Python | Python, C++ |
| Ринкова популярність | Популярний серед користувачів та використовується у різних індустріях | Широко використовується в індустрії та серед професіоналів | Використовується в індустрії та серед професіоналів | Використовується в різних сферах, але менш поширений у порівнянні з іншими |

Як інструмент управління проєктами, який візуально відображає часові рамки та послідовність завдань у проєкті було вирішено використати діаграму Ганта[26]. Враховуючи простоту проєкта та кількість фактичних виконавців, що дорівнює одному, застосування більш складних засобів планування вбачається недоцільним.

Програмних засобів та онлайн-платформ для створення діаграм Ганта сьогодні існує багато, зокрема популярними є: Microsoft Project, Asana, Trello, Smartsheet, Google Sheets, Microsoft Excel, Canva.

У ВПУ № 7 м.Кременчука є навчальна ліцензія Canva та розгорнутий сервіс Google Workspace for Education Fundamentals. Враховуючи зручність користування сервісами та наявність готових шаблонів [27], вибір був зроблений на користь Google Sheets, які є частиною Google Workspace.

2.5 Висновки до розділу 2

В другому розділі роботи було проаналізовано наявність на ринку програм-симуляторів збирання комп'ютера та виявлено відсутність безкоштовних придатних для використання в освітньому процесі розробок, що підтверджує доцільність створення власного симулятора.

Були визначені складові функціональної моделі програми-симулятори, вимоги до програми, розроблена функціональна схема та UML-діаграма використання.

Значну увагу приділено вибору та обгартуванню методології розробки продукту. Були проаналізовані традиційні методології, запропоновано алгоритм використання матриць для вибору методології. Модифіковано матриці Річарда Уотера та Кокса і Фіндика для реалізації вибору. Показано перспективність використання гібридних методологій, як таких, що в більшій мірі в порівнянні із класичними здатні задовольнити умовам проєкту та забезпечити отримання позитивного результату. В роботі запропоновано поєднання методології Waterfall та Lean Software Development. За основу моделі життєвого циклу програмного засобу була обрана каскадна модель, яку вдосконалено додаванням точок контролю на кожному етапі розробки.

Під час аналізу та вибору засобів планування та розробки була показана доцільність вибору Unity, в якості ігрового рушія, Blender в якості графічного редактора, а Google Sheets, як засобу для створення діаграми Ганта для планування роботи над проєктом. Зразок діграми див.додаток А.

РОЗДІЛ 3 ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМИ-СИМУЛЯТОРА НА РУШІЇ UNITY 3D

3.1 Розробка ескізів вікон програми

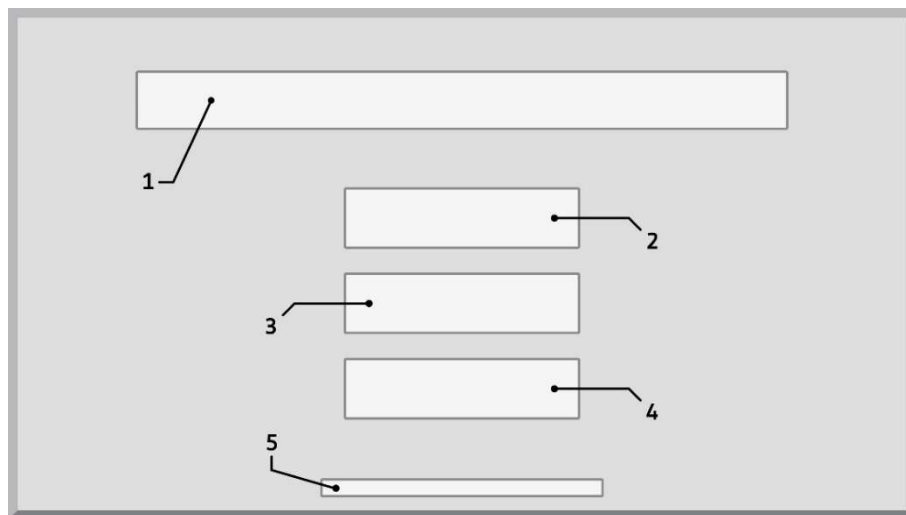
Насамперед були розроблені ескізи вікон програми. Ескіз вікна програми - це малюнок або макет, який відображає організацію та вигляд графічного інтерфейсу користувача програми [28]. Ескізи (або скетчі) графічних інтерфейсів використовуються при розробці програм для того, щоб швидко візуалізувати концепції та ідеї перед тим, як розпочати фактичну розробку.

На ескізах представлені позиціювання кнопок та інших елементів у вікні програми.

Головним призначенням ескізів була швидка перевірка ідей та створення основи для подальшої розробки.

В процесі роботи над ескізами частина з них була спрощена та перероблена. Отримані ескізи подані нижче.

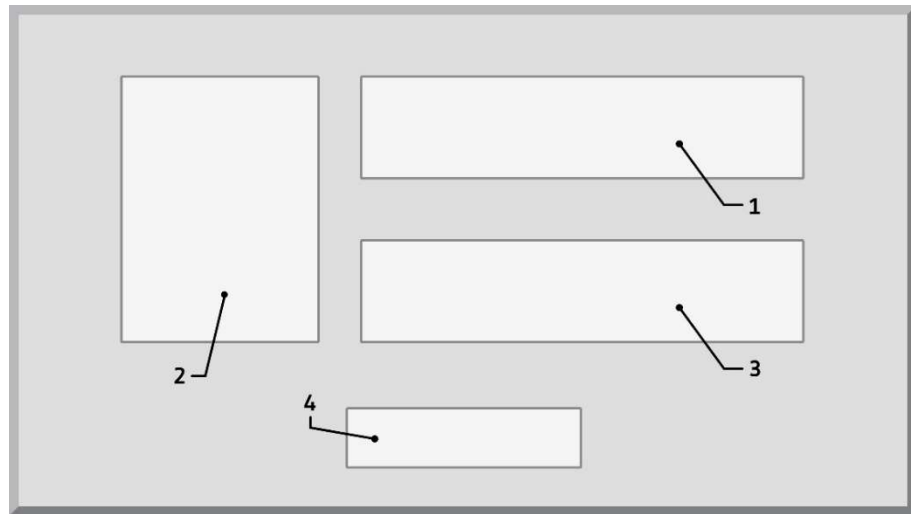
Ескіз головного вікна програми з приблизним розташуванням графічних елементів інтерфейсу представлено на рисунку 2.1.



1 – заголовок вікна; 2 – кнопка для початку симуляції; 3 – кнопка для відображення інструкції користувача; 4 – кнопка для виходу з програми; 5 – область для розміщення інформації про некомерційне використання

Рис.3.1 Ескіз головного вікна програми

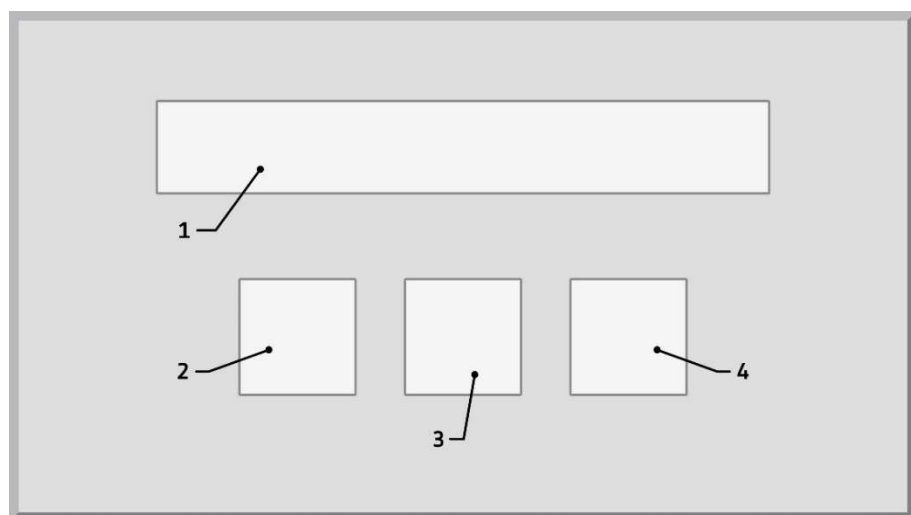
Ескіз вікна програми з інструкцією користувача з приблизним розташуванням графічних елементів інтерфейсу представлено на рис. 3.2.



1 – заголовок вікна; 2 – інструкції щодо використання клавіш клавіатури; 3 – - інструкції щодо використання кнопок інтерфейсу; 4 – кнопка для повернення до головного меню.

Рис. 3.2 Ескіз вікна програми з інструкцією користувача

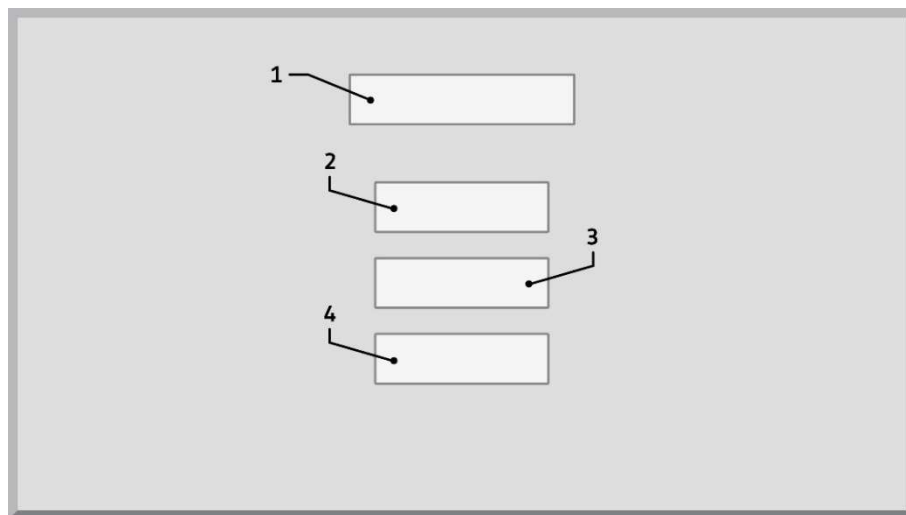
Ескіз вікна програми для вибору корпусу системного блоку персонального комп'ютера з приблизним розташуванням графічних елементів інтерфейсу представлено на рис.3.3.



1 – заголовок вікна; 2 – кнопка для вибору першого корпусу; 3 – кнопка для вибору другого корпусу; 4 – кнопка для вибору третього корпусу

Рис. 3.3 Ескіз вікна програми для вибору корпусу

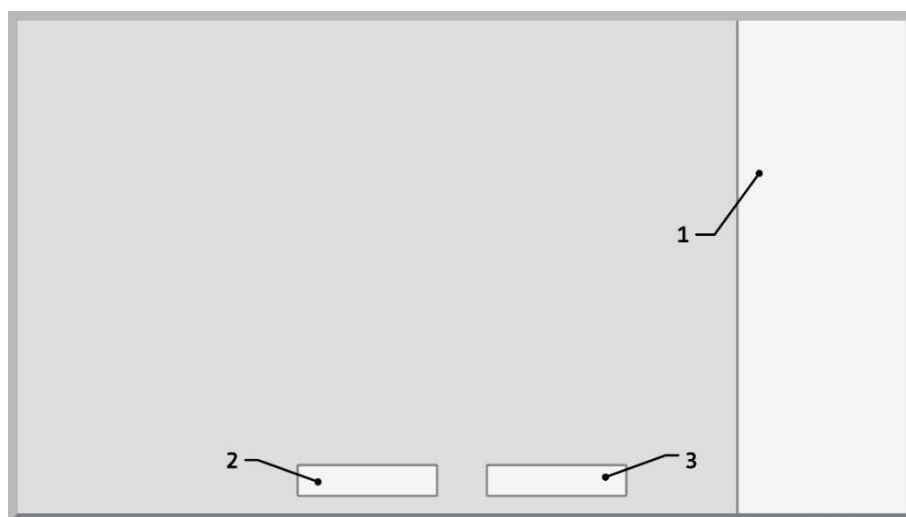
Ескіз вікна паузи програми з приблизним розташуванням графічних елементів інтерфейсу представлено на рисунку 3.4.



