Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування GoF»

Виконав: студент 2-го курсу, групи 1ПІ-21м спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Васянович Є.А.
(прізвище та ініціали)

Керівник: д.т.н., професор каф. ПЗ
Ліщинська Л.Б.
(прізвище та ініціали)
«16» 12 2022 р.

Опонент: завідувач кафедри ЗІ, д.т.н., професор
Лужецький В.А.
(прізвище та ініціали)
«16» 12 2022 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«16» 12 2022 р.

Вінниця ВНТУ - 2022 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
_____ Романюк О. Н.
« 16 » вересня 2022 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Васяновичу Євгенію Анатолійовичу

1. Тема роботи – розробка методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування GoF.

Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., професор каф. ПЗ, затверджені наказом вищого навчального закладу від « 15 » вересня 2022 р. № 205-А.

2. Строк подання студентом роботи
9 грудня 2022 р.

3. Вихідні дані до роботи: кольоровий режим – TrueColor; формат зберігання тривимірних моделей – «.fbx»; формат зберігання растрових зображень – «.png»; розмір основних текстур – 512x512; тип полігональної моделі – трикутна; середовище розробки – Unity; мова розробки – C#; операційна система – Windows 10 та вище; псевдомова програмування та транслятор до неї.

4. Зміст текстової частини: вступ, аналіз методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування, моделювання інтерактивного середовища вивчення шаблонів проєктування та проєктування програмних засобів, реалізація методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування, тестування програмного додатку, економічна частина, висновки, список використаних джерел.

5. Перелік графічного матеріалу: галузь застосування методів і засобів створення інтерактивного середовища, візуалізація шаблонів проєктування; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ліщинська Л. Б., д.т.н., професор каф. ПЗ	16.09.22	15.12.22
5	Глущенко Л. Д., к.е.н., доцент кафедри ЕПВМ	30.11.22	15.12.22

7. Дата видачі завдання 16 вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування	17.09.2022-28.09.2022	Вик.
2	Моделювання інтерактивного середовища вивчення шаблонів проєктування та проєктування програмних засобів	29.09.2022-25.10.2022	Вик.
3	Програмна реалізація методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування	26.10.2022-20.11.2022	Вик.
4	Тестування програмого додатку	21.11.2022-29.11.2022	Вик.
5	Економічна частина	30.11.2022-8.12.2022	Вик.

Студент _____ **Васянович Є. А.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Ліщинська Л. Б.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.5

Васянович Є. А. Розробка методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування GoF. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2022. 127 с.

На укр. мові. Бібліогр.: 49 назв; рис.: 32; табл.: 5.

У магістерській кваліфікаційній роботі розроблено інтерактивне середовище для вивчення шаблонів проєктування. Також проаналізовано методи вивчення складної комплексної інформації, на основі чого створено програмний застосунок. В результаті додаток складається з консольного інтерфейсу для програмування поставленої задачі та візуалізованої частини для наочності.

Розроблено спеціальну псевдомову для навчання для алгоритм транслятор для виконання програмного коду. Знайдено спосіб зберігання текстур графічних моделей із заощадженням пам’яті.

В економічній частині описано доцільність розробки, план витрат на реалізацію науково-дослідної роботи, можливі доходи від її впровадження. Досліджено ефективність вкладення інвестицій у проєкт і можливість залучення інвесторів.

Ключові слова: шаблони проєктування, інтерактивне середовище навчання, псевдомова, транслятори програмного коду.

ABSTRACT

Vasyanovych Y. A. Development of methods and software tools for creating an interactive environment for studying GoF design patterns. Master's thesis on the specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2022. 127 p.

In Ukrainian language Bibliographer: 49 titles; Fig.: 32; tab.: 5.

In the master's qualification work, an interactive environment for studying design patterns was developed. Methods of studying complex complex information were also analyzed, on the basis of which a software application was created. As a result, the application consists of a console interface for programming the given task and a visualized part for clarity.

A special pseudo-language was developed for learning the translator algorithm for executing the program code. Found a memory-saving way to store graphics model textures.

The economic part describes the feasibility of the development, the cost plan for the implementation of scientific research work, possible income from its implementation. The effectiveness of investing in the project and the possibility of attracting investors have been studied.

Keywords: design paterns, interactive learning environment, pseudo-language, software code translators.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Щороку комп'ютерні технології ростуть, поширюються їх вплив на життя людей, а разом з тим розвиваються і методи їх створення. Сучасне програмне забезпечення створюється зазвичай не однією людиною, а цілими командами. Час розробки може сягати місяці, а потім ще іде етап його підтримки та покращення. А тому потрібні технології, що допоможуть ефективніше створювати нові додатки та поліпшити процес розробки.

Кожне програмне забезпечення чи будь-який додаток, що має вагомий розмір та функціонал, починається із планування архітектури. Архітектура – це основа, що задає темпи розробки та подальшого розвитку ПЗ. Від неї залежить ціна на підтримку і розробку нової функціональності, трудовитрати на побудову цілої системи з використанням даної архітектури. Тобто формально від архітектури залежить найважливіший параметр розробки - собівартість. А побічно ще і можливість повторного використання коду [1]. Існує багато рішень при розробці архітектури: це правила, принципи, певні структури. Але з розвитком індустрії розробки програмного забезпечення було накопичено набір типових способів вирішення задач проєктування програмного забезпечення, які враховують різні архітектурні та функціональні особливості систем. Ці рішення були названі шаблонами проєктування [2]. Для сучасного програміста знання цих шаблонів є необхідністю, адже вони мають багато рішень різноманітних задач програмування. Проте, щоб ефективно їх застосовувати розробник повинен не просто їх знати, а вміти застосувати у потрібний момент.

Основні задачі, що вирішують патерни проєктування:

- Перевірені рішення. Розробник витрачає менше часу, використовуючи готові рішення.

- Стандартизація коду. Розробник менше помиляється при проєктуванні, використовуючи типові уніфіковані рішення, оскільки всі приховані в них проблеми вже давно знайдено.
- Загальний словник програмістів. Вимовляючи назву патерна програміст зрозуміє, про що йде мова і зможе легко працювати з кодом [3].

Шаблон проєктування в розробці програмного забезпечення – повторювана архітектурна конструкція, що представляє собою рішення проблеми проєктування в рамках деякого часто виникаючого контексту [4]. Це рішення не надає готового програмного коду, а лише описує абстрактний алгоритм рішення певної задачі. Тому саме питання їх вивчення змушує шукати нові та ефективні підходи.

Найпоширенішими шаблонами проєктування є GoF (Gang of Four) або ж «банда чотирьох». Вони класифікуються за призначенням на наступні:

- породжувальні;
- структурні;
- поведінкові.

Для вивчення шаблонів вже існуючі методи надають лише звичайну текстову інформацію, опис коду чи алгоритму, а також діаграму класів. В найкращому випадку буде наведено приклад з аналогією з життя. Такі способи подачі складної інформації є малоефективними. Тому потрібно вдосконалити методи щодо вивчення шаблонів проєктування. Один із варіантів – візуалізація на основі тривимірної графіки.

Тривимірна графіка складається з моделі та текстури. Текстури представляють собою двовимірну картинку, що згодом накладається на тривимірну модель. Таким чином для кожної моделі потрібно мати свою текстуру, а також і додаткові, якщо вони присутні. Це збільшу загальний обсяг додатку, що їх використовує та може навіть знижувати продуктивність. Таким чином, є актуальним питання оптимізації зберігання текстур.

Створення інтерактивного середовища має на меті поєднати елементи практики, візуалізації та звичайні методи вивчення шаблонів проєктування. Єдиний спосіб практикування патернів – це їх програмування. Тому постає питання створення мови програмування та транслятора до неї. Сучасні методи трансляції є досить ефективні в контексті розробки програмного забезпечення. Проте, для навчальних цілей вони можуть бути надлишковими. Компілятори формують вихідний код у окремий файл, що може бути виконаний повторно, в той час як інтерпретатори зчитують програмний код кожного разу при виконанні. Тому, актуальним є розробка методу трансляції та програмних засобів на його основі, що дозволить підвищити ефективність перетворення коду за рахунок подолання зазначених недоліків.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення ефективності вивчення шаблонів проєктування за рахунок комбінації методів сприйняття інформації, а саме використання методів візуалізації інформації у поєднанні з виконанням задач на програмування.

Основними задачами дослідження є:

- провести аналіз методів вивчення складної та комплексної інформації з метою покращення ефективності вивчення шаблонів проєктування;
- проаналізувати основні аналоги програмних засобів вивчення шаблонів проєктування;
- удосконалити метод вивчення шаблонів проєктування;
- вдосконалити методи створення трансляторів для мов програмування;
- запропонувати метод текстурування з метою підвищення ефективності зберігання графічної інформації;
- розробити псевдомову програмування для виконання практичних задач;
- розробити інтерактивне середовище з візуалізацією для вивчення шаблонів;

- провести тестування розробленого додатку.

Об’єкт дослідження – процеси побудови інтерактивного навчального середовища для вивчення шаблонів проєктування.

Предмет дослідження – методи і засоби створення інтерактивного навчального середовища для вивчення шаблонів проєктування.

Методи дослідження. У процесі дослідження використовувалися: методи сприйняття інформації для засвоєння навчального матеріалу, методи текстурування при розробці тривимірних моделей, алгоритми трансляції мов програмування для виконання програмного коду, комп’ютерне моделювання для аналізу та перевірки отриманих результатів.

Наукова новизна отриманих результатів.

1. Удосконалено метод вивчення шаблонів проєктування на основі інтерактивного середовища, що дозволить підвищити ефективність їх вивчення, за рахунок використання засобів візуалізації, а також інтегрованого середовища розробки для практичного тренування навичок створення шаблонів.

2. Запропоновано комбінований метод трансляції мови програмування, що використовує переваги компілятора та інтерпретатора і дозволяє уникнути зайвих трансляції програмного коду, при цьому не зберігаючи виконуваний код в окремий файл.

3. Запропоновано метод зберігання текстур для тривимірних низько-полігональних моделей на основі однієї основної текстури та декількох додаткових мап, що дозволяє заощадити використання пам’яті додатком на 25%.

Практична цінність отриманих результатів полягає у тому, що на основі отриманих у магістерській кваліфікаційній роботі теоретичних положень запропоновано програмні засоби для вивчення шаблонів проєктування з використанням візуалізації інформації.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. Автору належать такі результати: інтерактивне середовище вивчення шаблонів проєктування [6], аналіз методів створення мов програмування, комбінований

метод трансляції мови програмування, метод текстурування на основі однієї текстури.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на конференціях: «L Науково-технічна конференція підрозділів Вінницького національного технічного університету» (Вінниця, 2021), міжнародна науково-практична інтернет-конференція «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 2022).

Публікації. Основні результати досліджень опубліковані в наукових конференціях – в тезах доповіді на конференції «L Науково-технічна конференція підрозділів Вінницького національного технічного університету» (Вінниця, 2021) [5] та конференції «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 2022) [6].

1 АНАЛІЗ МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНОГО СЕРЕДОВИЩА ВИВЧЕННЯ ШАБЛОНІВ ПРОЄКТУВАННЯ

1.1 Аналіз методів сприйняття складної інформації

Ми живемо в світі, де інформація оточує людину звідусіль. Вона зустрічається в різних проявах, видах та буває зовсім різною. Сучасна людина багато часу проводить в інтернеті, де вона найчастіше зустрічається з текстовою інформацією. Дивлячись телевізор ми сприймаємо візуальні дані, а слухаючи наприклад музику – відповідно звукові. Проте серед особливостей сприйняття різних видів інформації є одна суттєва деталь – 90 відсотків інформації сприймається через зір (тобто текстова і візуальна). До того ж науковці встановили, що якщо інструкція до ліків містить лише текст, людина засвоїть лише 70 відсотків даних, а з малюнками – до 95. Також візуальна інформація сприймається швидше у шістдесят тисяч разів в порівнянні з текстовою. Тому візуалізовані дані є фаворитом у сприйнятті людиною.

Візуалізація – це наочне представлення інформації у вигляді графіків, рисунків тощо. На рисунку 1.1 наведено приклад представлення однієї і тієї ж інформації у двох варіантах – текстовому (таблиця) і візуалізованому (графік).

A	B	C	D	E	F	G	H
122,457	135,266	123,856	144,586	152,335	111,354	165,235	143,253
I	J	K	L	M	N	O	P
168,253	145,236	152,324	164,253	132,254	143,879	142,356	122,386

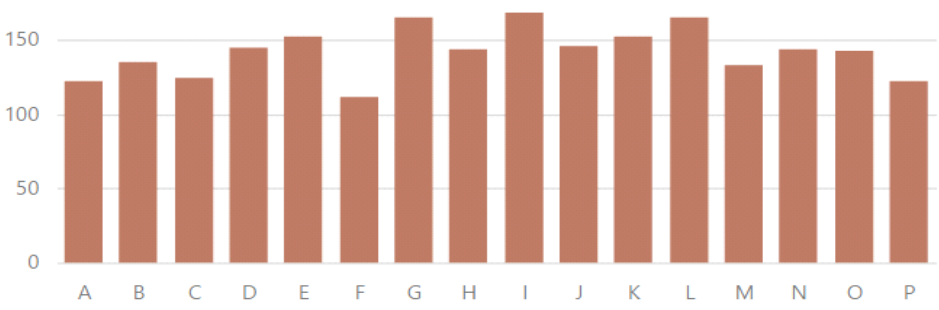


Рисунок 1.1 – Приклад представлення текстової (верхній рисунок) та графічної (нижній рисунок)

Візуалізація даних має ряд переваг:

- акцентування уваги на різних аспектах даних;
- аналіз великих наборів даних;
- зменшення інформаційного перевантаження і фокусування уваги;
- виділення взаємозв’язків і відносин з інформації [7].

Шаблони проєктування ПЗ – це способи вирішення задач проєктування. Це є зразок вирішення проблеми чи задачі і відображає відношення між класами та об’єктами [8]. Шаблони представляють досить складну для розуміння інформацію, що має певні зв’язки та структуру. Тому для їх вивчення краще використовувати певні схеми та діаграми. Але сам по собі шаблон є досить абстрактною інформацією і людині важко її сприймати. Саме тому в таких випадках потрібно використання прикладів та аналогій. Але найкраще сприймається візуальна інформація. Для цього є доцільним використання візуалізації певних прикладів на основі аналогій з життя для розуміння шаблону проєктування. На рисунку 1.2 зображено приклад такої візуалізації. Певні елементи інформації переносяться на життєві аналоги, наприклад сутності проєктуються на людей або предмети, а методи – на їх дії.