1 – заголовок вікна; 2 – кнопка для відновлення процесу симуляції; 3 – кнопка для повернення до головного меню; 4 – кнопка для виходу з програми

Рис.2.4 Ескіз вікна паузи програми

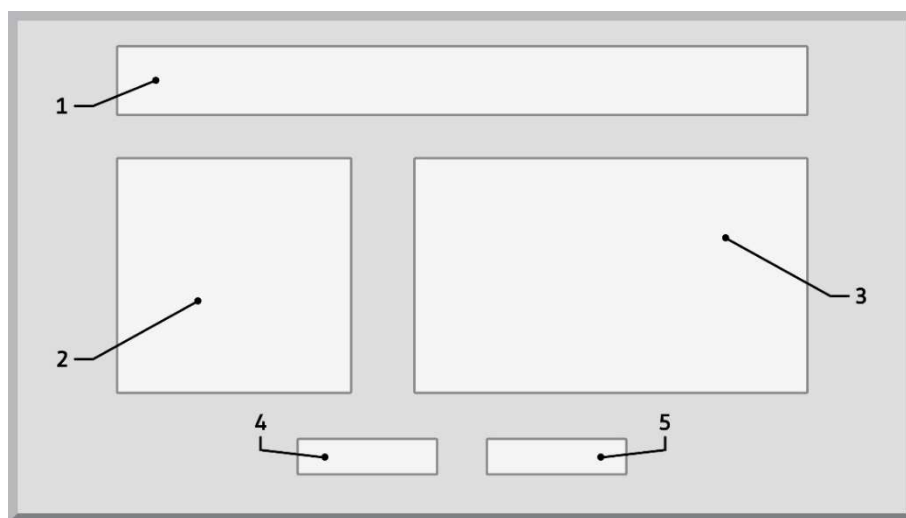
Ескіз вікна програми ігрової сцени з приблизним розташуванням графічних елементів інтерфейсу представлено на рис. 3.5.



1 – інвентар з компонентами ПК; 2 – кнопка для перевірки зібраної конфігурації;
3 – кнопка для видалення останнього влаштованого елементу в корпус

Рис.3.5 Ескіз вікна програми ігрової сцени

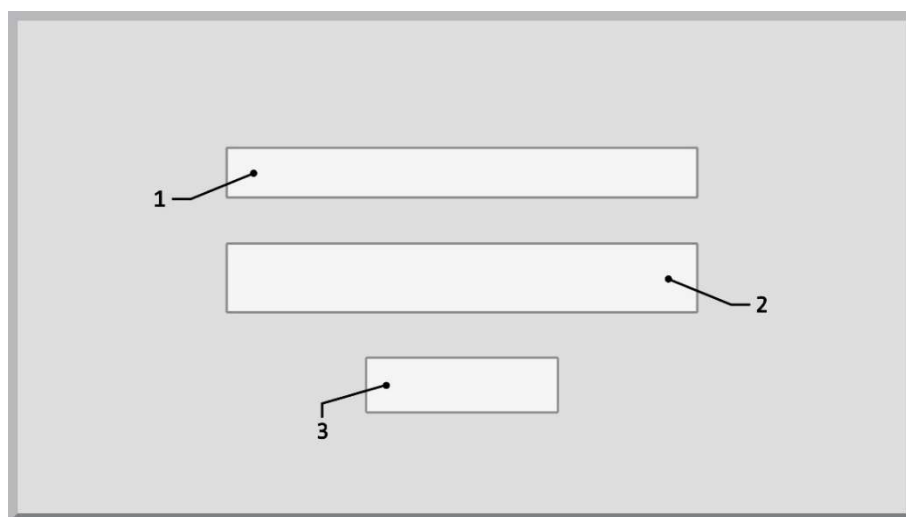
Ескіз вікна програми з описом характеристик компонента з приблизним розташуванням графічних елементів інтерфейсу представлено на рис. 3.6.



1 – назва компонента; 2 – зображення компонента; 3 – область з детальним описом характеристик; 4 – кнопка для генерації компонента; 5 – кнопка для закриття вікна

Рис. 3.6 Ескіз вікна програми з інформацією про компонент

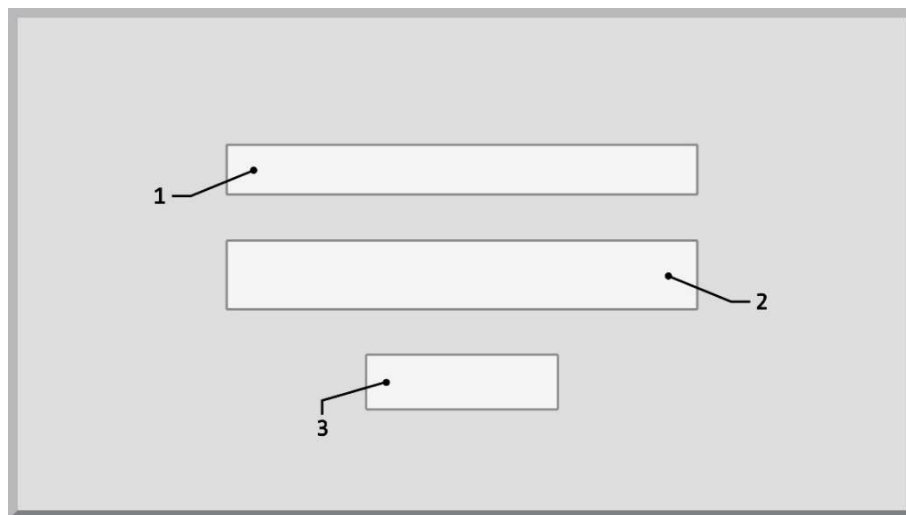
Ескіз вікна програми з повідомленням про помилку збирання системного блоку ПК з приблизним розташуванням графічних елементів інтерфейсу представлено на рисунку 3.7.



1 – заголовок вікна; 2 – повідомлення про помилку збирання системного блоку ПК; 3 – кнопка для повернення до етапу збирання системного блоку ПК

Рис. 3.7 Ескіз вікна програми з повідомленням про помилку збирання системного блоку ПК

Ескіз вікна програми з повідомленням про успішне збирання системного блоку ПК з приблизним розташуванням графічних елементів інтерфейсу представлено на рисунку 3.8.



1 – заголовок вікна; 2 – повідомлення про успішне збирання системного блоку ПК;
3 – кнопка для повернення до головного меню

Рис. 3.8 Ескіз вікна програми з повідомленням про успішне збирання системного блоку ПК

3.2 Створення моделей елементів корпусу та комплектуючих

Алгоритм створення моделей в Blender 3D є майже однаковим для усіх моделей, полягає у використанні примітивів із застосуванням до них модифікаторів та редакторів з подальшим текстурингом, освітленням та рендерингом. Варіант створеної 3D-моделі материнської плати представлено на рис. 3.9.



Рис. 3.9 Варіант 3D-моделі материнської плати

Остаточний варіант 3D-моделі ЦП представлено на рис. 3.10.

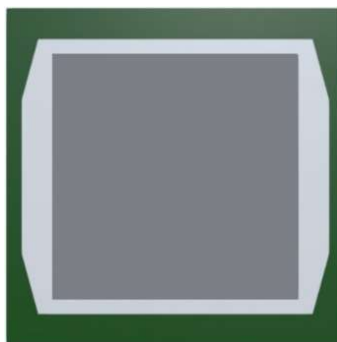


Рис. 3.10 Варіант 3D-моделі ЦП (вигляд зверху)

Зразок 3D-моделі графічного адаптера представлено на рис. 3.11.

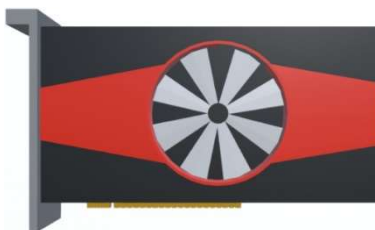


Рис. 3.11 Варіант 3D-моделі графічного адаптера

3D-модель модуля оперативної пам'яті представлена на рис 3.12.



Рис. 3.12 Варіант 3D-моделі оперативної пам'яті

Варіант 3D-моделі системи охолодження для ЦП представлено на рис. 3.13.

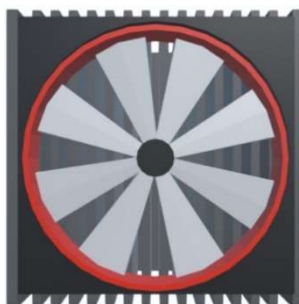


Рис. 3.13 Варіант 3D-моделі системи охолодження для ЦП (вигляд зверху)

Варіант 3D-моделі жорсткого диска представлено на рис. 3.14.



Рис. 3.14 Варіант 3D-моделі жорсткого диска (вигляд зверху)

Варіант 3D-моделі блоку живлення представлено на рис. 3.15.

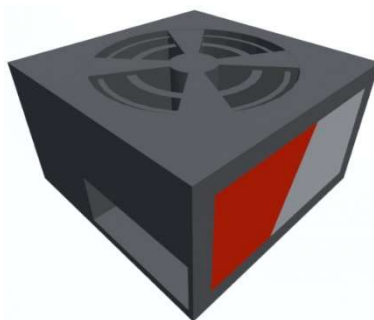


Рис. 3.15 Варіант 3D-моделі блоку живлення (вигляд збоку)

Остаточний варіант 3D-моделі комп'ютерного корпусу представлено на рис. 3.16.



Рис. 3.16 Варіант 3D-моделі комп'ютерного корпусу (вигляд збоку)

Не дивлячись на порівняно незначну кількість створених моделей, цей етап роботи виявився доволі тривалим у часі. Моделі було додано до проєкту у форматі fbx.

3.3 Механізм реалізації логіки симулятора

Описаний у другому розділі принцип організації логіки симулятора забезпечений шляхом реалізації поданих нижче алгоритмів, представлених у вигляді блок-схем. Так алгоритм призначений для відображення інвентарю з компонентами ПК, представлено на рис.3.17.

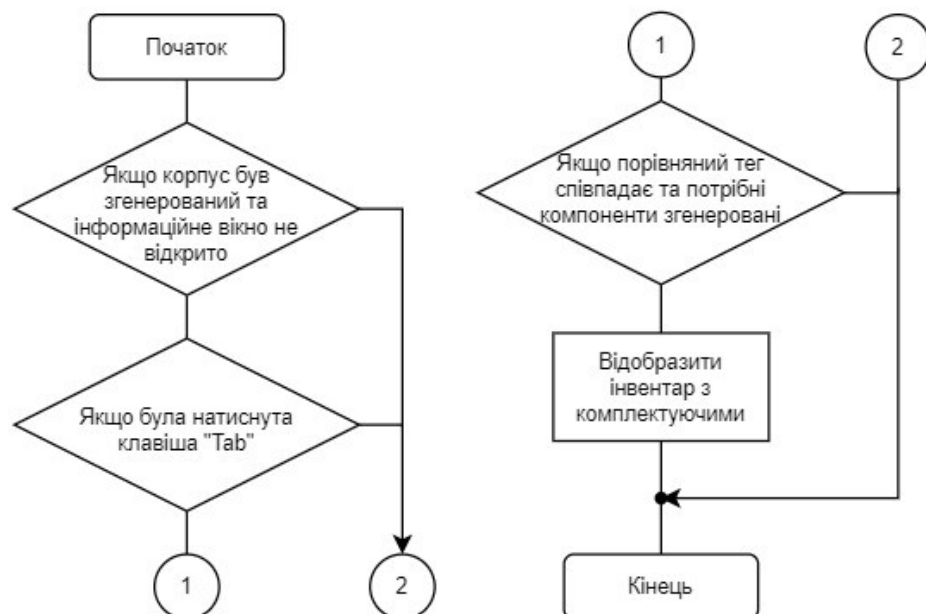


Рис. 3.17 Блок-схема алгоритму для відображення інвентарю

Блок-схему алгоритму, призначеного для вставлення компонентів в корпус системного блока ПК можна представити за допомогою блок-схеми на рис. 3.18.