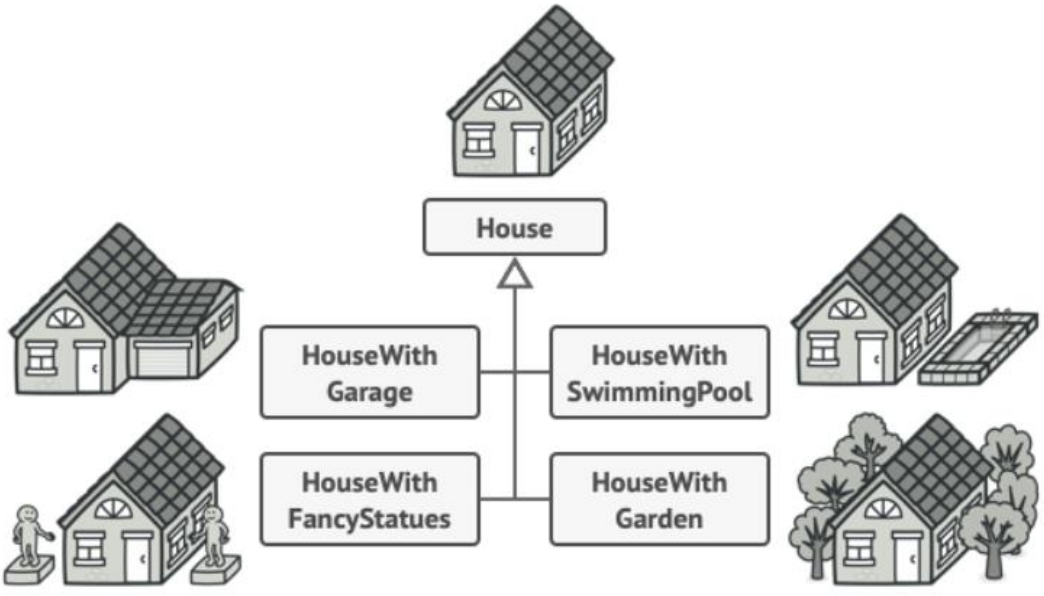


Рисунок 1.2 – Приклад візуалізації – замість класів будинків (аналогія з життям) подають їх рисунки

Іншим аспектом засвоєння інформації є спосіб її вивчення. До цієї категорії належать слухове сприйняття або аудіювання, читання інформації, запис інформації, практика, а також навчання інших [9]. Розглянемо кожен метод навчання окремо та порівняємо їх за критеріями ефективності. Перший критерій – «складність сприйняття», визначає наскільки складно працювати з таким видом інформації. 1 бал – надто складна, 5 балів – легка. «Швидкість роботи з інформацією» – критерій, що визначає час, який необхідно провести з даним видом інформації, щоб його засвоїти. 1 бал – декілька днів, 5 балів – декілька годин. Критерій «незалежність від інших методів» визначає можливість застосування методу окремо від інших. 1 бал – метод є повністю залежним, 3 бали – метод незалежний. Останній критерій визначає відсоток засвоєння інформації. В таблиці 1.1 наведено найпоширеніші способи сприйняття і вивчення інформації та їх ефективність.

Таблиця 1.1 – Методи вивчення інформації та їх ефективність

Метод	Критерії якості				Сума
	Складність сприйняття	Швидкість роботи з інформацією	Незалежність від інших методів	Відсоток засвоєння	(A + B + C) * %
Лекція	3/5	2/5	2/3	5%	0,35
Читання	2/5	1/5	3/3	10%	0,6
Відео/аудіо	4/5	3/5	2/3	20%	1,8
Дискусія	3/5	4/5	2/3	50%	4,5
Практика через дію	4/5	3/5	2/3	75%	6,75
Навчання інших, миттєве застосування знань	4/5	3/5	3/3	90%	9

Порахувавши суму балів та помноживши на відсоток засвоєння інформації визначено відносний коефіцієнт ефективності методу [10]. Відповідно,

зважаючи на результати таблиці можна зробити висновок, що відсоток засвоєння інформації та коефіцієнт ефективності мають схожі відносні показники. Таким чином, найефективнішими з даних методів є навчання інших та навчання через практику, а найгіршими – читання тексту та його слухання.

Отже, серед методів засвоєння інформації найкращими є ті, що використовують практику, посередині розташовують методи візуалізації, та найнижчою ланкою є текстові. Проте, варто зауважити, що методи мають інші важливі характеристики, наприклад незалежність від інших способів чи час вивчення інформації. Таким чином, візуалізація не може розповісти усі деталі, а читання є найповільнішим методом. Тому, для підвищення ефективності є сенс комбінування цих методів.

1.2 Порівняльний аналіз аналогів програмних засобів вивчення шаблонів проєктування

В мережі існує багато інформації щодо шаблонів проєктування, а також ресурсів для їх вивчення. З іншого боку, програмних додатків значно менше. Розглянемо найпопулярніші сайти та додатки і проведемо ґрунтовний їх аналіз. Таким чином визначимо, чи доцільною є розробка власного програмного забезпечення. Отже, проаналізуємо наступні аналоги ресурсів для вивчення шаблонів проєктування:

- веб-ресурс “Refactoring guru”;
- веб-ресурс “Metanit”;
- додаток “GoF Design Patterns”;
- додаток “Design Patterns in C#”.

Refactoring guru (рис. 1.3) – мережевий ресурс, що надає можливість вивчення не тільки патернів проєктування, а інших принципів написання чистого коду. До переваг можна віднести: ґрунтовний аналіз шаблонів, діаграми класів та інші рисунки, наявність прикладів з аналогіями, поради щодо написання,

програмний код на псевдомові та інших мовах програмування, локалізацію вмісту веб-ресурсу на десяти мовах спілкування. Також до переваг можна віднести приємний дизайн та зручну навігацію. Проте недоліком ресурсу є те, що він потребує підключення до мережі [11].

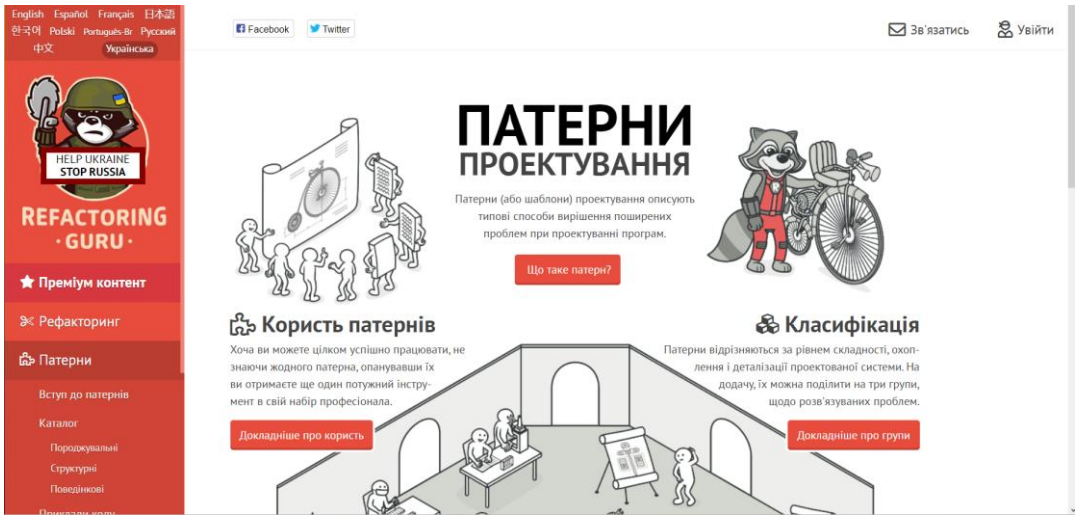


Рисунок 1.3 – Веб-ресурс Refactoring guru

Metanit (рис. 1.4) – також інтернет-ресурс, який надає загальну інформацію про усі шаблони проєктування GoF. Перевагами цього сервісу є повний опис патернів мовою програмування C#, а також наявність діаграм класів та приклади реалізації на основі аналогії. Недоліками є необхідність підключення до мережі, реалізація лише однією мовою, а також локалізація ресурсу однією мовою спілкування [12].

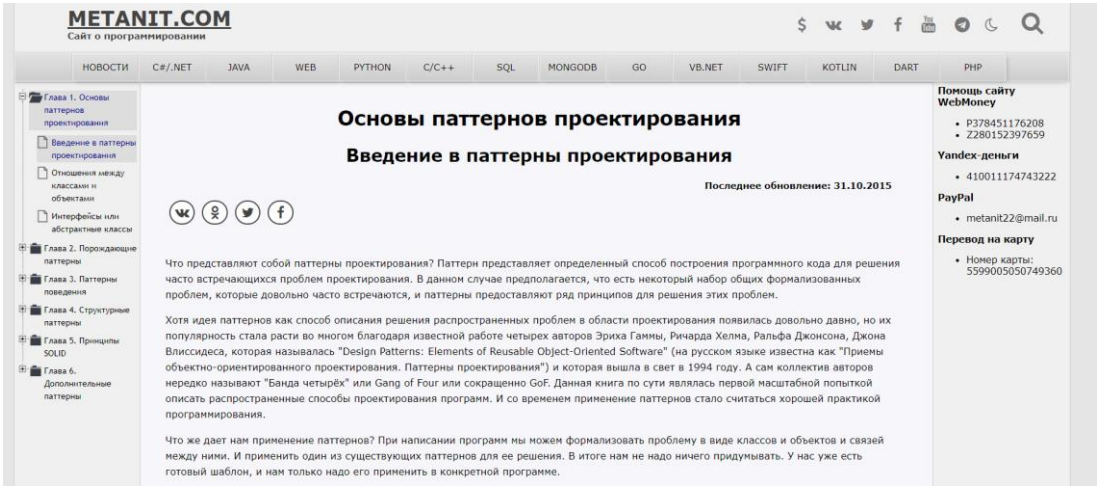


Рисунок 1.4 – Веб-ресурс Metanit

GoF Design Patterns (рис. 1.5) – програмне забезпечення для вивчення шаблонів проєктування на мові Java. Додаток має наступні переваги: опис шаблонів проєктування без доступу до мережі. Недоліками є неповний безкоштовний контент, підтримка однієї мови як програмування так і спілкування [13].

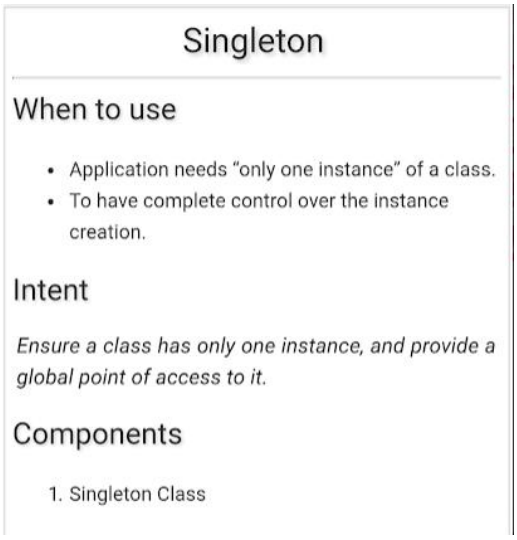


Рисунок 1.5 – Інтерфейс додатку GoF Design Patterns

Design Patterns in C# (рис. 1.6) – додаток для вивчення шаблонів проєктування на мові C#. Не потребує підключення до мережі та пропонує опис шаблонів на мові програмування C#. Має безкоштовний доступ до всього контенту. Серед недоліків присутні наступні: одна мова програмування, відсутність візуалізації, скорочений опис шаблонів проєктування [14].

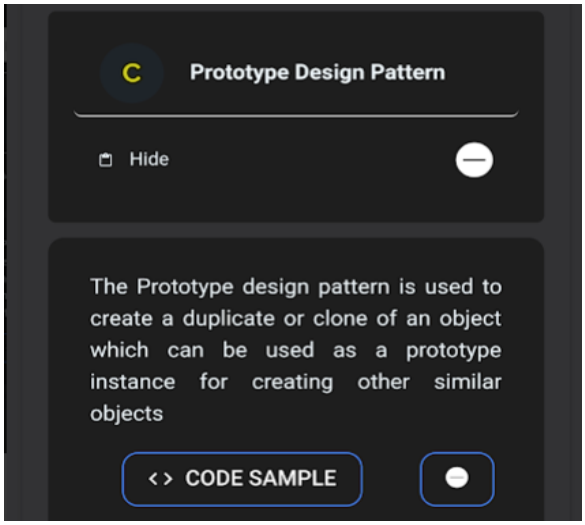


Рисунок 1.6 – Інтерфейс додатку Design Patterns in C#

Оцінимо розглянути додатки за п’ятьма категоріями за бальною шкалою. Кожна категорія має свою максимальну кількість балів, таким чином визначаючи важливість. Категорія «графічна складова» визначає наявність у ресурсу графічних даних, рисунків, схем, таблиць, візуалізації та має максимум 5 балів. Категорія «текстова складова» визначає наскільки якісно та детально надається опис шаблонів проєктування, а також використання в описі аналогій, максимум 3 бали. Категорія «безкоштовна ліцензія» визначає ресурс як безкоштовний, частково безкоштовний та платний. Має максимум 3 бали, що означає безкоштовний. Зведемо результати дослідження аналогів до таблиці 1.2. Категорія «підключення до мережі» визначає чи потрібно ресурсу використання мережі, два бали якщо ні і один якщо так. Категорія «підтримка псевдомови» показує чи навчання відбувається на псевдомові програмування, а також інших мовах. Три бали, якщо підтримує псевдомову та інші мови, два бали якщо тільки псевдомову та один за використання будь-якої мови програмування.