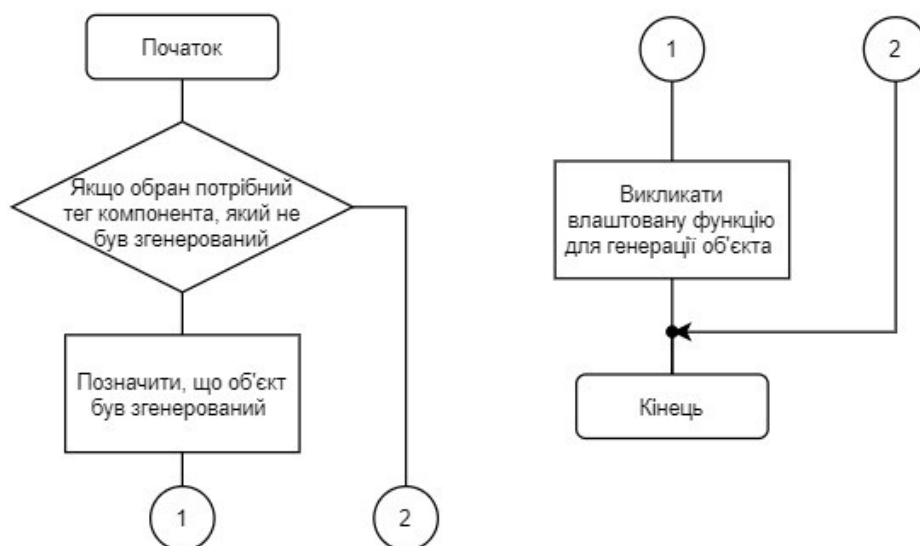


Рис. 3.18 Блок-схема алгоритму для генерації об'єктів

Блок-схему алгоритму, призначеного для видалення компонентів з корпусу системного блока ПК, представлено на рис.3.19.

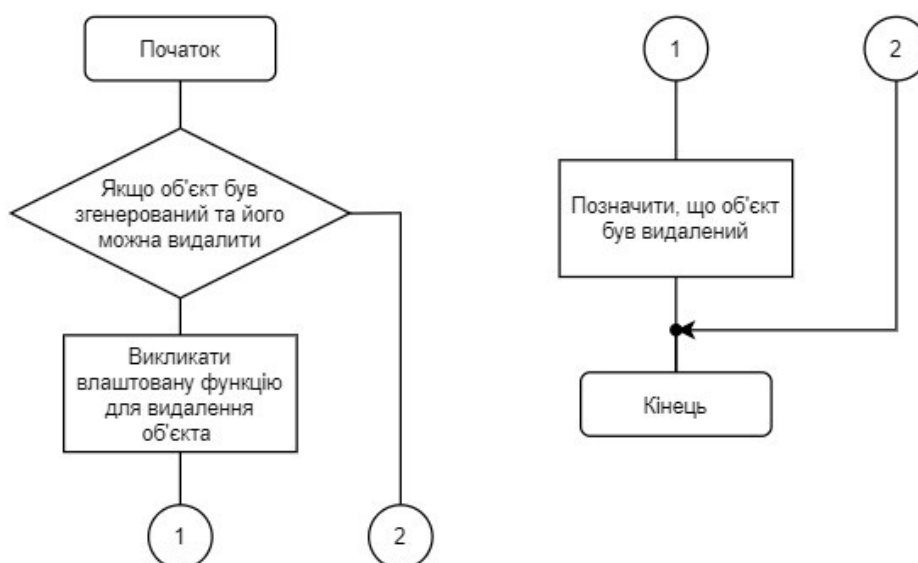


Рис. 3.19 Блок-схема алгоритму для видалення об'єктів

Блок-схему алгоритму, призначеного для перевірки зібраної конфігурації ПК, представлено на рис. 3.20.

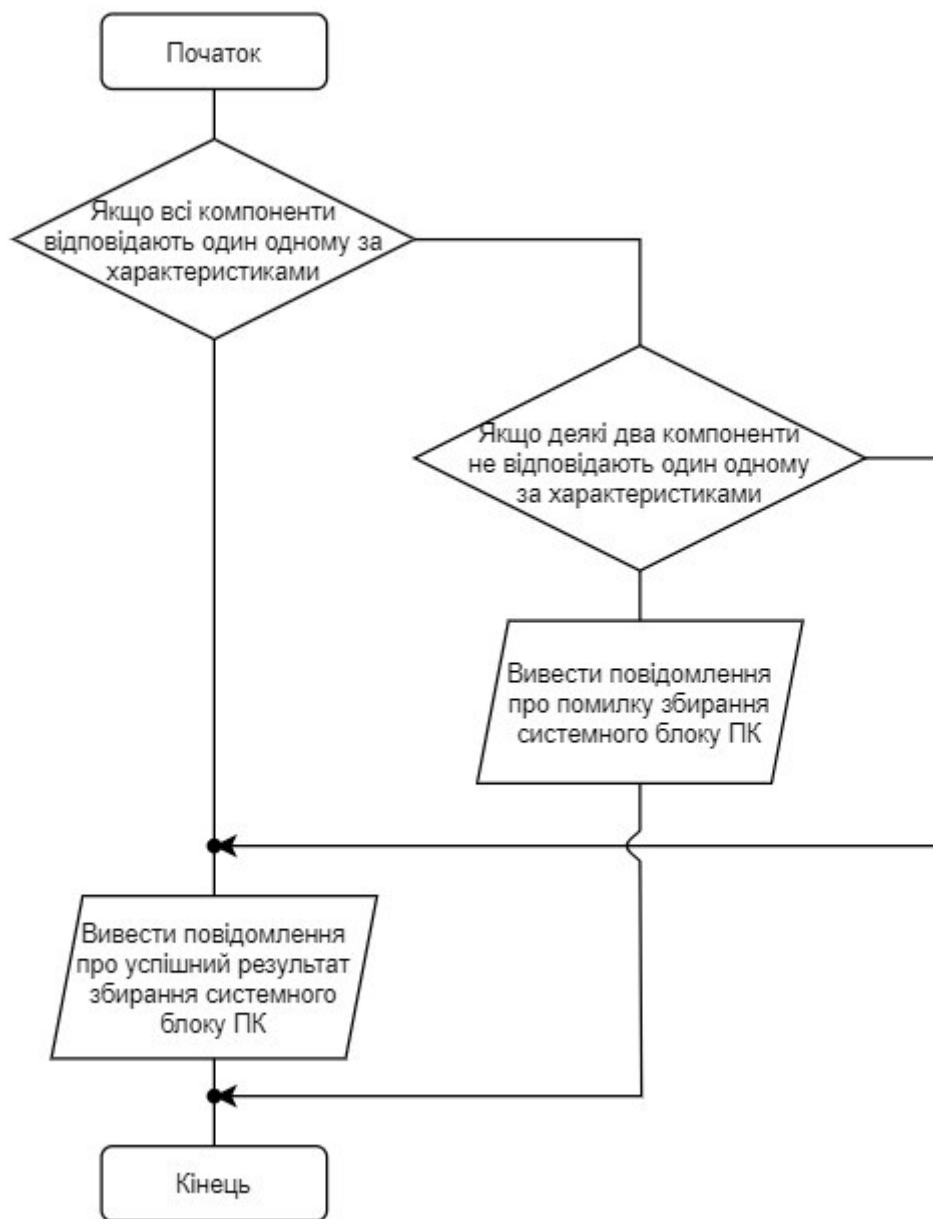


Рисунок 3.20 Блок-схема алгоритму для перевірки зібраної конфігурації ПК

Для програмування симулятора для збирання системного блоку ПК було створено вісім класів, які забезпечують реалізацію логіки. На рис 3.21 представлено структуру UML-діаграму класів.

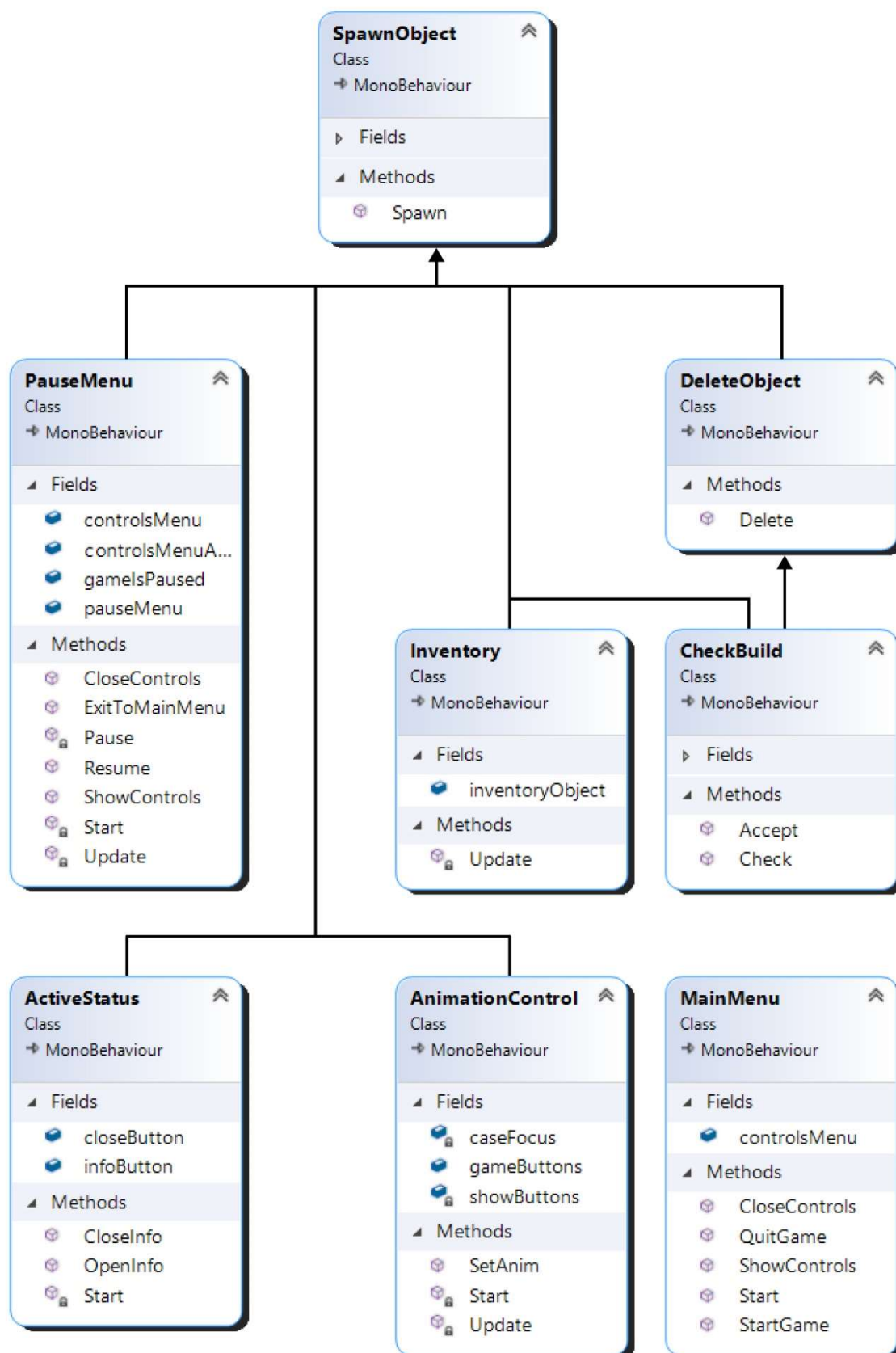


Рисунок 3.21 UML-діаграма класів програми з різними видами зв'язку

SpawnObject є головним класом серед всіх. Його функції відповідають за процес генерації об'єкта на ігровій сцені. Всі класи, окрім MainMenu, є дочірніми класами. Це означає, що від батьківського класу SpawnObject залежить поведінка функцій інших класів.

Клас PauseMenu потрібен для виклику меню паузи, за допомогою якого користувач зможе призупинити процес симуляції. В меню також розташовані три кнопки, які виконують такі функції:

- а) кнопка «RESUME» – відповідає за відновлення процесу симуляції;
- б) кнопка «CONTROLS» – необхідна для перегляду інструкції користувача;
- в) кнопка «MAIN MENU» – призначена для виходу в головне меню.

Видалення останнього згенерованого об'єкта з корпусу здійснюється за допомогою класу DeleteObject. В залежності від значень певних змінних у класі SpawnObject, функція визначає який компонент потрібно видалити.

Клас CheckBuild необхідний для перевірки конфігурації створеної користувачем. Всього існує два варіанти розвитку події:

Якщо компоненти були підібрані не за відповідними характеристиками, тоді виводиться повідомлення про не коректну конфігурацію, а також вказується, що саме не сумісно з іншими компонентами.

Якщо компоненти відповідають один одному за характеристиками, тоді виводиться повідомлення про правильну зібрану конфігурацію.

Залежність класу CheckBuild від класу DeleteObject обумовлена тим, що таким зв'язком накладається обмеження – якщо не згенеровані всі складові системного блоку ПК, тоді здійснити перевірку неможливо.

Inventory – необхідний клас для виконання маніпуляції з інвентарем. В залежності від значень певних змінних у класі SpawnObject, при виклику інвентарю буде відображено деяка група компонентів, наприклад, після генерації материнської плати, в інвентарі буде відображено тільки процесори.

Клас ActiveStatus відповідає за стан спеціального вікна з детальною інформацією про конкретний компонент (назва та характеристики). Такий

клас створений з метою обмежити деякий функціонал програми, щоб уникнути помилок.

Клас `AnimationControl` призначений для активації анімації, яка потрібна для фокусування на певну ігрову область, тільки один раз і в тому випадку, коли на сцені з'являється корпус. Головним об'єктом анімації є камера, що відображає на екран монітора ігрове середовище користувачу.

`MainMenu` є незалежним класом. Його взаємозв'язок з іншими класами відсутній, тому що він містить функції, які викликаються тільки кнопками у головному меню, а саме:

- а) кнопка «START GAME» – відповідає за початок процесу симуляції;
- б) кнопка «CONTROLS» – необхідна для перегляду інструкції користувача;
- в) кнопка «QUIT» – призначена для виходу з програми.

3.4 Програмування, збірка та тестування симулятора

Завершальний етап реалізації навчального симулятора, призначеного для збирання системного блоку ПК, полягає в написанні скриптів, компонуванні 3D-моделей та оформленні ігрових сцен у 3D-середовищі за допомогою ігрового рушія Unity[29]. Графічний інтерфейс користувача програми має схожі елементи у порівнянні з ПЗ Blender.

Для початку розробки симулятора було створено новий проєкт з 3D-середовищем з двома порожніми ігровими сценами (система освітлення та камера додаються за замовчуванням).

Першим кроком було реалізовано сцену, що містить графічні елементи інтерфейсу головного меню. За допомогою UI-об'єктів створено область, на якій розміщено текст (заголовок та авторське право) та кнопки для виклику певних функцій. Для початку симуляції було створено скрипт `MainMenu.cs` (див. додаток Б), який містить функцію, що викликається при натисканні на

кнопку «START GAME». Вона відповідає за зміну сцени з головним меню на основну ігрову сцену, де відбувається весь процес симуляції збирання.

Головне меню програмного додатку представлено на рис. 3.22.



Рис 3.22 Головне меню симулятора збирання системного блоку ПК

Також даний скрипт містить функцію, необхідну для відображення меню з інструкцією користувача, яка викликається при натисканні на кнопку «CONTROLS» (рис. 3.23). Натиснувши на кнопку «CLOSE», користувач повертається до головного меню. Остання кнопка «QUIT» призначення для виходу з програми.



Рис. 3.23 Меню з інструкцією користувача

Надалі реалізовано основну ігрову сцену, призначену для проведення процесу симуляції. Вона містить всі 3D-об'єкти, систему освітлення, камеру та деякі елементи графічного інтерфейсу.

Спочатку було спроектовано область на ігровій сцені, в якій розміщуються 3D-моделі складових системного блоку ПК (рис. 3.24).

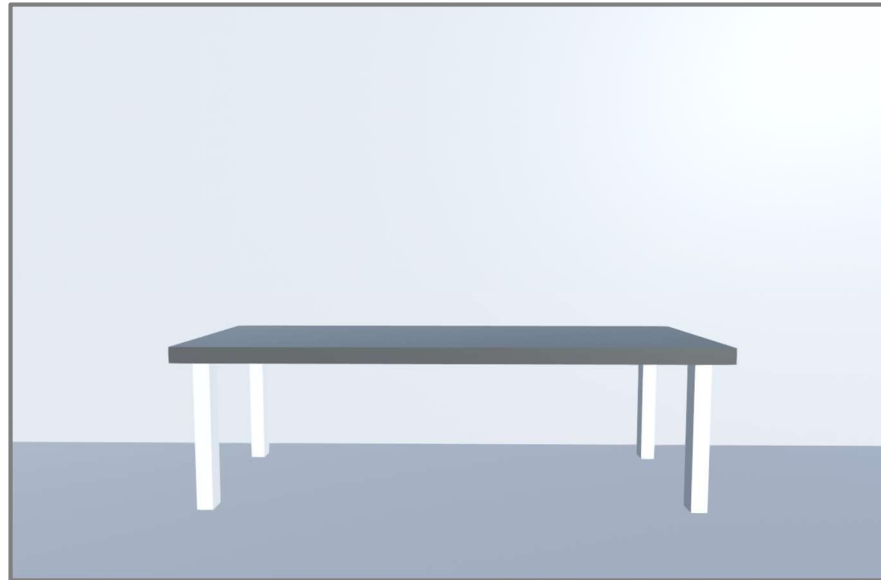


Рис. 3.24 Область ігрової сцени для розміщення компонентів ПК

Для реалізації графічного інтерфейсу користувача, що з'являється під час симуляції, створено за допомогою спеціальних UI-об'єктів як у попередній сцені. До створеної області з інтерфейсом було прив'язано основну камеру, а після початку процесу симуляції буде відображено спеціальне вікно для вибору корпусу (рис. 3.25).



Рис.3.25 Інтерфейс меню для вибору корпусу

Зображення з корпусом є кнопкою, яка при натисканні відображає вікно з детальним описом його характеристик (рис. 3.26).

Для генерації 3D-моделей компонентів було створено скрипт `SpawnObject.cs` (див. додаток В). При натисканні на кнопку «Spawn» генерується 3D-модель корпусу на робочому столі. Кнопка «Close» реалізована для виходу у попереднє меню для вибору іншого корпусу.



Рис. 3.26 Вікно з детальним описом характеристик корпусу

Після додавання першого об'єкта, за допомогою реалізованого скрипту `AnimationControl.cs` (див. додаток Г), запускається анімація камери, яка наближає його до робочого місця (рис. 3.27). Для маніпуляції з інвентарем було створено скрипт `Inventory.cs` (див. додаток Д), який відображає конкретну групу компонентів ПК (наприклад, тільки процесори або оперативну пам'ять) в залежності від згенерованих елементів.

При натисканні на кнопку із зображенням компонента, яка розташована в області інвентарю, буде відкрито вікно з детальним описом характеристик певного елемента системного блоку ПК (рис. 3.28).



Рис. 3.27 Вигляд основної ігрової сцени з інтерфейсом користувача

Для відстежування стану вікна з описом характеристик елемента було реалізовано скрипт `ActiveStatus.cs` (див. додаток Д) з метою обмежити деякий функціонал програми, щоб уникнути помилок.



Рис. 3.28 Вікно з детальним описом характеристик компонента

Для видалення останнього згенерованого об'єкта було реалізовано скрипт `DeleteObject.cs` (див. додаток Е), який містить функцію, що викликається при натисканні на кнопку «Delete». Щоб перевірити

правильність підібраних компонентів, створено скрипт CheckBuild.cs (див. додаток Ж). Він спрацьовує при натисканні на кнопку «Check» за умови, що всі необхідні елементи системного блоку ПК були влаштовані в корпус, як представлено на рисунку 3.29.

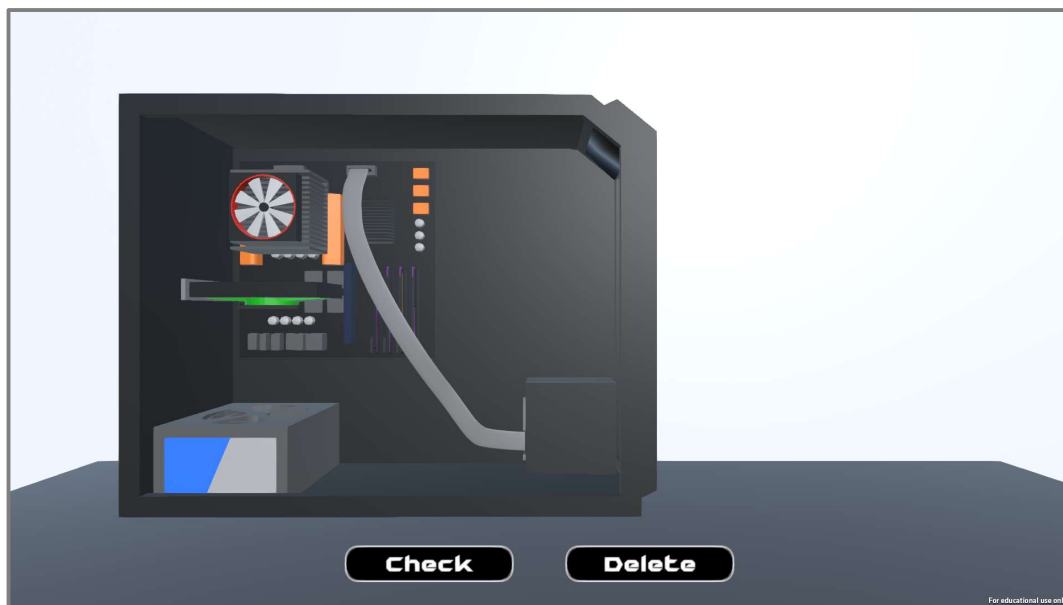


Рис. 3.29 Готова конфігурація системного блоку ПК

Якщо користувач підібрав компоненти за характеристиками, які не є сумісними, тоді, при натисканні на кнопку «Check» відображається вікно з повідомленням про неправильну конфігурацію з причиною помилки (рис. 3.30).

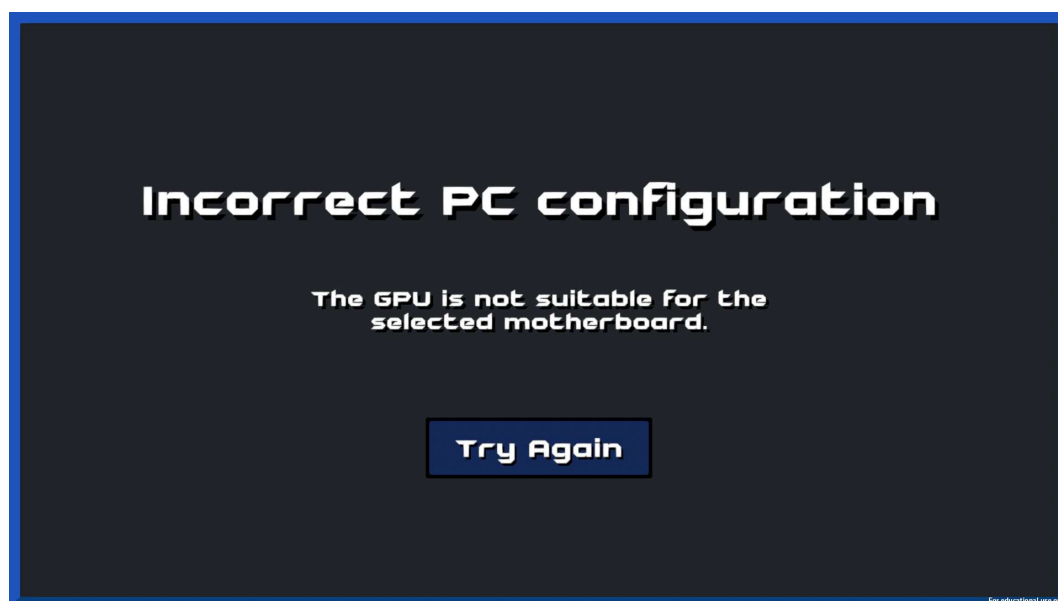


Рис. 3.30 Вікно з повідомленням про несумісність компонентів

В іншому випадку, якщо користувач зробив все вірно, при натисканні на кнопку «Check» відображається вікно з повідомленням про успішне збирання конфігурації (рис. 3.31).