Таблиця 1.2 – Порівняльні характеристики ресурсів для вивчення шаблонів проєктування

Критерій	Refactoring guru	Metanit	GoF Design Patterns	Design Patterns in C#	Власний додаток
Графічна складова	4/5	2/5	1/5	1/5	5/5
Текстова складова	3/3	2/3	2/3	1/3	2/3
Безкоштовна ліцензія	3/3	3/3	2/3	2/3	3/3
Підключення до мережі	1/2	1/2	2/2	2/2	2/2
Підтримка псевдомови	3/3	1/3	1/3	1/3	3/3
Загальна оцінка ефективності	14/16	9/16	8/16	7/16	15/16

Аналізуючи дані таблиці 1.1 можна зробити висновок, що реалізація власного додатку є доцільною. Єдиним аналогом, що має схожий рівень

ефективності навчання є Refactoring guru. Проте в даній таблиці не розглядався іще один важливий критерій, а саме інтерактивність, тобто навчання практикою, адже жоден ресурс не надає такої можливості. Саме тому розробка власної реалізації є актуальною, щоб поєднати вже вирішені завдання та покрити недоліки аналогів.

1.3 Аналіз методів створення трансляторів для мов програмування

Транслятор — це спеціальна програма, яку використовують для перекладу програм користувача, написаних мовою програмування високого рівня, у так звані машинні коди, зрозумілі процесору. Транслятори бувають різних видів:

- компілятори;
- декомпілятори;
- асемблери;
- дизасемблери;
- інтерпретатори;
- препроцесори.

Розглянемо деякі з цих засобів більш детально [15].

Компілятор перетворює вихідний код програми у певний набір інструкцій та зберігає їх у файлі. При цьому він не виконує код. Замість цього виконується збережений файл з готовими командами по бажанню користувача необхідну кількість разів. Компілятори використовують для багаторазового виконання коду [16].

Інтерпретатор транслює та виконує кожну команду вихідного коду крок за кроком. Тому при кожному використанні програми відбувається процес трансляції коду, що є не ефективним при багаторазовому використанні. Проте вони мають перевагу при разовому виконанні коду на компіляторами [17].

Препроцесори – це програми, що перетворюють певним чином вихідний код програми на інший код для подальшого використання. Наприклад, це може

бути текстова підстановка. Після обробки препроцесором програмний код може бути використаний іншим транслятором [18].

Кожен транслятор має певні загальні етапи виконання На рисунку 1.7 наведено процес трансляції по етапах. Загальна структура є наступною:

1. лексичний аналіз;
2. синтаксичний аналіз;
3. видозалежний аналіз;
4. оптимізація;
5. генерація коду.

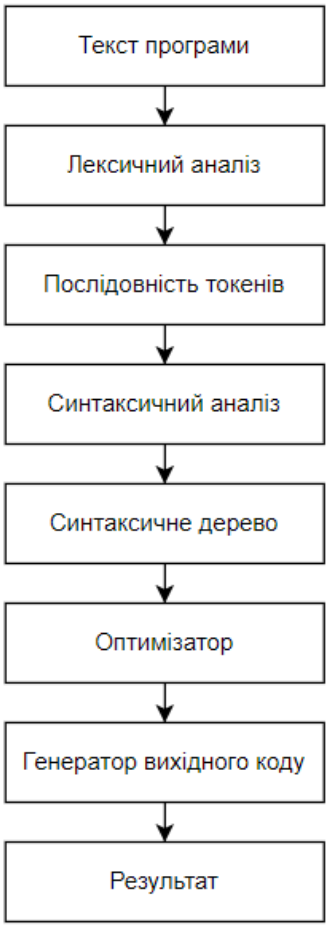


Рисунок 1.7 – Процес трансляції

Лексичний аналіз розпочинає первинну обробку коду. Він приймає на вхід рядок і розбиває його на лексеми або токени. Кожен токен має певний тип (число, оператор, змінна тощо). Також на цьому етапі видаляються усі непотрібні або

незначущі для компілятора речі (коментарі, пробіли та ін.). Лексичний аналізатор є скінченим автоматом. Аналіз коду відбувається за один прохід. Зазвичай лексичний аналізатор складається з двох частин: сканування та обчислення. Спочатку він сканує та виділяє лексеми в коді, а потім перетворює їх в окремі значення [19].

Наступною фазою компіляції є синтаксичний аналізатор. Він на вхід отримує результат лексичного аналізатора і розбирає граматичну структуру згідно із заданою граматикою. Під час процесу аналізу текст програми перетворюється в структуру дерева, яке згодом легко піддається обробці. Існує два методи формування структури: низхідний та висхідний [20].

Видозалежний аналіз – полягає перевірці правильності типів даних, що використовуються в програмі. Також на даному етапі перевіряється дотримання контекстних умов мови програмування.

Оптимізація – необов’язкова фаза процесу трансляції. Вона потрібна для підвищення ефективності по різних показниках таких як швидкість виконання або ж обсяг пам’яті. До найпоширеніших методів оптимізації відносять: константні обчислення, зменшення сили операцій, виділення загальних виразів, чищення циклів тощо. Останньою фазою трансляції є генерація коду. Під час цієї фази генерується кінцевий код програми. Також на даному етапі вирішується ряд інших завдань: розподіл пам’яті, розподіл регістрів та ін.

Таким чином, існує два основні типи трансляторів – компілятори та інтерпретатори, що використовують схожий процес перетворення коду. Цей процес може містити не усі пункти, згадані вище, а лише потрібні для певного завдання. Різниця двох методів трансляції полягає в особливостях їх застосування, а саме компілятори використовуються при багаторазовому виконанні одного і того ж коду, а інтерпретатори – одноразовому. Але і ці методи не завжди підходять під конкретну задачу і потрібен пошук більш гнучкого рішення.

1.4 Особливості програмних засобів середовища програмування

Інтегроване середовище розробки – комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду [21].

До основних компонентів середовища розробки програмного коду можна віднести:

- текстовий редактор для набору програмного коду;
- мова програмування і компілятор до неї;
- компонувальник;
- система усунення синтаксичних помилок;
- бібліотека програмних модулів.

Кожен з цих інструментів стає у нагоді при розробці програмного коду. Спочатку користувач вводить певний текст з клавіатури і він виводиться у вікно текстового редактора. Програма виправлення помилок постійно перевіряє код на їх наявність. Готова програма без помилок може бути запущена на виконання за допомогою транслятора та компонувальника [22].

На противагу інтегрованому середовищу протиставляють так звані «легкі редактори» – вони не такі потужні, проте прості, доступні, а найголовніше – мають високу швидкодію. Їх зазвичай використовують для редагування декількох файлів коду. Але і вони не настільки прості, як здаються, адже є можливість розширення за допомогою плагінів [23].

В будь якому разі інтегроване середовище розробки є важливою частиною процесу розробки коду. Воно забезпечує програміста усіма важливими інструментами, необхідними для комфортного та ефективного програмування. Натомість інтерактивне середовище для вивчення шаблонів проєктування включає процес написання коду, а отже потрібно розробити і інтегроване

середовище. Серед основних модулів потрібно розробити текстовий редактор та мову програмування з компілятором. Додатковим модулем може виступати систему виявлення помилок. Така конфігурація разом з вікном виводу інформації буде називатися консоллю розробника та забезпечить зручний процес програмування для вирішення поставлених задач.

Отже, інтегроване середовище зазвичай складається з таких основних модулів: текстовий редактор коду, мова програмування і компілятор, систему виявлення помилок, а також бібліотека модулів і компонувальник. Проте цей список не є однозначним списком і може конфігуруватися в залежності від задачі. Так, для інтерактивного середовища потрібне використання редактору коду разом з мовою програмування, а також підсвічення коду, система виявлення помилок.

1.5 Висновки

Вивчення шаблонів проєктування потребує особливого підходу, адже такого роду інформацію можна визначити як складну для сприйняття. Тому в результаті аналізу з’ясувалося, що найкращим підходом до вивчення патернів є методи візуалізації та тренування (практика). Детальний аналіз аналогів дав зрозуміти, що існуючі методи вивчення не надають інтерактивної частини вивчення шаблонів, що зменшує ефективність цього процесу. Тому прийнято рішення, що розробка інтерактивного середовища зможе подолати ці недоліки.

Найважливішою частиною інтерактивного середовища є програмування та реалізація шаблонів. Найефективнішою мовою програмування для цього буде псевдо мова, адже вона не прив’язана до конкретного синтаксису. Таким чином користувач зосереджується на вивченні структури шаблону. Зважаючи на це потрібно розробити спеціальну мову програмування та транслятор до неї. Тому в цьому розділі було розглянуто методи створення транслятора для мов програмування, а також розглянуто особливості інтегрованих середовищ розробки коду для створення такого.

Проаналізувавши стан галузі шаблонів проєктування та провівши аналіз існуючих аналогів було визначено задачі, які необхідно виконати для розробки інтерактивного середовища:

- визначити найефективніший підхід до вивчення шаблонів проєктування;
- розробити песвдомову програмування та транслятор до неї;
- розробити методи текстурування тривимірних моделей для візуалізації;
- створити консоль розробника для користувача та її складові (редактор коду, підсвітка коду, консольний вивід, вибір класів для редагування);
- розробити інтерактивне середовище вивчення шаблонів проєктування;
- провести тестування програмного додатку.

Сформулюємо нефункціональні вимоги до ПЗ. Будь-який сучасний додаток має відповідати ряду важливих вимог, таких як: надійність, гнучкість, стійкість до критичних помилок, швидкість роботи, ергономічність інтерфейсу. Таким чином, під надійністю розуміють захист особистих даних користувачів та важливої інформації. Гнучкість відповідає в першу чергу за розширення додатку. Щодо інтерфейсу, він має бути зручним у увикористанні та економити час користувача на виконання певних операцій. Отже, програмний додаток повинен дотримуватися усіх цих вимог.

Системні та апаратні вимоги доадтку для вивчення шаблонів проєктування:

- операційна система Windows 10 або вище;
- оперативна пам’ять – мінімум 4 ГБ;
- об’єм дискового простору – 500 МБ.

Отже, при розробці методів та додатку інтерактивного середовища вивчення шаблонів проєктування потрібно дотримуватися поданих вище вимог. В результаті створений продукт матиме високу якість і найдійну основу для подальшого розвитку.

2 МОДЕЛЮВАННЯ ІНТЕРАКТИВНОГО СЕРЕДОВИЩА ВИВЧЕННЯ ШАБЛОНІВ ПРОЄКТУВАННЯ ТА ПРОЄКТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

2.1 Розробка методу вивчення шаблонів проєктування на основі інтерактивного середовища

Шаблони проєктування складаються в першу чергу з певної структури та містять програмний код. Таким чином можна визначити, що вони містять як мінімум тестову інформацію. Проте архітектура будь-якої сутності легше подавати у вигляді графічних даних, схем, таблиць. Як вже зазначалося, окремо методи навчання не мають такої ефективності, ніж коли вони об’єднані, тобто комбінуються. Таким чином потрібно визначити підхід, що має найвищу ефективність при вивчення шаблонів проєктування.

Суть інтерактивного навчання у тому, що навчальний процес відбувається за умови постійної, активної взаємодії всіх учнів. Організація інтерактивного навчання передбачає моделювання життєвих ситуацій, використання рольових ігор, вирішення проблеми на основі аналізу обставин та відповідної ситуації [24].

Інтерактивне навчання передбачає:

- моделювання життєвих ситуацій;
- вирішення творчих завдань;
- спільне розв'язання проблем [25].

Основна складова, на якій ґрунтуватиметься додаток для вивчення патернів, це інтерактивне середовище. Воно складається з мови програмування, транслятора та візуального представлення даних. Отже, перший метод, що буде використано в додатку, це практика. Користувач має змогу програмувати шаблони, маючи конкретне поставлене завдання. Він матиме доступ до певних інструментів розробника, таких як консоль та псевдо-мова програмування. Це є

першим кроком на шляху до інтерактивності, адже відбувається взаємодія користувача з системою та іде зворотній зв’язок.

Наступним методом, що застосовується в цьому додатку є візуалізація. Абстрактні образи шаблонів перетворюються у конкретні життєві ситуації та моделюються за допомогою тривимірної графіки. Таким чином, усе, що програмує користувач, матиме відображення у на певній ігровій сцені, що значно підвищує інтерактивність середовища та заодно наочність шаблонів проєктування. Назвемо цей підхід динамічною візуалізацією.

Іншим аспектом візуалізації є використання звичайних схем та діаграм, що дають пояснення принципів роботи патернів. Хоча вже є представлення у вигляді аналогії, проте зв’язок з абстрактним поясненням користувач також повинен мати, так як воно є базовим і необхідним для розуміння шаблонів. В даному випадку використовується статична візуалізація [26].

Відповідно, без додаткових пояснень буде важко щось зрозуміти, тому важливим є застосування текстових методів навчання. Вони використовуватимуться всюди: пояснення шаблонів проєктування, завдання та підказки, довідкова інформація. Використані методи навчання, що поєднуються в єдине ціле, зображено на рисунку 2.1.

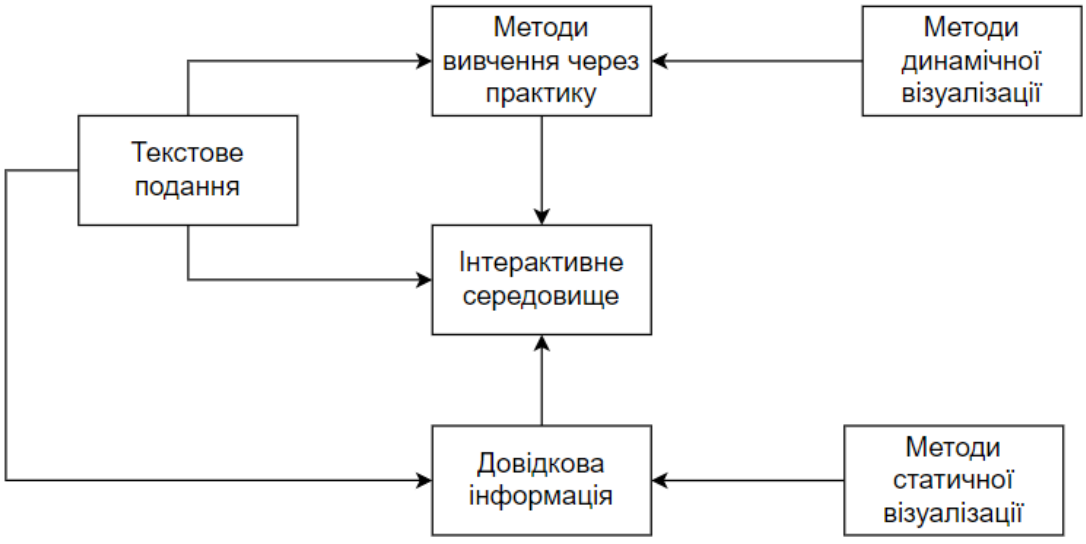


Рисунок 2.1 – Схема взаємодії методів навчання та модулів інтерактивного середовища

Такий підхід дозволить поєднати переваги використаних методів та покрити їх недоліки, що значно збільшить ефективність вивчення шаблонів проєктування за рахунок кращого засвоєння інформації (методи практики) та зростання швидкості навчання (методи візуалізації).