Рис.3.31 Вікно з повідомленням про успішне збирання системного блоку

Для зручності було створено скрипт PauseMenu.cs (див. додаток II) для призупинення процесу симуляції, а також надання доступу до деяких функцій програми (рис. 3.32). Таким чином, за допомогою цього вікна, у користувача є можливість повернутись до головного меню або, за потреби, подивитись інструкцію користувача. Для відновлення процесу симуляції необхідно натиснути на кнопку «RESUME» або кнопку Esc на клавіатурі.



Рис. 3.32 Вікно меню паузи

Для тестування симулятора було використано Profiler в Unity –це інструмент, який дозволяє аналізувати та відстежувати різні аспекти продуктивності вашої гри або додатку. Він дозволяє перевірити використання центрального процесора (CPU), відеокарти (GPU) та інших системних ресурсів[30].

Цей інструмент допомагає виявляти місця у коді, де може відбуватися зниження продуктивності або де можуть виникати проблеми, що уповільнюють роботу вашого додатку.

Profiler в Unity надає різноманітну інформацію про час, який витрачається на відображення кожного кадру, використання пам'яті, виклики функцій, ресурси, використані для відображення графіки та багато іншого. Він також надає графічне відображення цих даних, що спрощує аналіз та виявлення можливих проблем у додатку.

Результати тестування за допомогою Profiler представлені на рис. 3.33.

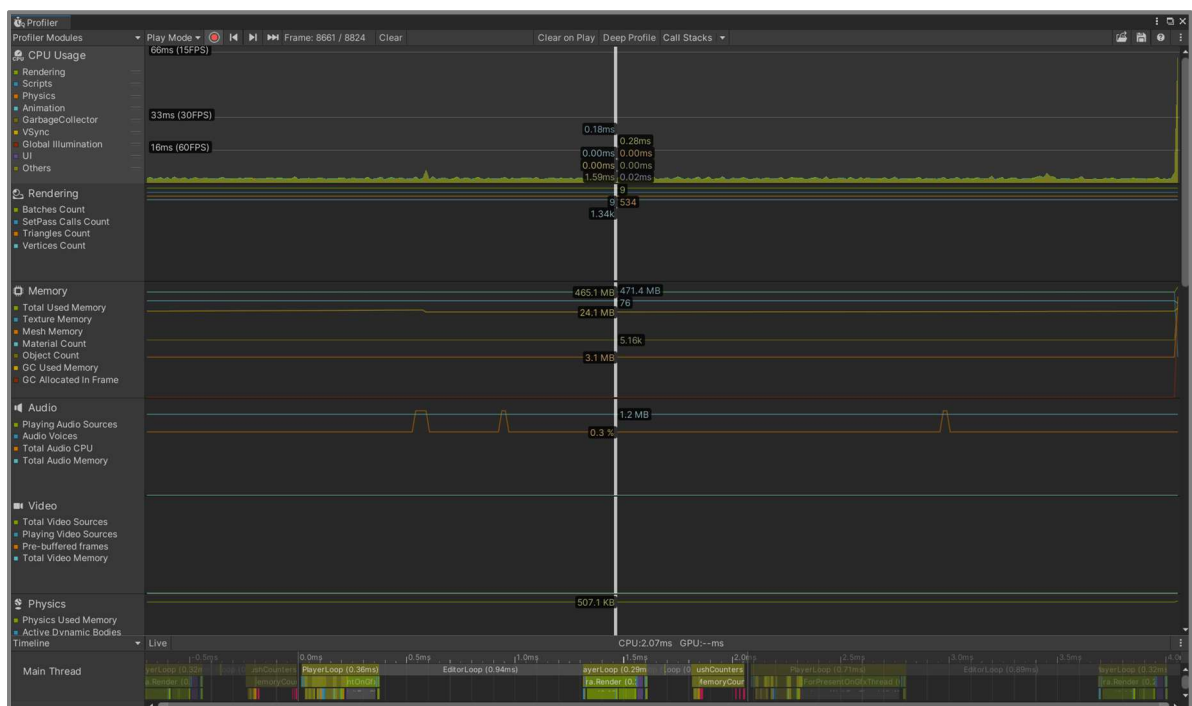


Рис. 3.33 Результати тестування роботи програми інструментом Profiler

Після проведених тестувань було визначено такі мінімальні системні вимоги для симулятора збирання системного блоку ПК:

- а) операційна система: Microsoft Windows 7/8.1/10;

- б) процесор: Intel Core 2 Duo 2.66 GHz;
- в) оперативна пам'ять: 1 GB;
- г) відеокарта: Hardware Accelerated Graphics with 1GB;
- д) жорсткий диск: 200 MB вільного простору.

3.5 Техніко-економічне обґрунтування розробки симулятора

Техніко-економічне обґрунтування вартості розробки програмного забезпечення - це процес оцінки витрат і потенційних користей від розробки програмного продукту з точки зору технічної складності, затрат на ресурси, трудових витрат та очікуваних економічних переваг.

Для оцінки трудовитрат використано модель оцінки COSOMO 2 [31].

COSOMO (Constructive Cost Model) 2 - це модель оцінки витрат на розробку програмного забезпечення, яка використовується для прогнозування трудовитрат і термінів виконання проєктів програмування. Ця модель вважається покращенням оригінальної моделі COSOMO, яка була розроблена Беррі Боемом у 1981 році.

Для проведення розрахунків використано онлайн-калькулятор [32], а результати наведено у додатку К.

Орієнтовна трудомісткість проєкту оцінена у 4,06 людино-місяців.

Реальна кількість годин, що була витрачена на розробку становила в середньому 2,5 години на день. Загальна кількість годин витрачених на розробку склала 217,5 годин (2,5 години x 87 робочих днів).

Так як розробка виконувалась на базі закладу освіти, використовувались обчислювальна техніка та енергетичні і водні ресурси, праця діючого співробітника – автора роботи, відповідно можна виконати розрахунок потенційних витрат закладу освіти, які будуть складатися із заробітної плати із нарахуваннями, оплати електроенергії, водопостачання та водовідведення, амортизації основних засобів.

Розрахунок потенційних витрат на заробітну плату за час витрачений на розробку у табл. 3.1.

Таблиця 3.1

Розрахунок потенційних витрат на заробітну плату

| Кількість відпрацьованих годин | Вартість однієї години, грн. | Звання (15%), грн. | Вислуга років(30%), грн | Надбавка за престижність праці 30%, грн. | Сума, грн. | Нарахування на фонд заробітної плати(22%), грн. | Всього, грн. |
|--------------------------------|------------------------------|--------------------|-------------------------|--|------------|---|--------------|
| 217,5 | 51,87 | 1692,26 | 3384,52 | 3384,52 | 19743,02 | 4343,46 | 24 086,48 |

Вартість водопостачання та водовідведення, що витрачається на 1 людину наведена у табл. 3.2.

Таблиця 3.2

Розрахунок потенційних витрат на водопостачання та каналізацію плати

| Послуга | Вартість куб.м , грн. | Середня норма витрат на годину, куб. м. | Кількість годин робочого часу | Вартість послуги, грн. |
|----------------|-----------------------|---|-------------------------------|------------------------|
| Водопостачання | 14,30 | 0,00083 | 217,5 | 25,82 |
| Водовідведення | 17,50 | 0,00083 | 217,5 | 31,52 |
| Всього: | 57,34 грн. | | | |

Вартість електроенергії розрахуємо виходячи із потужності, що споживається комп'ютером на якому здійснювалась розробка, а саме 450 Вт на годину та витрат на освітлення приміщення, оскільки робота виконувалась переважно у вечірній час після основної роботи. Сукупна споживана

потужність освітлювальної системи приміщення, де проводилась розробка становить 90 Вт на годину. Розрахунок вартості електричної енергії наведений у табл. 3.3.

Таблиця 3.3

Розрахунок потенційних витрат на електроенергію

| Послуга | Вартість 1кВт*год | Витрата на годину, кВт | Кількість годин робочого часу | Вартість послуги, грн. |
|----------------|----------------------|---------------------------|--|---------------------------|
| Робота ПК | 8,00 | 0,45 | 217,5 | 783,00 |
| Освітлення | 8,00 | 0,09 | 217,5 | 156,60 |
| Всього: | | 939,60 грн. | | |

Амортизації нараховується у розмірі 0,00475% за 1 годину роботи від первісної вартості устаткування. Для ПК (у зборі) з первісної вартістю за даними бухгалтерського обліку 24650 грн. розмір амортизації складатиме $24650 \text{ грн.} \times 0,00475 / 100 \times 217,5 \text{ год.} = 254,67 \text{ грн.}$

Отже сукупні витрати на розробку складатимуть 25 338,085 грн.

Якщо припустити, що заклад освіти зможе забезпечити придбання для учнів індивідуальних версій програми PC Building Simulator за вартістю зазначеною у пункті 2.1., то витрати тільки на 1 групу учнів (30 осіб) складуть 12450 грн. Якщо навіть виходити із припущення, що симулятор буде використовуватися в процесі підготовки тільки однієї групи за професією «Оператор з обробки інформації та програмного забезпечення» щорічно, то його окупність складе лише трохи більше 2 років. Цей термін може буде зменшений за рахунок впровадження симулятора у підготовку кваліфікованих робітників за професією «Оператор комп'ютерного набору», а також «Зварник» (предмет «Інформаційні технології в галузях металообробки»).

3.6 Висновки до розділу 3

В третьому розділі описано процес практичної реалізації симулятор на рушії Unity 3d.

Неведено ескізи вікон програм із позначеннями елементів інтерфейсу. Описано створення тривимірних моделей корпусу та комплектуючих та наведено їх зразки. Наведені алгоритми реалізації логіки програми та описані класи мови C#, що розроблені для її забезпечення. Представлено діаграму клаїв. Описана процедура програмування, збірки та тестування, наведені зразки коду скриптів. В результаті тестування визначено мінімальні системні вимоги до ПК, який здатний підтримувати роботу симулятора.

Техніко-економічні розрахунки виконано з використанням моделі оцінки COSOMO 2 із застосуванням веб-додатку для проведення обчислень. Розраховані потенційні витрати на створення симулятора для закладу освіти, на базі якого виконувалась розробка. Показана можливість швидкої окупності проекту.

ЗАГАЛЬНІ ВИСНОВКИ

В кваліфікаційній роботі досліджено парадигму навчання засновану на використанні ігрових проєктів навчального призначення. Були проаналізовані роботи Елфіна Тофлера, Масара Ібуки, Ікуджіра Нонака, Хіротаки Такеучі, Тетяни Лугової, Джеймса Пола Гі, Джейн Макгонігал, Роберта Функе, Крістофера Шеллі, Джейсона Шока.

Визначено, що в рамках зазначеної парадигми знання представляється як результат активної діяльності, що містить елементи креативності та інноваційності, що людина не залежно від віку навчається через досвід, а її переживання та попередня діяльність має прямий вплив на знання, що формується.

Одним із перспективних засобів формування знання нового типу виступає геймфікація - це використання елементів гри у негравальних контекстах, наприклад зокрема в освіті для стимулювання участі, мотивації та досягнення певних цілей.

У сфері освіти гейміфікація є потужним інструментом для залучення уваги здобувачів, підвищення їх мотивації до навчання та покращення результатів. Вона допомагає краще засвоювати матеріал, сприяючи ігровому підходу до навчання, який часто більш привабливий та захоплюючий. Також сприяє розвитку співпраці, командної роботи та креативного мислення серед здобувачів.

Визначено складові освітньої гейміфікації та загальна структура геймдизайну, який розглянуто з точки зору проєктної діяльності, що використовує в якості методологічної основи моделі MDA та DFE.

Як вид ігрового програмного забезпечення досліджено програми-симулятори навчального призначення. Показано, що цей вид програмного забезпечення реалізує активне або дієве навчання, має великий потенціал для формування професійних компетентностей здобувачів освіти як під час аудиторного так і відділеного (у т.ч. дистанційного навчання).

Описано проблеми розробки симуляторів навчального призначення та їх актуальність для вітчизняної освіти.

Аналіз наявних на ринку симуляторів, які можна використати для симуляції збирання системного блоку довів, що кількість подібних програм є порівняно невеликою, а найбільш розповсюджені PC Building Simulator, 3DMark, PC Architect, Build Your Own PC Simulator для закладу освіти, як і для здобувачів не є безкоштовними, не задовольняють усім вимогам, що можуть бути висунені до ПЗ навчального призначення. Отже ще раз було підтверджено доцільність власної розробки.