2.2 Розробка алгоритму для трансляції псевдокоду програмування

Розроблена псевдомова програмування не має певного конкретного шаблону, за яким би вона створювалася. А тому створений транслятор повинен підходити конкретно для цієї мови. Визначимо загальну структуру створюваного алгоритму. Якщо створити транслятор на базі інтерпретатора, то потрібно буде постійно виконувати читання та виконання коду. З іншого боку компілятор буде читати код один раз, а потім лише виконувати його. Але на процес компіляції буде витрачатися зайвий час, до того ж потрібно зберігати код програми у файлі. Тому найкращим варіантом є комбінування цих методів. Суть алгоритму наступна: спочатку код буде певним чином оброблятися, далі з кожного рядка створюватиметься певний набір команд. Під час виконання виконуватимуться ці команди, що зберігається в оперативній пам’яті. При внесенні змін потрібно буде створювати нові команди. На рисунку 2.2 зображено загальні етапи такого транслятора.

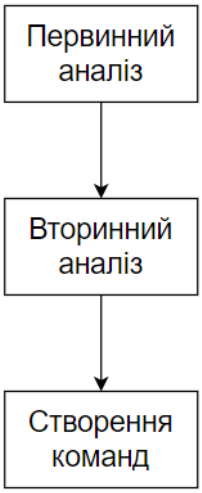


Рисунок 2.2 – Основні етапи трансляції

Перший етап аналізу – це первинна обробка коду. Програмний код зчитається з редактора коду при натисканні на відповідну кнопку. Далі аналізатор проходиться по тексту, прибирає усі непотрібні символи, пробіли, кольорові теги та проводить форматування. Це підготує код до вигляду, зручного для наступних етапів аналізу.

Наступний етап – вторинна обробка. Тут відбувається лексичний аналіз коду. В даному випадку усі слова коду розбиваються на лексеми пробілами. Математичні вирази типу «(x+ 12 *or > 1 or(1 > abs(2)))» приводяться до наступного вигляду: «(x + 12 * or > 1 or (1 > abs (2 + 3)))». Таким чином код стає повністю підготовленим до фінального етапу.

Останній етап – фаза формування команд. Кожен рядок програми аналізується на наявність ключових слів чи символів. В залежності від цього створюються команди. Готові команди містять інформацію, потрібну безпосередньо для виконання коду. На рисунку 2.3 зображено ієрархію класів, що відповідають за створення команд.

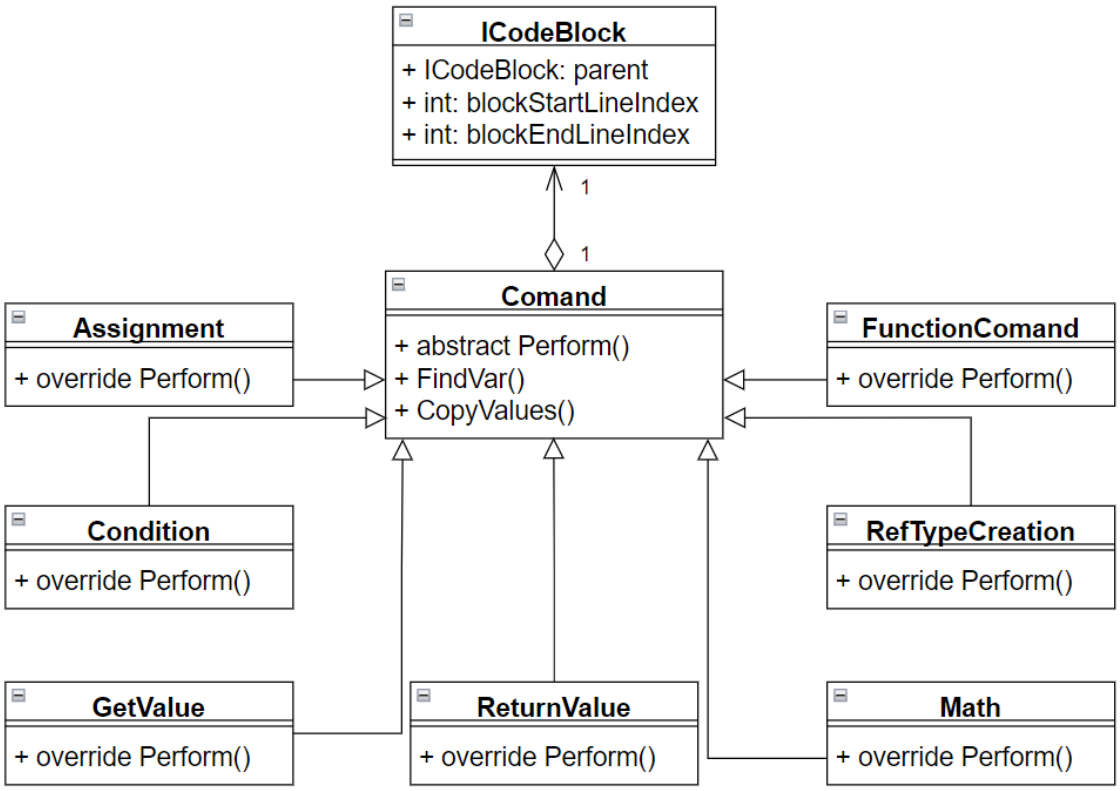


Рисунок 2.3 – Діаграма класів створення команд

Базовим класом команд є абстрактний клас «Comand». Він містить посилання на об’єкт типу «ICodeBlock», тобто на клас, що містить інформацію до якого блоку коду належить команда. Також клас команди реалізовує три функції. Методи «FindVar» та «CopyValues» є допоміжними та використовуються для пошуку певних змінних, створених користувачем, та копіювання значень відповідно. Метод «Perform» є абстрактним, тому він реалізується у всіх класах наслідниках. Його виклик відбувається під час виконання коду.

Клас «Assignment» відповідає за привласнення змінним значень. Його виклик відбувається як при звичайному використанні оператора привласнення так і при передачі змінних у метод. Він використовує методи пошуку та копіювання змінних базового класу, щоб здійснити операцію привласнення.

Клас «GetValue» використовується для створення допоміжної команди. Його роль – отримувати різні значення та повертати їх для подальшого виконання команд. Це стосується змінних і літералів.

Клас «FunctionComand» відповідальний за виконання користувацьких методів. При виконанні цієї команди важливим етапом є встановлення відповідного контексту. Адже, виклик функції відбувається відповідно до певного екземпляру класу, тому потрібно мати доступ до всіх змінних цього екземпляру. Також дана команда відповідає за передачу аргументів у функцію.

Клас «ReturnValue» відповідає за команду повернення значення з функції.

Клас «RefTypeCreation» описує команду створення змінної посилального типу. Основна задача цієї команди сформувати усі відповідні внутрішні змінні цього типу, викликати конструктор класу, якщо такий наявний. Також потрібно прослідкувати за наявністю механізму наслідування та створити відповідні змінні.

Клас «Condition» відповідає за обрахунки істинності умовних операторів та блоків розгалуження коду. Він повертає результат обчислень під час виконання.

Клас «Math» відповідає за математичні операції. Він містить два поля з операндами та оператором. На базі цих значень повертається результат. Довгі математичні вирази формуються у ланцюжок. Потім виконуючи одну математичну команду виконується наступна.

Алгоритм формування ланцюжка команд зображено на рисунку 2.4. Даний алгоритм використовує два списки команд: кінцевий та проміжний. Проміжний список формується на основі математичного виразу. До нього додаються усі значення, оператори та операнди у вигляді команди «GetValue» або вже готових «Math» команд. Далі у препроцесі «HandleMath» створюється одна математична команда, яка заміняє відповідні оператори та операнди, що зменшує проміжний список команд. Таким чином як тільки залишиться одна команда ланцюжок математичних команд буде сформовано. Потім ця команда додається до кінцевого списку команд.

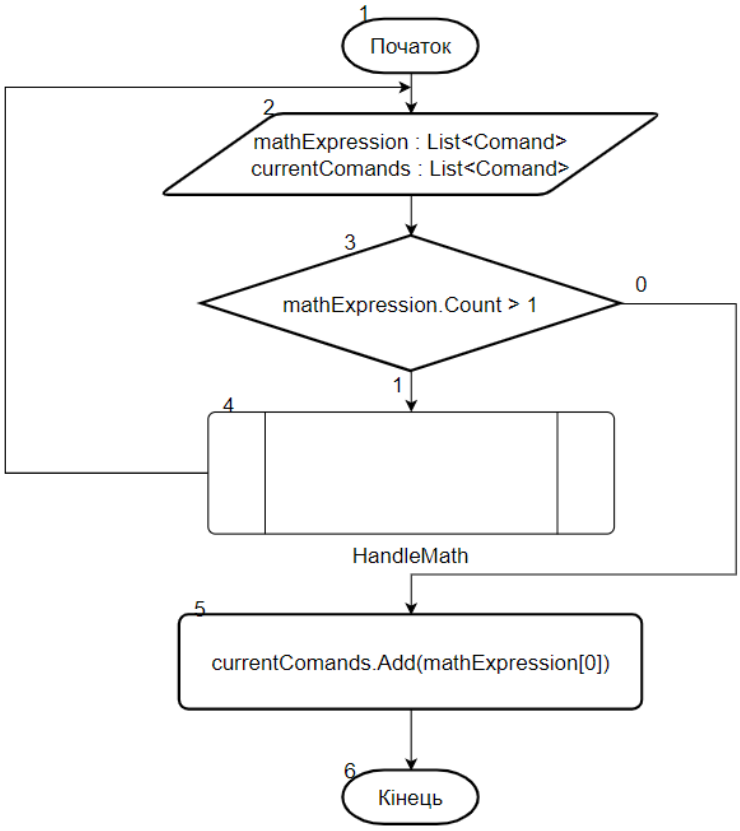


Рисунок 2.4 – Алгоритм створення математичного ланцюга команд

Розглянемо алгоритм методу «HandleMath». На вхід методу надходить вираз у вигляді списку операторів, операндів, змінних, що формують набір токенів. В якості типу для початкового токену використовується команда «GetValue». Перш за все потрібно знайти найглибшу операцію, тобто найглибшу пару дужок. Така операція має найвищий пріоритет. Потім в цьому виразі визначимо математичний оператор, що має найвищий пріоритет. У випадку однакових пріоритетів береться перший оператор зліва. Далі з обраних оператора та операндів створимо математичну команду. Зі списку токенів видаляється оператор та операнди, натомість додається дана команда. Таким чином формується математичний ланцюжок команд. В кінці алгоритму зайві дужки видаляються.

Усі вищезгадані команди, а також аналіз коду відбуваються у класі «CodeAnalizator» за допомогою функцій «ReadCode» «PreprocessCode», а також ряду допоміжних, що формують команди.

Виконання користувацького коду відбувається у класі «Interpretator». Даний клас містить код усіх запрограмованих користувачем класів. Також він має основну функцію – «RunLines». На рисунку 2.5 зображено алгоритм цієї функції.

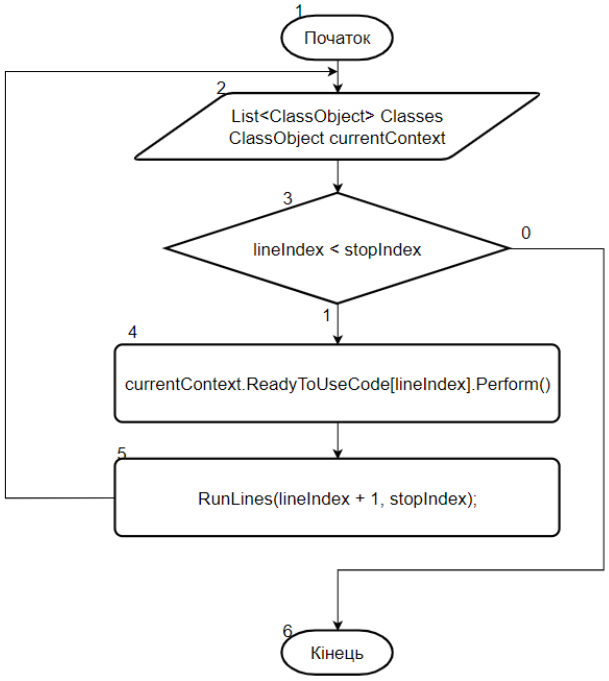


Рисунок 2.5 – Алгоритм методу «RunLines»

Цей метод отримує індекси початку та кінця виконання коду. Далі в циклі йде виконання команди за командою. Після виконання команди викликається цей метод з наступним початковим індексом. В ході виконання програми контекст може змінитися і будуть виконуватися нові команди відповідного класу.

Деякі частини коду мають блочну структуру, наприклад функції, цикли, розгалуження. Ці частини використовують метод «RunLines» для переходів між рядками коду. Розглянемо даний процес на прикладі блоків з логічними умовами. На рисунку 2.6 наведено алгоритм роботи методу «HandleCondition».