Автором було запропоновано та описано власну функціональну модель програми-симулятора навчального призначення та представлено функціональну схему розробки. Детально розглянуто найбільш поширені сучасні методології розробки Waterfall, Agile, Scrum, Kanban, DevOps, Lean Software Development, RAD, XP. Для оцінки та вибору оптимальної методології розробки обрано метод складання оціночних матриць та запропоновано алгоритм їх використання. Було поєднано матрицю Річарда Уотера з матрицею Кокса та Фіндика та введено бальне оцінювання критеріїв, що дало змогу прийти до висновку поєднати методологій Waterfall та Lean Software Development та створити створити гібридну модель, яка поєднує в собі елементи структурованої послідовності та фокус на оптимізації процесів, мінімізації витрат. Класична каскадну модель життєвого циклу програмного забезпечення модифіковано з урахуванням гібридизації методологій.

Аналіз аналогічних Unity засобів розробки ігрових проєктів Unreal Engine, CryEngine, Godot показав, що саме Unity є оптимальною платформою та рушієм для створення ігрового проєкту навчального призначення.

Під час розгляду графічних редакторів, що можуть бути використані для розробки було визначено рекомендації до редактору та види застосування графіки в ігрових проєктах. Вибір графічного редактора здійснювався з таких продуктів як Blender 3D, Autodesk Maya, 3ds Max, Cinema 4D. За

функціоналом, вартістю продукту та іншими критеріями перевагу було надано Blender 3D.

Для здійснення планування та управління проєктом було проаналізовано наступні засоби та онлайн-платформи: Microsoft Project, Asana, Trello, Smartsheet, Google Sheets, Microsoft Excel, Canva. Враховуючи зручність користування сервісами та наявність готових шаблонів, вибір було зроблено на користь Google Sheets, які є частиною Google Workspace.

Для реалізації інтерфейса симулятора було розроблено ряд ескізів. Комплектуючі ПК та корпус для збирання були виконані у вигляді тривимірних моделей у форматі fbx. Логіку симулятора реалізовано з використанням мови програмування C#.

Тестування симулятора виконано за допомогою інструмента Profiler в Unity та отримано наступні системні вимоги до ПК, на яких може бути запущений симулятор:

- а) операційна система: Microsoft Windows 7/8.1/10;
- б) процесор: Intel Core 2 Duo 2.66 Гц;
- в) оперативна пам'ять: 1 Гб;
- г) відеокарта: Hardware Accelerated Graphics with 1Гб;
- д) жорсткий диск: 200 Мб вільного простору.

В техніко-економічному обґрунтуванні проєкту виконано розрахунок трудовитрат з використанням онлайн-сервісу, що здійснює обчислення на основі моделі оцінки СОСОМО 2. Виконано розрахунок вартості проєкту та визначено, що термін окупності для закладу освіти не перевищить двох років.

Результати виконаного дослідження апробовано на трьох науково-практичних конференціях:

- а) I Міжнародна науково-практична конференція «Current methods of improving outdated technologies and methods» (Більбао, Іспанія);
- б) XI Міжнародна науково-практична конференція «Modern problems of science, education and society» (Київ, Україна);

в) V міжнародна науково-практична конференція “Global science: prospects and innovations” (Ліверпуль, Великобританія).

Результати дослідження впровадження в освітній процес Вищого професійного училища № 7 м.Кременчука Полтавської області.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

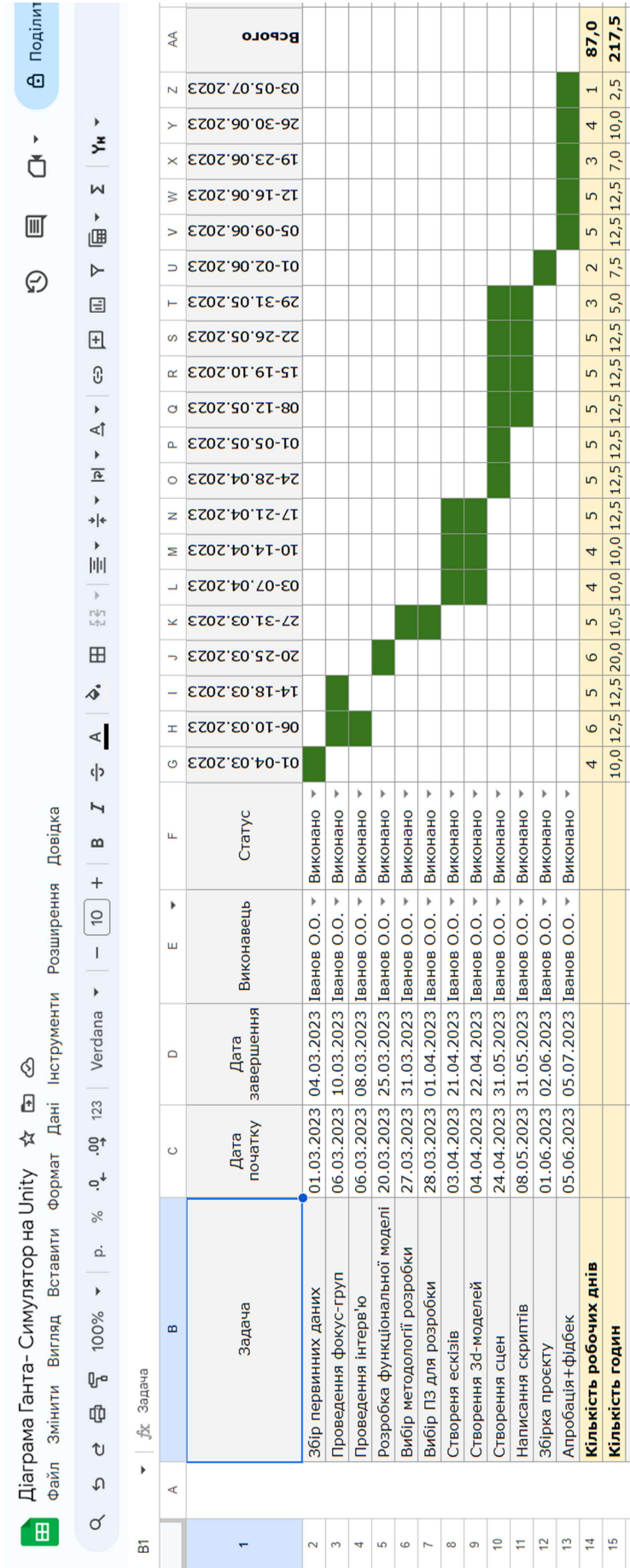
1. Іванов О. Використання хмарних сервісів для формування професійних компетентностей здобувачів освіти//Global science: prospects and innovations. Proceedings of the 5th International scientific and practical conference. Cognum Publishing House. Liverpool, United Kingdom. 2023. Pp.390-397.URL: <https://sci-conf.com.ua/wp-content/uploads/2023/12/GLOBAL-SCIENCE-PROSPECTS-AND-INNOVATIONS-28-30.12.23.pdf> (дата зверення 30.12.2023).
2. Alvin T. Future shock. New York: Random House, 1970. 505 p.
3. Масару Ібука. Після трьох вже пізно. Київ. Сварог. 2021. 152 с.
4. Nonaka, I. and Takeuchi, H. (1995) The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation. Oxford University Press, New York.
5. Лугова Т.А. Гейміфікація методів класифікації в контексті інформаційного суспільства/Т.А. Лугова, В.Р. Раєва//Філософія і гуманізм. Одеса, 2018. Вип.1(7). С.51-58. URL: http://www.philhum.esy.es/uploads/Fil_Hum_7.pdf (дата зверення 01.12.2023).
6. Gee James Paul. What Video Games Have to Teach Us about Learning and Literacy. New York: Palgrave Macmillan, 2003. Pp.2, 14, 203-210. URL: <http://newlearningonline.com/literacies/chapter-2/gee-on-whatvideo-games-have-to-teach-us-about-learning-and-literacy>(дата зверення 02.12.2023).
7. Zichermann Gabe. Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps/ G. Zichermann, C. Cunningham. Sebastol, California: O'Reilly Media. 257 p.
8. James Paul Gee. The Anti-Education Era: Creating Smarter Students through Digital Learning. St. Martin's Publishing Group, 2013. 256 p.
9. Sfiri Anastasia. Game-based Learning. URL: <https://pdfs.semanticscholar.org/presentation/d10c/a95c7c2143b6c4122b5d4946141db781643a.pdf>(дата зверення 02.12.2023).

10. Що таке дизайн відеоігор: основні принципи геймдизайну.
URL:<https://vokigames.com/ua/scho-take-dizajn-videoigor-osnovni-printsipi-gejmdizajnu/>(дата звернення 04.12.2023).
11. ISO/IEC/IEEE 15288:2023 Systems and software engineering - System life cycle processes. URL: <https://www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:15288:ed-2:v1:en/>(дата звернення 04.12.2023)..
12. Jane McGonigal Reality Is Broken: Why Games Make Us Better and How They Can Change the World. URL: https://hci.stanford.edu/courses/cs047n/readings/Reality_is_Broken.pdf(дата звернення 04.12.2023).
13. Hunicke R., LeBlanc M., Zubec R. MDA: A Formal Approach to Game Design and Game Research Hunicke R. MDA: A formal approach to game design and gameresearch. In: Proceedings of the AAAI Workshop on Challenges in Game AI. vol. 4, 2004.
14. Winn B. The Design, Play, and Experience Framework. Handbook of Research on Effective Electronic Gaming in Education. Richard Ferdig (editor), 2009. Volume 3, Chapter 58.
15. Загірняк М.В. Віртуальні лабораторні системи і комплекси – нова перспектива наукового пошуку і підвищення якості підготовки фахівців з електромеханіки/ М.В.Загірняк, Д.Й.Родькін, О.П.Чорний// Електромеханічні і енергозберігаючі системи. 2009. Вип. 2/2009 (6). С. 2-7(дата звернення 05.12.2023).
16. Довідка за результатами вивчення питання щодо організованого початку 2022/2023 навчального року у закладах професійної (професійно-технічної) освіти. Державна служба якості освіти. URL: https://sqe.gov.ua/wp-content/uploads/2022/11/Dovidka_vivchennya_PP-TO_2022-2023_SQE.pdf(дата звернення 06.12.2023).
17. Іванов О. Функціональна модель програми-симулятора навчального призначення. Abstracts of I International Scientific and Practical Conference. Bilbao, Spain. Pp. 263-266. URL: <https://eu-conf.com/wp->

- content/uploads/2023/11/CURRENT-METHODS-OF-IMPROVING-OUTDATED-TECHNOLOGIES-AND-METHODS.pdf (дата звернення 10.01.2024).
- 18.Стисло Т. Р. Методологія розробки програмного забезпечення / Т. Р. Стисло, О. В. Стисло//Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 72): матеріали Міжнародної наукової інтернет-конференції (Тернопіль, Україна - Переворськ, Польща, 15-16 листоп. 2022 р.) / ГО "Наукова спільнота"; WSSG w Przeworsku. - Тернопіль: ФОП Шпак В. Б. - С. 66-67.
 - 19.Pressman R. S. Software engineering: A practitioner's approach. 7th ed. Dubuque, IA : McGraw-Hill, 2009.
 - 20.Cockburn A. Crystal clear: A human-powered methodology for small teams. Boston : Addison-Wesley, 2005. 312 p.
 - 21.Іванов О. Традиційні та гібридні методології розробки програмного забезпечення: обґрунтування вибору// Modern problems of science, education and society. Proceedings of the 9th International scientific and practical conference. SPC "Sci-conf.com.ua". Kyiv, Ukraine. 2023. Pp.388-393.URL: <https://sci-conf.com.ua/wp-content/uploads/2024/01/MODERN-PROBLEMS-OF-SCIENCE-EDUCATION-AND-SOCIETY-8-10.01.24.pdf>.
 - 22.Технології програмування та створення програмних продуктів: конспект лекцій/укладач О. В. Алексенко. Суми: Сумський державний університет, 2013. 133 с.
 - 23.Розробка комп'ютерних ігор за допомогою Unity 3D: електронний навчальний посібник для підготовки студентів спеціальності 121 «Інженерія програмного забезпечення»/Укладач: О.М. Ляшенко. Херсон: видавництво ФОП Вишемирський В.С., 2018. 220 с.
 - 24.Unity проти Unreal Engine - який рушій обрати для гри та чому. Зважуємо всі за та проти з розробниками. URL:<https://gamedev.dou.ua/articles/unity-or-unreal-engine/> (дата звернення 07.12.2023).

25. Gantzler T. Game Development Essentials: Video Game Art. Cengage Delmar Learning, 2004. 320 p.
26. Діаграма Ганта: опис, особливості та призначення. URL:<https://rd168.com.ua/diagrama-ganta-opis-osoblivosti-ta-priznachennja/> (дата звернення 07.12.2023).
27. Безкоштовні шаблони діаграми Ганта в Excel, Google таблицях та GanttPRO. URL:<https://ganttpro.com/ru/gantt-chart-template/> (дата звернення 08.12.2023).
28. Sketching User Experiences: The Workbook / B. Buxton et al. Morgan Kaufmann, 2011. 272 p.
29. Unity Manual. <https://docs.unity3d.com/Manual/> (дата звернення 08.12.2023).
30. Навчальний модуль «Розробка комп'ютерних ігор за допомогою Unity 3D» для підготовки студентів спеціальності 121 «Інженерія програмного забезпечення»/О.М. Ляшенко. Херсон: ХНТУ, 2017. 220 с.
31. Barry Boehm. Cost Estimation with COCOMO II. URL: https://www.researchgate.net/publication/228600814_Cost_estimation_with_COCOMO_II (дата звернення 08.12.2023).
32. COCOMO. URL:<https://cocomo.vercel.app/calc> (дата звернення 08.12.2023).

ДОДАТОК А. Діаграма Ганта



ДОДАТОК Б. Лістинг реалізації програмного коду скрипта MainMenu.cs

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

public class MainMenu : MonoBehaviour
{
    public GameObject controlsMenu;

    public void Start()
    {
        controlsMenu.SetActive(false);
    }

    public void StartGame()
    {
        SceneManager.LoadScene(1);
    }

    public void ShowControls()
    {
        controlsMenu.SetActive(true);
    }

    public void CloseControls()
    {
        controlsMenu.SetActive(false);
    }

    public void QuitGame()
    {
        Application.Quit();
    }
}
```


ДОДАТОК В. Лістинг реалізації програмного коду скрипта SpawnObject.cs

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

public class SpawnObject : MonoBehaviour
{
    public Transform spawnPosition;
    public GameObject objectToSpawn;

    public static bool infoActiveStatus = false;

    public static bool case_is_spawned = false;
    public static string case_name;

    public static bool mb_is_spawned= false;
    public static bool mb_can_delete = true;
    public static string mb_name;

    public static bool cpu_is_spawned = false;
    public static bool cpu_can_delete = true;
    public static string cpu_name;

    public static bool fan_is_spawned = false;
    public static bool fan_can_delete = true;
    public static string fan_name;

    public static bool gpu_is_spawned = false;
    public static bool gpu_can_delete = true;
    public static string gpu_name;

    public static bool ram_is_spawned = false;
    public static bool ram_can_delete = true;
    public static string ram_name;

    public static bool hdd_is_spawned = false;
    public static bool hdd_can_delete = true;

    public static bool psu_is_spawned = false;
    public static bool psu_can_delete = true;
    public static string psu_name;

    public void Spawn()
    {
        if (
            !case_is_spawned
            && objectToSpawn.CompareTag("Case")
        ) {
```

```

        Instantiate(
            objectToSpawn,
            spawnPosition.position,
            Quaternion.Euler(-90, (float)90.5, 0)
        );
        case_is_spawned = true;
        infoActiveStatus = false;
        case_name = objectToSpawn.ToString();
    }

    if (
        !mb_is_spawned
        && objectToSpawn.CompareTag("MB")
    ) {
        Instantiate(
            objectToSpawn,
            spawnPosition.position,
            Quaternion.Euler(0, 0, -90)
        );
        mb_is_spawned = true;
        infoActiveStatus = false;
        mb_name = objectToSpawn.ToString();
    }

    if (
        mb_is_spawned
        && !cpu_is_spawned
        && objectToSpawn.CompareTag("CPU")
    ) {
        Instantiate(
            objectToSpawn,
            spawnPosition.position,
            Quaternion.Euler(0, 0, 0)
        );
        cpu_is_spawned = true;
        mb_can_delete = false;
        infoActiveStatus = false;
        cpu_name = objectToSpawn.ToString();
    }

    if (
        cpu_is_spawned
        && !fan_is_spawned
        && objectToSpawn.CompareTag("Fan")
    ) {
        Instantiate(
            objectToSpawn,
            spawnPosition.position,
            Quaternion.Euler(0, 0, 0)
        );
        fan_is_spawned = true;
    }

```

```

        cpu_can_delete = false;
        infoActiveStatus = false;
        fan_name = objectToSpawn.ToString();
    }

    if (
        fan_is_spawned
        && !gpu_is_spawned
        && objectToSpawn.CompareTag("GPU")
    ) {
        Instantiate(
            objectToSpawn,
            spawnPosition.position,
            Quaternion.Euler(90, 0, 90)
        );
        gpu_is_spawned = true;
        fan_can_delete = false;
        infoActiveStatus = false;
        gpu_name = objectToSpawn.ToString();
    }

    if (
        gpu_is_spawned
        && !ram_is_spawned
        && objectToSpawn.CompareTag("RAM")
    ) {
        Instantiate(
            objectToSpawn,
            spawnPosition.position,
            Quaternion.Euler(0, 0, 0)
        );
        ram_is_spawned = true;
        gpu_can_delete = false;
        infoActiveStatus = false;
        ram_name = objectToSpawn.ToString();
    }

    if (
        ram_is_spawned
        && !hdd_is_spawned
        && objectToSpawn.CompareTag("HDD")
    ) {
        Instantiate(
            objectToSpawn,
            spawnPosition.position,
            Quaternion.Euler(0, 90, 0)
        );
        hdd_is_spawned = true;
        ram_can_delete = false;
        infoActiveStatus = false;
    }
}

```

```

if (
    hdd_is_spawned
    && !psu_is_spawned
    && objectToSpawn.CompareTag("PSU")
) {
    Instantiate(
        objectToSpawn,
        spawnPosition.position,
        Quaternion.Euler(-90, 0, 180)
    );
    psu_is_spawned = true;
    hdd_can_delete = false;
    infoActiveStatus = false;
    psu_name = objectToSpawn.ToString();
}
}
}

```

ДОДАТОК Г. Лістинг реалізації програмного коду скрипта
AnimationControl.cs

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

public class AnimationControl : MonoBehaviour
{
    private Animator caseFocus;
    private Animator showButtons;
    public GameObject gameButtons;

    void Start()
    {
        caseFocus = GetComponent<Animator>();
        showButtons = gameButtons.GetComponent<Animator>();
    }

    void Update()
    {
        if (SpawnObject.case_is_spawned)
        {
            caseFocus.enabled = true;
            SetAnim(1);
        }
    }

    public void SetAnim(int id)
    {
        showButtons.SetInteger("Start", id);
    }
}
```

ДОДАТОК Д. Лістинг реалізації програмного коду скрипта Inventory.cs

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

public class Inventory : MonoBehaviour
{
    public GameObject inventoryObject;

    void Update()
    {
        if (
            SpawnObject.case_is_spawned
            && !SpawnObject.infoActiveStatus
        ) {
            if (Input.GetKeyDown(KeyCode.Tab))
            {
                if (
                    !SpawnObject.mb_is_spawned
                    && inventoryObject.CompareTag("InventoryMB")
                ) {

inventoryObject.SetActive(!inventoryObject.activeSelf);
                }

                else if (
                    SpawnObject.mb_is_spawned
                    && !SpawnObject.cpu_is_spawned
                    && inventoryObject.CompareTag("InventoryCPU")
                ) {

inventoryObject.SetActive(!inventoryObject.activeSelf);
                }

                else if (
                    SpawnObject.cpu_is_spawned
                    && !SpawnObject.fan_is_spawned
                    && inventoryObject.CompareTag("InventoryFan")
                ) {

inventoryObject.SetActive(!inventoryObject.activeSelf);
                }

                else if (
                    SpawnObject.fan_is_spawned
                    && !SpawnObject.gpu_is_spawned
                    && inventoryObject.CompareTag("InventoryGPU")
                ) {
```

```

inventoryObject.SetActive(!inventoryObject.activeSelf);
    }

    else if (
        SpawnObject.gpu_is_spawned
        && !SpawnObject.ram_is_spawned
        && inventoryObject.CompareTag("InventoryRAM")
    ) {

inventoryObject.SetActive(!inventoryObject.activeSelf);
    }

    else if (
        SpawnObject.ram_is_spawned
        && !SpawnObject.hdd_is_spawned
        && inventoryObject.CompareTag("InventoryHDD")
    ) {

inventoryObject.SetActive(!inventoryObject.activeSelf);
    }

    else if (
        SpawnObject.hdd_is_spawned
        && !SpawnObject.psu_is_spawned
        && inventoryObject.CompareTag("InventoryPSU")
    ) {

inventoryObject.SetActive(!inventoryObject.activeSelf);
    }
}
}
}
}
}
}
}

```

ДОДАТОК Е. Лістинг реалізації програмного коду скрипта ActiveStatus.cs

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

public class ActiveStatus : MonoBehaviour
{
    public Button infoButton;
    public Button closeButton;

    void Start()
    {
        infoButton.onClick.AddListener(OpenInfo);
        closeButton.onClick.AddListener(CloseInfo);
    }

    public void OpenInfo()
    {
        SpawnObject.infoActiveStatus = true;
    }

    public void CloseInfo()
    {
        SpawnObject.infoActiveStatus = false;
    }
}
```


ДОДАТОК Ж. Лістинг реалізації програмного коду скрипта DeleteObject.cs

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

public class DeleteObject : MonoBehaviour
{
    public static void Delete()
    {
        if (SpawnObject.mb_can_delete)
        {
            Destroy(GameObject.FindGameObjectWithTag("MB"));
            SpawnObject.mb_is_spawned = false;
        }

        if (SpawnObject.cpu_can_delete)
        {
            Destroy(GameObject.FindGameObjectWithTag("CPU"));
            SpawnObject.cpu_is_spawned = false;
            SpawnObject.mb_can_delete = true;
        }

        if (SpawnObject.fan_can_delete)
        {
            Destroy(GameObject.FindGameObjectWithTag("Fan"));
            SpawnObject.fan_is_spawned = false;
            SpawnObject.cpu_can_delete = true;
        }

        if (SpawnObject.gpu_can_delete)
        {
            Destroy(GameObject.FindGameObjectWithTag("GPU"));
            SpawnObject.gpu_is_spawned = false;
            SpawnObject.fan_can_delete = true;
        }

        if (SpawnObject.ram_can_delete)
        {
            Destroy(GameObject.FindGameObjectWithTag("RAM"));
            SpawnObject.ram_is_spawned = false;
            SpawnObject.gpu_can_delete = true;
        }

        if (SpawnObject.hdd_can_delete)
        {
            Destroy(GameObject.FindGameObjectWithTag("HDD"));
            SpawnObject.hdd_is_spawned = false;
            SpawnObject.ram_can_delete = true;
        }
    }
}
```

```
    if (SpawnObject.psu_is_spawned)
    {
        Destroy(GameObject.FindGameObjectWithTag("PSU"));
        SpawnObject.psu_is_spawned = false;
        SpawnObject.hdd_can_delete = true;
    }
}
```