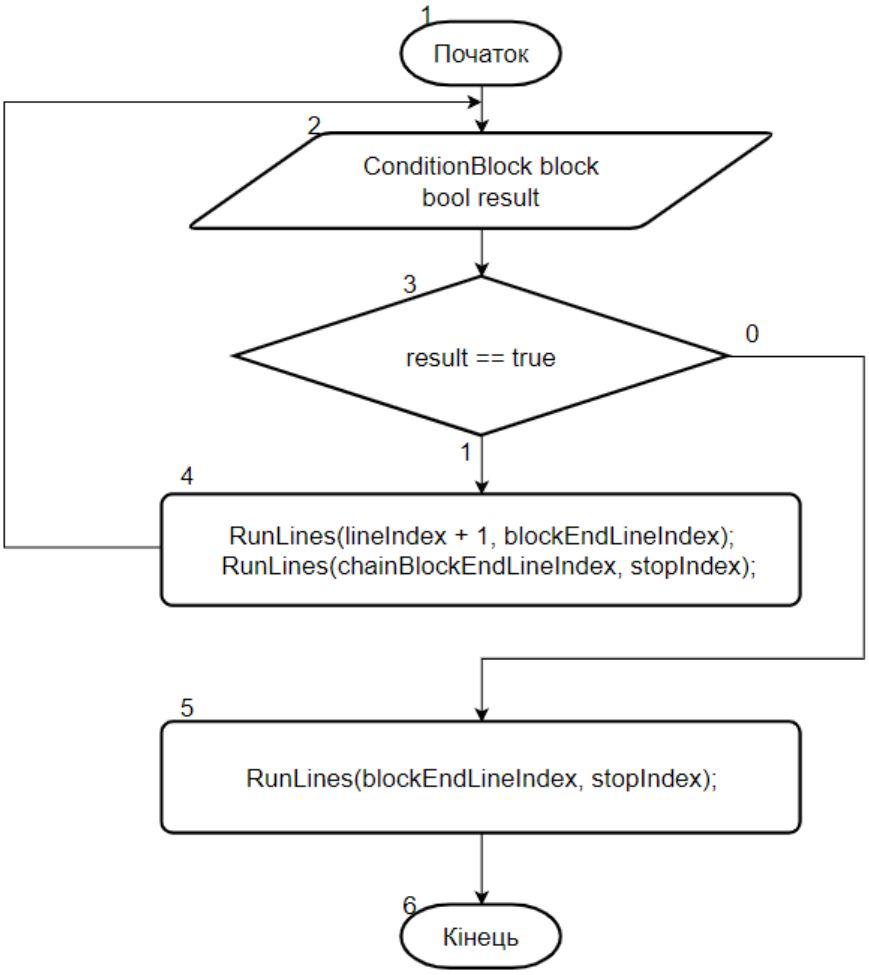


Рисунок 2.6 – Алгоритм методу «HandleCondition»

Умовні блоки складаються з наступних операторів: «if», «else if», «else». Ці блоки можу утворювати логічні ланцюги. Клас «ConditionBlock» містить інформацію саме про такі блоки та їх зв’язок. На вхід даного методу надходить результат виконання операції. Якщо він позитивний, то спочатку запускається даний умовний блок, а потім вказівник переходить до виконання коду у кінці всього ланцюга цих блоків. Якщо ж результат негативний – код виконується з наступної точки ланцюга або, якщо вона відсутня, з кінця умовного блоку.

Отже, розроблений алгоритм має переваги перед звичайними методами компіляції та інтерпретації. Він використовує препроцесинг вихідного коду, а потім створює набір інструкцій на виконання та зберігає їх в оперативній пам’яті. Таким чином виконання коду відбувається швидко та багаторазово. Це задовольняє вимогам інтегрованого середовища, беручи до уваги, що програмний код виконується впродовж однієї сесії, тобто одного завдання.

2.3 Розробка методу зберігання текстур тривимірних моделей

Одна із складових інтерактивного середовища – візуалізація шаблонів проєктування на основі тривимірних моделей. Тривимірні моделі складаються з певного набору інформації щодо їх форми (меш) та кольору (текстури). Текстура – це спосіб надання поверхні 3D моделі кольору, фактури, блиску, матовості та інших фізичних властивостей. Поняття текстурування є важливим етапом модулювання, так як використовує для зберігання даних растрове зображення, а це впливає на пам’ять програми, що її використовує. Крім того за допомогою текстур можна відтворити малі об’єкти на поверхні, на створення яких за допомогою полігонів пішло б значно більше ресурсів. Це можуть бути різні нерівності та рельєфи на тривимірній моделі [27].

Тому важливо обрати правильний метод текстурування, щоб знизити витрати пам’яті, що використовує програмне забезпечення. Загалом можна виділити два способи текстурувати об’єкт: за допомогою текстури та процедурно.

Процедурне текстурування — метод створення текстур у комп'ютерній графіці, при якому зображення створюється за допомогою програмного алгоритму (процедурного алгоритму), а не задається наперед [28]. Такий спосіб надає гнучкість при роботі з моделлю, адже кожен параметр можна регулювати будь-якої миті. Проте, щоб використати цю текстуру, її потрібно запекти. В результаті запікання утворюється звичайне растрове зображення з можливістю створення додаткових мап текстури.

Текстури можуть бути утворені з урахуванням фільтрації або без неї. З використання фільтрації вони інтерполюються, тим самим прибираючи артефакт утворення пікселів при наближенні до неї.

Іншим видом текстурування є PBR або ж рендер, що спирається на фізику. Залежно від поверхні об’єкта, від його стану (чи відбиває він, як дзеркало, чи має шорсткість, чи є краплі бруду/води на поверхні) змінюється вигляд самого об’єкта. PBR складається з набору параметрів:

- Color.
- Metallic.
- Specular.
- Roughness.
- Glossiness.
- Ambient Occlusion.
- Height.
- Normal map [29].

Таким чином, склавши усі параметри в результаті вийде готовий фізично коректний матеріал.

Кожен з каналів може використовувати певну текстуру або є мапу. Однією з основних таких текстур є мапа рель’єфу. Normal Mapping – це технологія, яка використовується для імітації нерівностей поверхні на об'єкті. Вона дозволяє надати моделі глибини, при цьому сам меш залишається без змін. Таким чином

можна створити низько-полігональну модель з даною текстурою високо-полігональної, що зменшить витрати пам’яті на зберігання [30].

Для створення інтерактивного середовища вивчення шаблонів проєктування в цілях збереження пам’яті додатку доцільно використати низько полігональні моделі. Як вже зазначалося, тривимірна модель складається із полігональної сітки і текстур. Проте ці два компоненти не пов’язані між собою, адже знаходяться в різних вимірах. Для їх поєднання використовується UV розгортка [31]. «UV mapping» — процес в 3D моделюванні, який полягає в накладанні двовимірного зображення на тривимірну модель. Власне під час цього процесу відбувається перетворення двовимірних координат у тривимірні. Таким чином текстури накладаються на сітку полігонів [32]. На рисунку 2.7 зображено приклад накладання розгортки на модель.

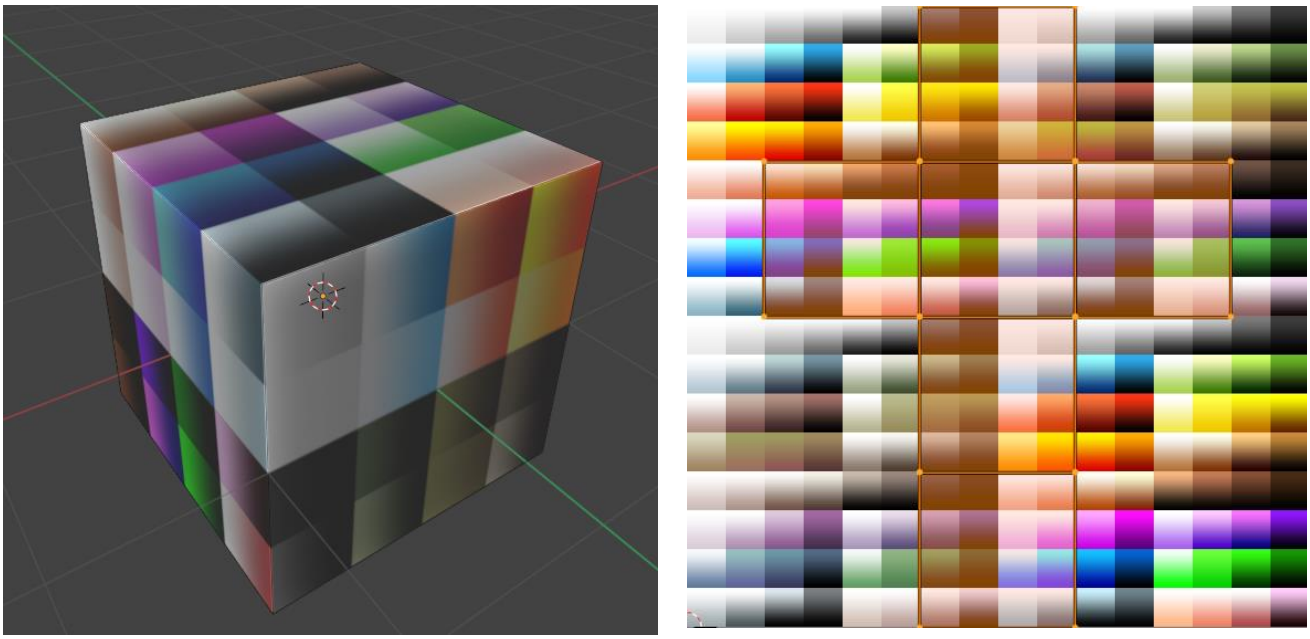


Рисунок 2.7 – Процес накладання текстури на модель або «UV mapping» (рисунок зліва – модель з текстурою, рисунок справа – текстура та UV-розгортка)

Об'єкти, в залежності від щільності полігональної сітки розділяють на два типи: низькополігональні (low poly) та високополігональні (high poly).

Низькополігональні ("лоупольні") - це об'єкти з мінімальною кількістю полігонів. Їх форма може виглядати грубо, але це вимушена жертва в тих випадках, коли потрібно заощадити ресурси комп’ютера [33].

Для низької кількості полігонів у моделі недоцільно створювати окрему текстуру для кожної моделі. Звісно можна використати одну текстуру на декілька моделей розгорнувши модель відповідно. Проте це все одно не є оптимальним рішенням. Зважаючи на факт, що всі моделі використовують низьку кількість полігонів, можна використати одну текстуру для усіх моделей. Для цього створимо растрове зображення і нанесемо на нього потрібні кольори. На рисунку 2.8 зображено таку текстуру для 64 кольорів. Відповідно на кожен колір приходиться 1 піксель.

Рисунок 2.8 – Текстура, що містить 64 кольори

Таким чином можна розфарбувати модель у 64 кольори. Для цього лише потрібно правильно використати UV розгортку. Щоб зафарбувати певну зону полігонів, потрібно знайти її на розгортці, виділити та масштабувати до нуля. Таким чином усі точки, що знаходяться на моделі будуть відповідати лише одній точці на розгортці. Відповідно ця зона буде зафарбована в колір, над яким знаходиться ця точка. Таким чином лише переміщуючи її по розгортці можна легко змінити колір відповідної зони полігонів.

Проте 64 кольори – це серйозне обмеження. Тому замість цієї можна використати повноцінну карту кольорів з градієнтом. На рисунку 2.9 зображено текстуру з широким спектром кольорів по кожному каналу.

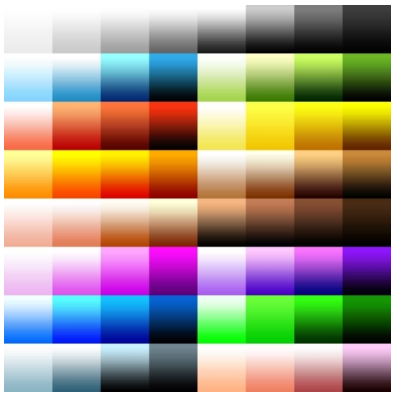


Рисунок 2.9 – Текстура з градієнтом, що містить широкий спектр кольорів

Така текстура займає більше простору, але це не має значення, бо лише один екземпляр цієї карти потрібен для текстурування усіх моделей. Але це не кінцевий варіант даного методу. Цю карту можна дублювати 4 рази і таким чином розширити кількість каналів для матеріалу. На рисунку 2.10 наведено цю текстуру.

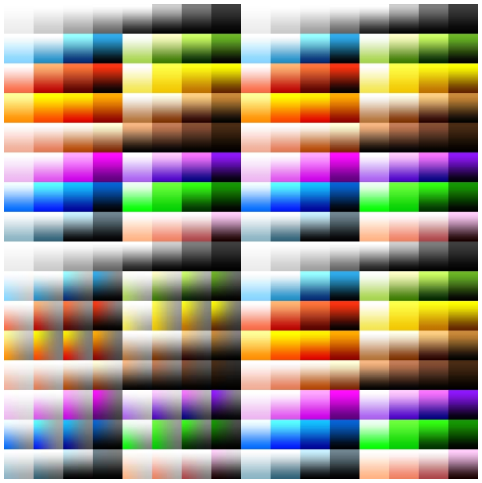


Рисунок 2.10 – Текстура з градієнтом, що повторюється 4 рази

Тепер до комплексного матеріалу для моделі можна використати 4 різні входи, тобто 4 канали. Для низько-полігональних моделей може знадобитися

використання прозорості, а також світіння. Візьмемо верхній лівий квадрат як вхід для основного кольору матеріалу. Далі створимо маски для ще двох каналів. На рисунку 2.11 зображено ці маски.

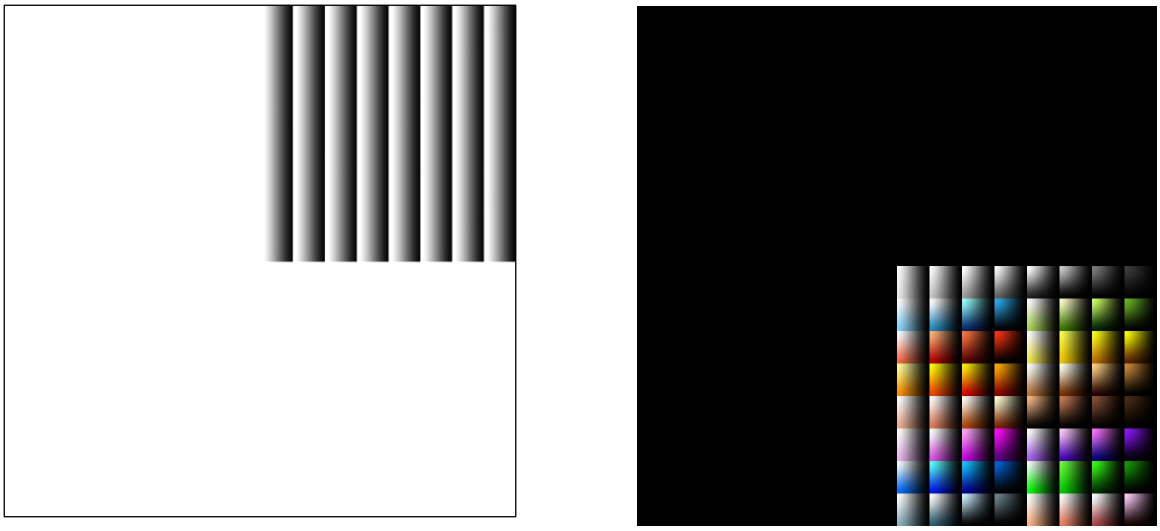


Рисунок 2.11 – Маска для прозорості (зліва) та світіння (справа)

Основна частина маски прозорості зафарбована білим кольором. Це означає, що в цих місцях модель буде зафарбована. А розмістивши координати UV розгортки в правому верхньому квадраті у правій частині кольорових квадратів (там де чорний колір на масці) можна зробити модель прозорою або напівпрозорою.

Маска світіння має інший основний колір – чорний. Він означає, що модель не має світіння за замовченням. Але розміщення координат розгортки у лівому нижньому квадраті змусить відповідні полігони світитися.

Таким чином, з’єднавши усі ці карти в одне ціле отримаємо можливість накладання текстур на низько-полігональні моделі з додатковими каналами для прозорості і світіння. Розгорнувши відповідним чином моделі на площині UV отримаємо бажаний результат. Для того, щоб розфарбувати будь-яку створену цим методом модель достатньо лише мати один матеріал, що складається з трьох текстур.

2.4 Висновки

Для створення інтерактивного середовища вивчення шаблонів проєктування розроблено спеціальну псевдомову, на якій користувач зможе виконувати поставлені завдання. Ця мови має усі необхідні характеристики, що потрібні для реалізації шаблонів. До них можна віднести підтримку ООП, змінних, операторів, класів, функцій, а також кольорове підсвічення коду. Щоб ця мова мала можливість виконання було створено комбінований алгоритм трансляції, що враховує можливості як компілятора так і інтерпретатора. До переваг даного алгоритму можна віднести повторюваність виконання коду без повторного аналізу та можливість зберігання команд в оперативній пам’яті. Таким чином реалізовано алгоритми для практичної частини середовища вивчення патернів проєктування.

Іншою важливою складовою інтерактивного середовища є візуалізація шаблонів. Вона базується на тривимірній графіці, що складається з моделей та тексту. Для заощадження пам’яті, використовуваної додатком, створено окремий алгоритм текстурування моделей. Він використовує всього одну основну текстуру та декілька додаткових для текстурування усіх моделей, використовуваних в додатку.

3 РЕАЛІЗАЦІЯ МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНОГО СЕРЕДОВИЩА ВИВЧЕННЯ ШАБЛОНІВ ПРОЄКТУВАННЯ

3.1 Варіантний аналіз та обґрунтування вибору засобів реалізації

Будь-яка розробка потребує ретельного аналізу засобів реалізації програмного продукту. Вибір цих засобів опирається на вимоги до ПЗ, а також на можливості розробника, знання технологій. Тому визначимо основні вимоги до даного продукту. Отже, кінцева розробка повинна містити наступні складові:

- Консоль для програмування коду та взаємодії з користувачем;
- Транслятор для мови програмування;
- Інтерактивна візуалізація шаблонів;
- Додаткова інформація про шаблони проєктування.

Виходячи з вимог, найкращим середовищем для створення застосунку є «Unity». Unity – це багато-платформовий інструмент для розробки відеоігор і застосунків, і рушій, на якому вони працюють [34]. Створені за допомогою Unity програми працюють на настільних комп'ютерних системах, мобільних пристроях та гральних консолях у дво- та тривимірній графіці, та на пристроях віртуальної чи доповненої реальності. «Unity» надає достатній набір інструментів для створення інтерактивного середовища та візуалізації шаблонів.

Наступною частиною, на яку потрібно звернути увагу є тривимірна графіка. Вона впливає на асоціативний механізм людини, а тому є важливою частиною середовища вивчення шаблонів проєктування. До найпопулярніших редакторів можна віднести Blender, Cinema 3D, Autodesk 3ds Max. Blender використовується для створення моделей в інді-іграх, анімацій, та іншого виду 3D графіки. Додаток має безкоштовну ліцензію, багато налаштувань, декілька рендерів та ін. Дистрибутив займає всього декілька сотень мегабайт. Cinema – має широкий функціонал, простий інтерфейс. Його основне призначення –

моушн-дизайн. 3ds Max є найпопулярнішим графічним редактором. Це дуже потужний інструмент, хоча є складним для новачків. Також програма має велику кількість бібліотек та розширень. До недоліків можна віднести відсутність безкоштовної ліцензії. Серед даних редакторів для розробки тривимірної графіки обрано Blender. Цей програмний пакет має усі потрібні засоби для розробки низько-полігональної графіки, поширюється безкоштовно та має необхідні інструменти для експорту [35].

Наступним засобом розробки є вибір мови програмування. Спираючись на вибір технології Unity очевидним рішенням є мова C#, так як це єдина мова, що підтримується цим рушієм. C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML[36].

Останній засіб розробки, без якого майже неможливо писати програмний код, є IDE (Integrated Development Environment). Він складається з набору декількох інструментів, а саме: текстовий редактор, компілятор, засоби збірки і відлагоджувач. Спираючись на обрану мову програмування потрібно вибрати один з наступних: Microsoft Visual Studio та Visual Studio Code. Visual Studio дозволяє створювати велику кількість різноманітних застосунків на багатьох мовах програмування. Вона має багато корисних інструментів, таких як IntelliSence, для роботи в команді, та можливість встановлення доповнень, до того є безкоштовна версія. Крім того вона має спеціальний набір інструментів для розробки під Unity. В свою чергу Visual Studio Code ж крос-платформенним, безкоштовним засобом розробки з великою кількістю плагінів, але має набагато менший базовий інструментарій [37]. Тому для розробки на Unity кращим вибором буде MS Visual Studio [38].

Усі діаграми, блок-схеми та інші види рисунків, що використовуютьcz в програмному додатку, розроблені за допомогою інтернет-ресурсу створення графіків Diagrams.net [39].

Отже, для розробки інтерактивного навчального середовища для шаблонів проєктування було обрано наступні програмні засоби: рушій розробки Unity для реалізації додатку, редактор тривимірної графіки Blender для створення візуалізації, мову програмування C# для програмування та MS Visual Studio в якості інтегрованого середовища розробки.

3.2 Структура та вимоги до створення псевдомови програмування

Мова програмування високого рівня схожа на звичайну мову, якою розмовляють люди. Зазвичай вона використовує латинський алфавіт та арабські цифри. Також більшість команд мають певне логічне значення для людини, що спрощує розуміння коду. Проте кількість слів у мові програмування значно менша від природної мови спілкування і слова вживаються лише за певними правилами і у певному порядку. Отож, мова програмування має фіксований набір слів і правил (синтаксис і семантика). Також мова програмування використовує різні оператори (зазвичай арифметичні, а також логічні) і деякі символи [40].

Синтаксис – це правила, що визначають комбінацію символів, які вживаються в певній мові програмування. Є інше поняття, що протиставляється синтаксису – семантика. Разом ці терміни визначають правила мови програмування та попереджають про можливі помилки.

Отже, синтаксис відповідає за помилки написаних слів, лексем і т.д. В той час семантика відповідає за помилки значень, наприклад, ділення на 0 або відсутнє значення змінної [41].

Зважаючи на факт, що шаблони проєктування є поняття абстрактне і незалежне від мови програмування, то для їх вивчення найкраще використати незалежну мову програмування, тобто псевдокод. Псевдокод – це неформальна мова програмування, як схожа за структурою на інші мови, але нехтує деталями коду, що не потрібні для розуміння алгоритму чи шаблону проєктування. Псевдомова є більш інтуїтивно зрозумілою. Також не існує формальних правил

написання цієї мови, тому єдині обмеження будуть накладені через вимоги стосовно патернів [42].

Отже, першою найважливішою вимогою до даного псевдокоду є підтримка об’єктно-орієнтованого програмування (ООП). Без цього неможливо реалізувати шаблони, адже вони орієнтовані на цю парадигму програмування. ООП включає в себе чотири принципи: наслідування, інкапсуляція, поліморфізм та абстракція. Дана мова повинна підтримувати як мінімум два принципи – наслідування і поліморфізм. Також, необов’язковим є ще й підтримка інкапсуляції (хоч вона і важлива при програмуванні, для розуміння шаблонів нею можна знехтувати). Очевидно, що до підтримки ООП входить також можливість створення класів та інтерфейсів та методів [43].

Наступною вимогою є підтримка змінних різних типів. В даному випадку важливими є змінні посилального типу, а також примітивних типів (в першу чергу числові та строкові змінні). В якості числового типу обрано змінну з плаваючою точкою.

Інша вимога до мови – це оператори. Вона повинна підтримувати базові арифметичні оператори, а також логічні оператори. Разом з тим повинна бути реалізована можливість розгалуження коду за рахунок цих операторів. Циклічність виконання є необов’язковою умовою, але може спростити написання коду.

Остання додаткова, але досить корисна функціональність до мови програмування – це кольорове підсвічення коду. Воно дозволяє легше орієнтуватися в написаній програмі завдяки виділенню основних структурних елементів мови.

На основі описаних вище вимог розробимо синтаксис псевдомови для вивчення шаблонів проєктування.

На рисунку 3.1 зображено синтаксис написання змінних та методів.

```
method Main()
{
    float x

    Test(x)
}

method Test(float f)
{
    print (f)
}
```

Рисунок 3.1 – Синтаксис змінних і методів

Оголошення змінної починається із типу змінної, потім іде її назва. Форма запису наступна “[тип змінної] [назва змінної]”.

Оголошення методу має наступний синтаксис “method [назва методу] ([тип першого параметру][назва першого параметру],…)”. Кожен метод є блоком коду, а отже використовує фігурні дужки, що містять тіло методу.

Виклик методу має наступний синтаксис “[назва методу] ([перший аргумент,…)”.

На рисунку 3.2 зображено синтаксис класів, реалізацію наслідування та поліморфізму.

```
class Program : Base
{
    override method Test(float f)
    {
        print (f)
    }
}
```

Рисунок 3.2 – Синтаксис класів на принципі ООП

Оголошення класу має наступний вигляд “class [назва класу]”. Після оголошення ідуть фігурні дужки з тілом класу.

Наслідування реалізовується за наступною формулою “class [назва похідного класу] : [назва базового класу]”.

Поліморфізм реалізовується в класах наслідниках на основі функцій. Базовий клас має наступну форму запису функції “virtual method [назва методу] ([тип першого параметру][назва першого параметру],…)”. Клас-наслідник має наступну форму запису функції “override method [назва методу] ([тип першого параметру][назва першого параметру],…)”.

На рисунку 3.3 зображено синтаксис інших блоків коду та операторів.

```
if (3 > 2 & 6 > 5)
{
    print (3 + 2)
}
else
{
    print ("no")
}
```

Рисунок 3.3 – Синтаксис операторів

Структура логічного розгалуження коду наступна: спочатку іде базова умова “if ([умова])” і фігурні дужки з блоком коду, далі може йти або фінальна умова з використанням ключового слова “else”, або наступна у ланці умова зі структурою “else if ([умова])”.

Серед логічних операторів, що використовуються у псевдомові додано наступні:

- “>” – оператор більше;
- “<” – оператор менше;
- “>=” – оператор більше або дорівнює;
- “<=” – оператор менше або дорівнює;
- “==” – оператор дорівнює;
- “!=” – оператор не дорівнює;

- “||” – оператор логічного додавання;
- “&&” – оператор логічного множення.

Серед арифметичних операторів, що використовуються у псевдомові реалізовано наступні:

- “+” – оператор додавання;
- “-” – оператор віднімання;
- “*” – оператор множення;
- “/” – оператор ділення.

Окремо розглянемо реалізацію оператора привласнення. На рисунку 3.4 зображено синтаксис варіантів використання оператора “=”.

```
method Main()
{
    float x = 5
    Base refVar = new Base()
}
```

Рисунок 3.4 – Синтаксис оператора привласнення

Для привласнення значень примітивних типів чи копіювання значень використовується наступний синтаксис: “[назва змінної] = [значення привласнення]”.

Для створення змінних посилальних типів використовується спеціальний синтаксис “[тип змінної] [назва змінної] = new [тип змінної] ()”.

Отже, в результаті створено псевдо-мову програмування, що складається з усіх необхідних компонентів, потрібних для програмування шаблонів проєктування. До них входять: змінні значимих і посилальних типів, ООП, класи та функції, різні оператори. Також описано синтаксис мови і принципи використання конструкцій.

3.3 Розробка структури додатку

Будь-який додаток чи веб-сайт, що має достатню кількість інформації, побудований за певною навігаційною структурою, що дозволяє “подорожувати” та переглядати різний контент цього ресурсу. Додаток для вивчення шаблонів проєктування не є винятком. Створена структура повинна дозволяти користувачеві отримувати швидкий доступ до різних частин додатку в будь-який момент часу.

Для навчальних додатків найкраще пасує деревовидна структура. Вона дозволяє легко систематизувати увесь навчальний матеріал по розділами і підрозділам відповідно. Вона складається з вузлів та гілок. Вузлами можуть бути основні розділи, в той час як гілки відповідають за більш вузьку, спеціалізовану інформацію [44]. Отже, розробимо навігацію додатку на основі цієї структури.

На рисунку 3.5 зображено загальну схему навігаційної структури додатку для вивчення шаблонів проєктування.