ДОДАТОК 3. Лістинг реалізації програмного коду скрипта CheckBuild.cs

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

public class CheckBuild : MonoBehaviour
{
    public GameObject resultInfo;
    public TextMeshProUGUI result;
    public TextMeshProUGUI resultShadow;
    public TextMeshProUGUI description;
    public TextMeshProUGUI descriptionShadow;
    public GameObject winMenu;
    public GameObject errorMenu;
    string errorTitle = "Incorrect PC configuration";
    string correctBuildTitle = "Congratulations!";
    string errorDescriptionMB = "The motherboard is not suitable
for the selected case.";
    string errorDescriptionCPU = "The CPU is not suitable for the
selected motherboard.";
    string errorDescriptionFan = "The Fan is not suitable for the
selected motherboard.";
    string errorDescriptionGPU = "The GPU is not suitable for the
selected motherboard.";
    string errorDescriptionRAM = "The RAM is not suitable for the
selected motherboard.";
    string errorDescriptionPSU = "The PSU is not suitable for the
selected case.";
    public static string[] case_black = { "ATX" };
    public static string[] case_dark_grey = { "ATX" };
    public static string[] case_grey = { "MicroATX" };
    public static string[] mb_purple = { "AM4", "DDR4", "PCI-E 3.0",
"PCI-E 4.0", "ATX" };
    public static string[] mb_orange = { "1700", "DDR5", "PCI-E 4.0",
"PCI-E 5.0", "ATX" };
    public static string[] mb_blue = { "AM5", "DDR5", "PCI-E 3.0",
"PCI-E 4.0", "ATX" };
    public static string[] mb_green = { "1700", "DDR5", "PCI-E 3.0",
"PCI-E 5.0", "ATX" };
    public static string[] mb_red = { "1151", "DDR4", "PCI-E 3.0", "-
", "MicroATX" };

    public static string[] cpu_intel_i5 = { "1151" };
    public static string[] cpu_intel_i7 = { "1700" };
    public static string[] cpu_intel_i9 = { "1700" };
    public static string[] cpu_ryzen_5 = { "AM4" };
    public static string[] cpu_ryzen_7 = { "AM4" };
```

```

public static string[] cpu_ryzen_9 = { "AM5" };
public static string[] fan_purple = { "1151", "-", "AM4", "-" };
public static string[] fan_orange = { "1151", "1700", "AM4", "AM5" };
};

public static string[] fan_blue = { "1151", "-", "AM4", "-" };
public static string[] fan_green = { "1151", "-", "AM4", "AM5" };
public static string[] fan_red = { "1151", "1700", "AM4", "AM5" };
public static string[] gpu_purple = { "PCI-E 3.0" };
public static string[] gpu_orange = { "PCI-E 3.0" };
public static string[] gpu_blue = { "PCI-E 4.0" };
public static string[] gpu_green = { "PCI-E 3.0" };
public static string[] gpu_red = { "PCI-E 4.0" };
public static string[] gpu_cyan = { "PCI-E 4.0" };
public static string[] ram_purple = { "DDR4" };
public static string[] ram_orange = { "DDR4" };
public static string[] ram_blue = { "DDR5" };
public static string[] ram_green = { "DDR5" };
public static string[] ram_red = { "DDR4" };
public static string[] psu_blue = { "MicroATX" };
public static string[] psu_green = { "ATX" };
public static string[] psu_red = { "ATX" };
private string[] current_case;
private string[] current_mb;
private string[] current_cpu;
private string[] current_fan;
private string[] current_gpu;
private string[] current_ram;
private string[] current_psu;
public void Accept()
{
    while (SpawnObject.mb_is_spawned)
    {
        DeleteObject.Delete();
    }

    Destroy(GameObject.FindGameObjectWithTag("Case"));
    SpawnObject.case_is_spawned = false;
    SpawnObject.infoActiveStatus = false;

    SceneManager.LoadScene(0);
}

public void Check()
{
    if (SpawnObject.psu_is_spawned)
    {
        resultInfo.SetActive(true);

        switch (SpawnObject.case_name)
        {
            case "Case [Black] (UnityEngine.GameObject)":

```

```

        current_case = case_black;
        break;

    case "Case [Dark Grey] (UnityEngine.GameObject)":
        current_case = case_dark_grey;
        break;

    case "Case [Grey] (UnityEngine.GameObject)":
        current_case = case_grey;
        break;

    default:
        Debug.Log("No matches");
        break;
}
switch (SpawnObject.mb_name)
{
    case "Motherboard [Purple] (UnityEngine.GameObject)":
        current_mb = mb_purple;
        break;

    case "Motherboard [Orange] (UnityEngine.GameObject)":
        current_mb = mb_orange;
        break;

    case "Motherboard [Blue] (UnityEngine.GameObject)":
        current_mb = mb_blue;
        break;

    case "Motherboard [Green] (UnityEngine.GameObject)":
        current_mb = mb_green;
        break;

    case "Motherboard [Red] (UnityEngine.GameObject)":
        current_mb = mb_red;
        break;

    default:
        Debug.Log("No matches");
        break;
}
switch (SpawnObject.cpu_name)
{
    case "CPU [I5] (UnityEngine.GameObject)":
        current_cpu = cpu_intel_i5;
        break;

    case "CPU [I7] (UnityEngine.GameObject)":
        current_cpu = cpu_intel_i7;
        break;
}

```

```

        case "CPU [I9] (UnityEngine.GameObject)":
            current_cpu = cpu_intel_i9;
            break;

        case "CPU [Ryzen 5] (UnityEngine.GameObject)":
            current_cpu = cpu_ryzen_5;
            break;

        case "CPU [Ryzen 7] (UnityEngine.GameObject)":
            current_cpu = cpu_ryzen_7;
            break;

        case "CPU [Ryzen 9] (UnityEngine.GameObject)":
            current_cpu = cpu_ryzen_9;
            break;

        default:
            Debug.Log("No matches");
            break;
    }
    switch (SpawnObject.fan_name)
    {
        case "Cooling Fan [Purple] (UnityEngine.GameObject)":
            current_fan = fan_purple;
            break;

        case "Cooling Fan [Orange] (UnityEngine.GameObject)":
            current_fan = fan_orange;
            break;

        case "Cooling Fan [Blue] (UnityEngine.GameObject)":
            current_fan = fan_blue;
            break;

        case "Cooling Fan [Green] (UnityEngine.GameObject)":
            current_fan = fan_green;
            break;

        case "Cooling Fan [Red] (UnityEngine.GameObject)":
            current_fan = fan_red;
            break;

        default:
            Debug.Log("No matches");
            break;
    }
    switch (SpawnObject.gpu_name)
    {
        case "GPU [Purple] (UnityEngine.GameObject)":
            current_gpu = gpu_purple;
            break;

```

```

        case "GPU [Orange] (UnityEngine.GameObject)":
            current_gpu = gpu_orange;
            break;

        case "GPU [Blue] (UnityEngine.GameObject)":
            current_gpu = gpu_blue;
            break;

        case "GPU [Green] (UnityEngine.GameObject)":
            current_gpu = gpu_green;
            break;

        case "GPU [Red] (UnityEngine.GameObject)":
            current_gpu = gpu_red;
            break;

        case "GPU [Cyan] (UnityEngine.GameObject)":
            current_gpu = gpu_cyan;
            break;

        default:
            Debug.Log("No matches");
            break;
    }
    switch (SpawnObject.ram_name)
    {
        case "RAM-G [Purple] (UnityEngine.GameObject)":
            current_ram = ram_purple;
            break;

        case "RAM-G [Orange] (UnityEngine.GameObject)":
            current_ram = ram_orange;
            break;

        case "RAM-G [Blue] (UnityEngine.GameObject)":
            current_ram = ram_blue;
            break;

        case "RAM-G [Green] (UnityEngine.GameObject)":
            current_ram = ram_green;
            break;

        case "RAM-G [Red] (UnityEngine.GameObject)":
            current_ram = ram_red;
            break;

        default:
            Debug.Log("No matches");
            break;
    }
}

```

```

switch (SpawnObject.psu_name)
{
    case "Power Supply [Blue] (UnityEngine.GameObject)":
        current_psu = psu_blue;
        break;

    case "Power Supply [Green] (UnityEngine.GameObject)":
        current_psu = psu_green;
        break;

    case "Power Supply [Red] (UnityEngine.GameObject)":
        current_psu = psu_red;
        break;

    default:
        Debug.Log("No matches");
        break;
}

if (
    current_case[0] == current_mb[4]
    && current_case[0] == current_psu[0]
    && current_mb[0] == current_cpu[0]
    && (
        current_mb[0] == current_fan[0]
        || current_mb[0] == current_fan[1]
        || current_mb[0] == current_fan[2]
        || current_mb[0] == current_fan[3]
    )
    && (
        current_mb[2] == current_gpu[0]
        || current_mb[3] == current_gpu[0]
    )
    && current_mb[1] == current_ram[0]
) {
    result.text = correctBuildTitle;
    resultShadow.text = correctBuildTitle;

    description.text = "The PC configuration was assembled
correctly.";
    descriptionShadow.text = "The PC configuration was
assembled correctly.";

    winMenu.SetActive(true);
}

else
{
    result.text = errorTitle;
    resultShadow.text = errorTitle;
}

```


ДОДАТОК II. Лістинг реалізації програмного коду скрипта PauseMenu.cs

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

public class PauseMenu : MonoBehaviour
{
    public GameObject pauseMenu;
    public GameObject controlsMenu;
    public static bool gameIsPaused = false;
    public static bool controlsMenuActive = false;

    void Start()
    {
        pauseMenu.SetActive(false);
        controlsMenu.SetActive(false);
        gameIsPaused = false;
        controlsMenuActive = false;
    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape) && !controlsMenuActive)
        {
            if (gameIsPaused)
            {
                Resume();
            } else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseMenu.SetActive(false);
        gameIsPaused = false;
    }

    void Pause()
    {
        pauseMenu.SetActive(true);
        gameIsPaused = true;
    }

    public void ShowControls()
    {

```

```

        controlsMenu.SetActive(true);
        controlsMenuActive = true;
    }

    public void CloseControls()
    {
        controlsMenu.SetActive(false);
        controlsMenuActive = false;
    }

    public void ExitToMainMenu()
    {
        while (SpawnObject.mb_is_spawned)
        {
            DeleteObject.Delete();
        }

        Destroy(GameObject.FindGameObjectWithTag("Case"));
        SpawnObject.case_is_spawned = false;
        SpawnObject.infoActiveStatus = false;
        SceneManager.LoadScene(0);
    }
}

```

Додаток К. Результат розрахунку трудомісткості проекту в онлайн-калькуляторі COSOMO 2

У нас Organic команда і нам потрібно написати 1 тисяч рядків коду

2.40
Людина-місяць

3.49
Місяців





















0.69
персоналу

> Таблиця коефіцієнтів

4.06
Трудомісткість у людино-місяцях

| | Дуже низький | Низький | Середній | Високий | Дуже високий | Критичний |
|-------------------------|----------------------------|---------------------------------------|------------------------------------|----------------------------|----------------------------|-------------------------|
| Необхідна надійність ПЗ | <input type="radio"/> 0.75 | <input type="radio"/> 0.88 | <input checked="" type="radio"/> 1 | <input type="radio"/> 1.15 | <input type="radio"/> 1.4 | <input type="radio"/> - |
| Розмір БД програми | <input type="radio"/> - | <input checked="" type="radio"/> 0.94 | <input type="radio"/> 1 | <input type="radio"/> 1.08 | <input type="radio"/> 1.16 | <input type="radio"/> - |

| | | | | | | |
|--------------------------|------|------|---|------|------|------|
| Складність продукту | 0.7 | 0.85 | 1 | 1.15 | 1.3 | 1.65 |
| Вимоги до швидкодії | - | - | 1 | 1.11 | 1.3 | 1.66 |
| Обмеження пам'яті | - | - | 1 | 1.06 | 1.21 | 1.56 |
| Нестійкість оточення | - | 0.87 | 1 | 1.15 | 1.3 | - |
| Час відновлення | - | 0.87 | 1 | 1.07 | 1.15 | - |
| Аналітичні здібності | 1.46 | 1.19 | 1 | 0.86 | 0.71 | - |
| Здібності до розробки ПЗ | 1.29 | 1.13 | 1 | 0.91 | 0.82 | - |

| | | | | | | |
|---------------------------------------|--|--|---|--|--|---|
| Досвід розробки |  1.42 |  1.17 |  1 |  0.86 |  0.7 |  - |
| Досвід використання віртуальних машин |  1.21 |  1.1 |  1 |  0.9 |  - |  - |
| Досвід мови |  1.14 |  1.07 |  1 |  0.95 |  - |  - |
| Інструменти розробки |  1.24 |  1.1 |  1 |  0.91 |  0.82 |  - |
| Методи розробки |  1.24 |  1.1 |  1 |  0.91 |  0.83 |  - |
| Графік розробки |  1.23 |  1.08 |  1 |  1.04 |  1.1 |  - |