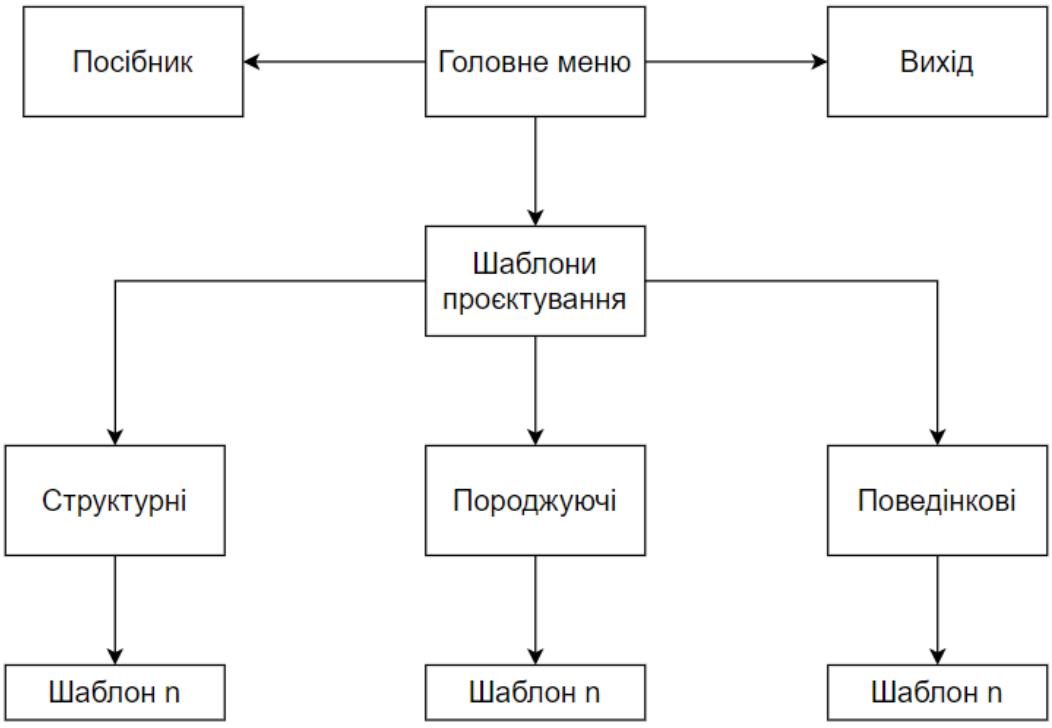


Рисунок 3.5 – Основна навігаційна структура додатку

При завантаженні додатку користувач увійде в головне меню. Звідси можна перейти до наступних частин застосунку: посібник, шаблони проєктування та вийти. Посібник містить навчальну інформацію, яку варто оглянути перед використанням додатку. Пункт шаблони проєктування відкриває діалогове вікно, де можна вибрати категорію шаблонів проєктування. Патерни GoF мають три категорії: структурні, породжуючі, поведінкові. Вибравши потрібну категорію, користувач може вибрати потрібний шаблон для його вивчення.

При виборі конкретного шаблону додаток завантажує відповідну сцену інтерактивного середовища, де він може вчитися та тренуватися. Сцена складається з візуалізованої части, консолі розробника, до якої користувач може звернутися в будь-який момент та довідкової інформації про шаблон проєктування.

Таким чином, для додатку вивчення шаблонів проєктування обрано деревоподібну структуру. Вона забезпечує систематизацію навчальних матеріалів за категоріями, наприклад за видами шаблонів проєктування: структурні, породжуючі та поведінкові.

3.4 Розробка програмної консолі

Консольний інтерфейс користувача є одним з основних компонентів інтерактивного середовища, адже це спосіб взаємодії з системою. Він виступає як редактор для написання коду, так і для виводу інформації на екран. Консоль створюється на базі інтерфейсу командного рядка. Цей механізм має ряд особливостей та переваг і недоліків, проте без нього неможливе написання коду.

Інтерфейс командного рядка загалом використовується в наступних випадках:

- система має обмежені ресурси (менші затрати пам’яті ніж в інших типів інтерфейсів);

- набір команд відбувається набагато швидше, ніж навігація по меню;
- пакетні файли – послідовність команд, що зберігається в одному файлі і є невеликою програмою.

В даному випадку цей інтерфейс використовуватиметься як редактор програмного коду.

До недоліків можна віднести:

- даний інтерфейс не є «дружнім» для новачків, а тому потребує певного пояснення чи документації;
- потрібен час, щоб звикнути до нових команд [45].

Отже, розробимо структуру консолі для програмування. Загалом вона складатиметься з трьох частин:

1. безпосередньо інтерфейс командного рядка, що використовується для написання коду;
2. вивід консолі (вивід інформації з коду на екран);
3. нумерація рядків коду (для зручнішої навігації);
4. можливість перемикання між програмованими класами;
5. кнопка запуску коду.

На рисунку 3.6 зображено схему розміщення основних елементів консолі.

Рисунок 3.6 – Структура консолі

Розберемо деталі створення кожної частини консолі.

Інтерфейс командного рядка є основною частиною консолі, що дозволяє писати код. Він займає найбільшу частину екрану та повинен підтримувати кольоровий текст для виділення коду, а також візуальні підказки та повідомлення про помилки.

Вивід консолі представляє собою невелике вікні, де буде відбуватися будь-який користувацький вивід інформації під час виконання коду. Під час чергового запуску коду вивід буде очищено.

Нумерація рядків коду здійснюється зліва зверху вниз для зручної навігації, а також інформування про помилки коду.

Спеціальний випадний список дозволятиме перемикатися між класами і змінювати наповнення інтерфейсу командного рядка відповідно до вибраного класу.

Кнопка запуску коду розпочне його виконання, а отже розпочнеться процес симуляції завдання.

На рисунку 3.7 зображено розроблену консоль, що відповідає поставленим до неї вимогам.

Рисунок 3.7 – Розроблена консоль

Також інтерфейс використовує спеціальний консольний шрифт під назвою «Profontwindows».

Отже, в даному розділі створено консоль розробника, яка призначена для програмування задач по шаблонам проєктування. Вона складається з редактору коду, консольного виводу, кнопок вибору класів для програмування та запуску коду на виконання. Таким чином, консоль надає користувачу усі потрібні інструменти для виконання завдань.

3.5 Реалізація інтерактивного середовища вивчення шаблонів проєктування

Розглянемо реалізацію інтерактивного середовища на прикладі шаблону «ланцюжок обов’язків». Для цього розберемо принцип роботи цього патерну. Його суть – обробляти певні запити обробниками в певному порядку. Кожен обробник формує “ланцюг”, по якому проходить запит та вирішує чи обробляти його самостійно чи передати далі [46].

Отже, змоделюємо ситуацію по аналогії до життя. Припустімо, що до бару заходять відвідувачі. Кожен відвідувач замовляє якусь страву чи напій. Отже, відвідувачі – це обробники. А завідувач закладом – той хто подає страви, тобто запити на обробку. Він завжди подає страву першому відвідувачу тому, що йому так зручніше. В той же час, перший на черзі має вирішити чи це те, що він замовив чи можливо варто передати далі. Таким чином, користувач додатку має запрограмувати даний шаблон, щоб вирішити цю ситуацію.

На рисунку 3.8 зображено загальну структуру шаблону «ланцюжок обов’язків». Інтерфейс «Handler» визначає спільного обробника. Зазвичай він містить методи обробки запиту та встановлення наступного обробника. Клас базового обробника реалізовує методи інтерфейсу. Також він має посилання на наступного обробника у ланцюжку. Конкретні обробники відповідно просто реалізовують клас базового. У класі клієнта формується ланцюжок обробників. Він може відправляти запити будь-якому з них.

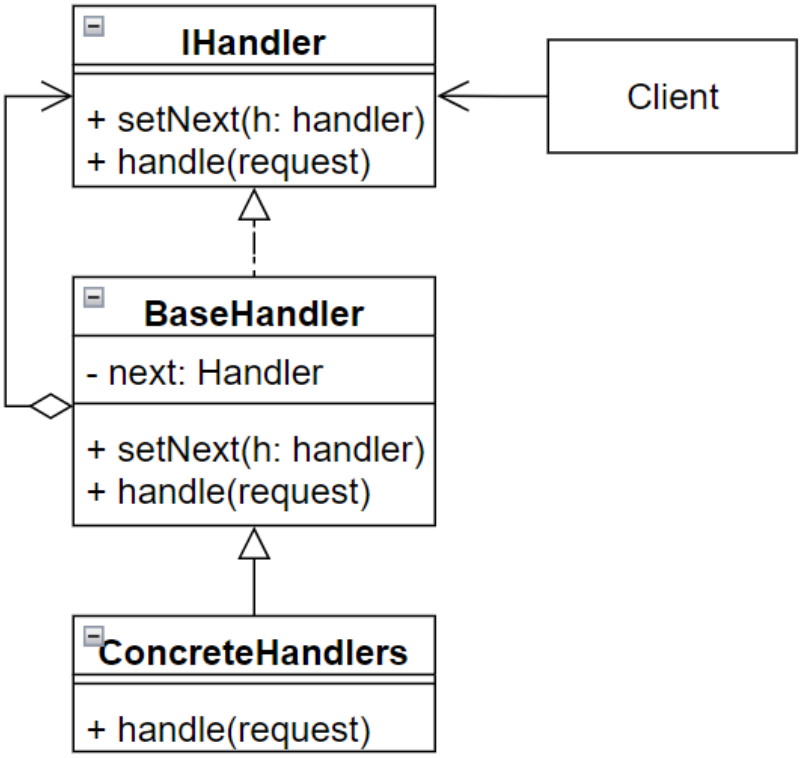


Рисунок 3.8 – Структура шаблону «ланцюжок обов’язків»

Маючи загальну структуру шаблону, реалізуємо його під змодельовану ситуацію. Отже, замість класу «BaseHandler» буде клас «CustomerHandler». Конкретними реалізаціями виступатимуть класи «HunterHandler» «SheriffHandler» та «WorkerHandler». Це віднповідно і є відвідувачі. В якості клієнта буде клас «Bar». Він відповідатиме за прийом запитів та їх обробку. Основним класом в будь якій програмі є «Program» з якого і починається виконання коду. Для користувача додатком ці класи вже будуть наповнені певним шаблоном, щоб можна було легше зорієнтуватися, а деякі реалізації навіть будуть заблоковані.

Запустивши завдання на виконання з даним шаблоном користувач відразу побачить сцену з баром та відвідувачами. Це і є візуалізація шаблону. Користувач може ознайомитися з ввідною інформацією щодо завдання. На рисунку 3.9 зображено початкову сцену з завданням шаблону «ланцюжок обо’язків».

Рисунок 3.9 – Сцена «бар», що є візуалізацією завдання до шаблону «ланцюжок обов’язків»

Натиснувши на клавішу «Tab» користувач може переключитися на інтерфейс консолі. В даному вікні відбуватиметься основний процес навчання. Його основною задачею буде запрограмувати шаблон проєктування відповідно до завдання. Серед функціоналу користувачу буде доступне випадне меню з доступними класами для реалізації, кнопка запуску коду на виконання, додаткові дані, потрібні для виконання завдання (функції та змінні, що завчасно підготовлені). На рисунку 3.10 зображено інтерфейс консолі розробника.

Рисунок 3.10 – Інтерфейс консолі розробника

Також користувач може в будь-який момент звернутися до довідника та дізнатися додаткову інформацію про конкретний шаблон проєктування. На рисунку 3.11 наведено інтерфейс цього довідника.

Рисунок 3.11 – Інтерфейс довідника по шаблонам проєктування

В разі успішного виконання завдання користувач отримає відповідне сповіщення. Також є можливість повернення до початкового коду, якщо щось пішло не так.

Отже, в результаті розробки створено інтерактивне середовище для вивчення шаблонів проєктування GoF. Реалізовано консоль розробника, де користувач має змогу безпосередньо писати та запускати на виконання програмний код. Відповідно до шаблону проєктування створено конкретне завдання, яке має бути запрограмовано та успішно виконано. Візуалізації показує хід виконання завдання.

3.6 Висновки

В результаті аналізу засобів реалізації інтерактивного середовища вивчення шаблонів проєктування обрано наступні: основне середовище

розробки Unity, так як воно дозволяє поєднати візуалізацію та інтегроване середовище розробки, Blender – графічний редактор тривимірних моделей, має усі засоби для створення тривимірної графіки, мову програмування C#, яку використовує Unity та IDE Visual Studio, так як воно має потрібні засоби для обраних технологій.

Розроблено навігаційну структуру додатку. Вона побудована на основі деревовидної структури, що дозволяє зручно розміщувати навчальні матеріали. Також створено консоль розробника, за допомогою якої користувач має програмувати шаблони проєктування. Вона надає весь необхідний функціонал (редактор коду, вивід текстової інформації, можливість вибору класів та запуск коду на виконання, має функцію підсвічення коду).

Реалізовано інтерактивне середовище та приклад шаблону «ланцюжок обов’язків». Дане середовище встановлює зв’язок між консоллю розробника та візуальною складовою, а також надає додаткову інформацію щодо завдання, яке потрібно виконати користувачу.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

4.1 Аналіз методів тестування

Тестування програмного забезпечення – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому його мають використовувати. До тестування входить як пошук помилок так і оцінка програмних засобів. Також під тестуванням розуміють процес перевірки відповідності поставлених вимог до реалізованої функціональності [47].

Основними цілями тестування можуть бути наступні:

- перевірка ПЗ на відповідність програмного продукту на виході відносно нормативних, бізнес, технічних, функціональних вимог та вимог користувачів;
- виявлення технічних помилок/багів з подальшим їх усуненням (Quality Control);
- оцінка зручності, продуктивності, безпеки, локалізації, сумісності та встановлення системи [48].

Тестування поділяють на статичне та динамічне. Статичне – коли тестуються результати розробки ПЗ. Це перевірка програмного коду без запуску самої програми. В той час динамічне має на меті перевірку під час запуску додатку. Під час статичного перевіряється документація. Динамічне тестування допомагає виявити помилки програмного забезпечення за рахунок виконання тестів.

Також тестування розподіляють за доступом до системи: «чорна скринька» та «біла скринька». Під час першого тестуються різні набори вхідних та вихідних даних, так як воно не має безпосереднього доступу до програмного коду. Тестування «чорної скриньки» не може покрити усі можливі варіанти, так як вхідних даних може бути багато. Завдяки цьому методу можна виявити

некоректну роботу функцій, помилки інтерфейсу, характеристики ПЗ. Тестування «білої скриньки» досліджує внутрішню роботу програми. Перевіряються усі модулі програми та їх взаємодія один з одним [48].

Серед розглянутих методів застосуємо методи динамічного тестування у поєднанні з обома методами «скриньки», так як вони доповнюють одне одного. Таким чином, буде протестовано основні функції додатку, а також його модулі і їх взаємодію.

4.2 Тестування методу трансляції програмного коду

Найголовнішу частину інтерактивного середовища являє собою саме програмування шаблонів проєктування. Тому перевіримо працездатність створеного методу трансляції та консолі загалом. Напишемо невеликий програмний код, який виконує певні операції. Результат виведемо з допомогою вбудованої функції «print». На рисунку 4.1 наведено даний тест-кейз.

Рисунок 4.1 – Тестування виконання основних операції та операторів на псевдо-мові програмування

З рисунку 4.1 видно, що усі операції виконано успішно. На вхід функції подано змінну, що означає вік людини. В залежності від віку виводиться одне з двох повідомлень.

Наступним тестом перевіримо працездатність об’єктно-орієнтованого програмування, так як це основа шаблонів проєктування. Для цього створимо декілька класів та використаємо принципи наслідування і поліморфізму. Результат тесту наведено на рисунку 4.2.

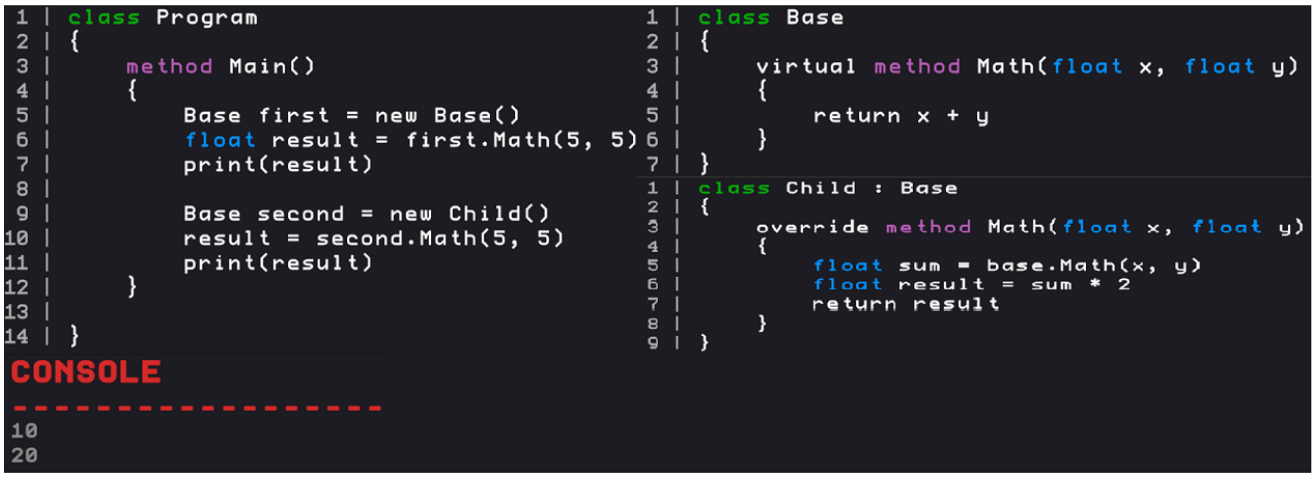


Рисунок 4.2 – Тестування реалізації наслідування та поліморфізму (рисунок зліва – клас «Program», рисунки справа – класи «Base» та «Child», рисунок внизу – консольний вивід програми)

На рисунку 4.2 зображено класи «Base» та «Child», що реалізують процес наслідування. Вони містять функцію «Math», що виконує арифметичну операцію. Ця функція перевантажена таким чином, що в залежності від контексту буде виконуватися код, що належить відповідному класу. Консольний вивід показує, що в першому випадку виконується метод базового класу, а в наступному – класу наслідника. Таким чином, з результатів тесту видно, що наслідування і поліморфізм реалізовано правильно.

Також в попередньому тесті було задіяно кнопки переходу між класами та запуску коду на виконання, а значить функціональна частина інтерфейсу консолі працює належним чином.

Звичайне виконання коду можливе і в умовах стандартного середовища програмування разом із типовою мовою програмування. Але інтерактивне середовище використовує саме візуалізацію шаблонів, тому перевіримо зв’язок програмного середовища і візуалізацію. Для цього до стандартних методів псевдо мови додають спеціальні зовнішні методи. Таким чином можна визначити певні заготовані методи, які користувач мусить використати, щоб виконати завдання. В прикладі з шаблоном «ланцюжок обов’язків» створимо метод «GiveFoodToCustomer». При виклику цієї функції виконання коду повинно призупинитися та розпочатися анімація видачі їжі. Результат тестування зображено на рисунку 4.3.

Рисунок 4.3 – Тестування зв’язку між модулем програмування та візуалізації (рисунок зверху – візуалізація методу видачі їжі, рисунок знизу – метод «GiveFoodToCustomer» та консольний вивід програми)

З рисунка вище зрозуміло, що при виконанні стороннього методу відбувається його візуалізація, а це означає що даний функціонал реалізовано правильно.

Таким чином, провівши тестування основного модуля інтерактивного середовища можна зробити висновок, що він працює належним чином, як функціональність так і інтерфейс.

4.3 Висновки

Тестування програмного забезпечення є важливою ланкою основного циклу його розробки. Тому проведено аналіз методів та засобів тестування та вирішено, що найкращим підходом до тестування додатку для вивчення шаблонів проєктування є використання декількох методів у поєднанні. Таким чином, можна перевірити різні аспекти роботи ПЗ: як функціональність так і інтерфейс.

Для тестування обрано частину додатку, що відповідає за виконання програмного коду, його трансляцію, а також зв’язок з візуальною частиною. В результаті тестування з’ясувалося, що транслятор псевдо-мови програмування працює належним чином, виконуючи команди користувача. Візуалізація реагує на команди від транслятора, що підтверджує працездатність цих модулів. Інтерфейс функціональний та відповідає на дії користувача.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення наукового аудиту науково-дослідної роботи

Для проведення наукового аудиту зазвичай оцінюють науковий ефект. До ознак наукового ефекту відносять: новизну роботи, рівень теоретичного опрацювання, перспективність, рівень розповсюдження результатів, можливість реалізації. Для характеристики наукового ефекту НДР використаємо перші два параметри: ступінь наукової новизни та рівень теоретичного опрацювання.

Отже, створене програмне забезпечення можна охарактеризувати як відносно нове (40 балів) за ступенем наукової новизни. Дана робота має елементи новизни в методах дослідження (розробка методу трансляції програмного коду, спосіб зберігання текстури тривимірних моделей), в роботі знайдено ефективне рішення щодо вивчення шаблонів проєктування на основі інтерактивної візуалізації даних.

За рівнем теотретичного опрацювання НДР можна віднести до третього рівня (та що створює нові алгоритми, програми). Третій рівень має 60 балів.

Знайдемо показник, який характеризує науковий ефект:

$$\mathrm{E}_{\text{нау}} = 0{,}6 \cdot k_{\text{нов}} + 0{,}4 \cdot \ k_{\text{теор}} = 0{,}6 \cdot 40 + 0{,}4 \cdot 60 = 48$$

Характеристика показника наукового ефекту здійснюється на основі граничних значень. Досягнутий рівень показника вважається достатнім. Такий рівень показника характеризується створенням інтерактивного середовища для вивчення шаблонів проєктування на основі методів вивчення складної та комплексної інформації, а також нових алгоритмів трансляції для розробленої мови програмування.

5.2 Проведення комерційного та технологічного аудиту науково-технічної розробки

Комерційного та технологічного аудиту проводиться для оцінки науково-технічного рівня та комерційного потенціалу розробки. Для цього було залучено 3-х незалежних експертів: к.т.н., доц. каф. ПЗ Черноволик Г. О. (Експерт 1), к.т.н., доц. каф. ПЗ Бабюк Н. П. (Експерт 2) та к.т.н., доц. каф. ПЗ Майданюк В. П. (Експерт 3).

Оцінювання здійснюється за 12-ма критеріями: технічна здійсненність концепції, ринкові переваги (наявність аналогів), ринкові переваги (ціна продукту), ринкові переваги (технічні властивості), ринкові переваги (експлуатаційні витрати), ринкові перспективи (розмір ринку), ринкові перспективи (конкуренція), практична здійсненність (наявність фахівців), практична здійсненність (наявність фінансів), практична здійсненність (потреба нових матеріалів), практична здійсненність (термін реалізації), практична здійсненність (розробка документів) [49]. Кожен критерій оцінюється з використанням 5-ти бальної шкали. Результати наведено в таблиці 5.1.

Таблиця 5.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Експерти		
	експерт 1	експерт 2	експерт 3
	Бали:		
1. Технічна здійсненність концепції	3	3	3
2. Ринкові переваги (наявність аналогів)	3	2	3
3. Ринкові переваги (ціна продукту)	3	3	4
4. Ринкові переваги (технічні властивості)	4	4	3
5. Ринкові переваги (експлуатаційні витрати)	3	3	2
6. Ринкові перспективи (розмір ринку)	2	3	3
7. Ринкові перспективи (конкуренція)	3	4	3
8. Практична здійсненність (наявність фахівців)	3	4	4

Продовження таблиці 5.1

Критерії	Експерти		
	експерт 1	експерт 2	експерт 3
	Бали:		
9. Практична здійсненність (наявність фінансів)	4	4	4
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	$СБ_1$=40	$СБ_2$=42	$СБ_3$=41
Середньоарифметична сума балів $СБ_с$	$СБ_с = \frac{СБ_1 + СБ_2 + СБ_3}{3} = 41$		

На основі результатів таблиці 5.1 можна зробити висновок, що розробка має високий рівень комерційного потенціалу і науково-технічного рівня. Такий рівень потенціалу було досягнуто за рахунок розширення навчальних можливостей науково-технічної розробки, а саме завдяки новим методам вивчення шаблонів проєктування на основі інтерактивності.

В таблиці 1.2 розглянуто аналоги ресурсів для вивчення шаблонів проєктування: інтернет-ресурси «Refactoring guru», «Metanit» та додатки «GoF Design Patterns», «Design Patterns in C#». Проведено порівняльний аналіз аналогів за декількома критеріями. Основними недоліками аналогів є необхідність підключення до мережі (інтернет-ресурси) або відсутність безкоштовного доступу до усього контенту (додатки). Також до недоліків можна віднести відсутність підтримки псевдо-мови в деяких ресурсах. Розроблюваний продукт вирішує дані проблеми, а також пропонує вдосконалення методів навчання, а саме створення інтерактивного середовища, що дозволяє програмувати шаблони

проєктування та методи візуалізації, що перетворюють структуру шаблонів в анімацію. А розроблений метод текстурування заощаджує використання пам’яті тривимірними моделями. Таким чином, розробка власної реалізації є актуальною, щоб поєднати вже вирішені завдання, покрити недоліки аналогів та застосувати нові методи навчання.

Отже, високий рівень комерційного потенціалу потенціалу та науково-технічної розробки в першу чергу пояснюється зростання ефективності вивчення шаблонів проєктування з рахунок комбінованого методу навчання, що поєднує практичні задачі та візуалізацію. По друге, додаток показує значно вищу якість в порівнянні з аналогами за рахунок удосконалених методів трансляції мови програмування та зберігання текстур. Як результат відбувається зменшення часу, необхідного для вивчення шаблонів проєктування та збільшується ефективність засвоєння інформації.

5.3 Прогнозування витрат на виконання науково-дослідної роботи

Витрати на здійснення науково-дослідної роботи розраховуються за наступними категоріями: витрати на оплату праці, відрахування на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, спецустаткування для наукових робіт, програмне забезпечення для наукових робіт, витрати на роботи, які виконують сторонні підприємства, установи і організації, накладні (загальновиробничі) витрати та ін. Обрахуємо витрати за кожною категорією.

5.3.1 Витрати на оплату праці

До витрат на оплату праці належать витрати на виплату основної та додаткової заробітної плати дослідникам та працівникам.

Основна заробітна плата розробників розраховується за формулою:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} \cdot t_i}{Т_р} \quad (5.1)$$

де k – кількість розробників;

$М_{пі}$ – місячний посадовий оклад конкретного розробника, грн;

t_i – кількість днів роботи розробника, дн.;

T_P – кількість робочих днів у місяці (вважаємо, що $T_P = 21$ день).

Обчислені результати за цією формулою занесемо в таблицю 5.2

Таблиця 5.2 – Витрати на заробітну плату розробників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Кількість днів роботи	Витрати на оплату праці, грн.
Науковий керівник	12000	571,43	20	11 428,57
Програмний інженер	9000	428,57	30	12 857,14
Всього				24 285,71

Додаткові витрати на оплату праці розраховуються за формулою:

$$З_{дод} = З_о \cdot \frac{Н_{дод}}{100\%} = 24\,285{,}71 \cdot 0.1 = 2\,428{,}57 \text{ (грн)}$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати (10 %).

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату працівників розраховується як 22 % від суми основної та додаткової заробітної плати за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} = (24\,285{,}71 + 2\,428{,}57) \cdot 0{,}22$$
$$= 5\,877{,}14 \text{ (грн)}$$

5.3.3 Спецустаткування для наукових робіт

Балансову вартість спецустаткування розраховують за формулою:

$$В_{спец} = \sum_{i=1}^{k} Ц_i \cdot С_{пр.i} \cdot К_i \quad (5.2)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$С_{пр.i}$ – кількість одиниць устаткування відповідного найменування, шт.;

$К_i$ – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування ($К_i$ = 1,1 … 1,12).

Візьмемо коефіцієнт додаткових витрат рівним 1,11. Проведені розрахунки зведено до таблиці 5.3.

Таблиця 5.3 – Витрати на придбання спецустаткування по кожному виду

Найменування	Кількість, шт.	Ціна за одиницю, грн.	Вартість, грн.
Ноутбук	1	30 000	30 000
Мишка	1	500	500
Зарядний пристрій	1	300	300
Носій інформації	1	800	800
Всього			31 600

Отже, $В_{спец}$ з урахування додакових витрат становить 35 076 грн.

5.3.4 Програмне забезпечення для наукових робіт

Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot С_{прг.i} \cdot К_i \quad (5.3)$$

де $Ц_{іпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$С_{пр.і}$ – кількість одиниць програмного забезпечення відповідного найменування, шт.;

$К_і$ – коефіцієнт, що враховує що враховує інсталяцію, налагодження програмного забезпечення ($К_і$ = 1,1 … 1,12).

Візьмемо коефіцієнт витрат на інсталяцію рівним 1,11. До програмного забезпечення, що необхідне при розробці, належить пакет програм Office 2021. Розрахуємо вартість ПЗ з урахуванням коефіцієнту додаткових робіт.

$$В_{прг} = 2190 \cdot 1 \cdot 1{,}11 = 2430{,}9 \text{ (грн)}$$

5.3.5 Амортизація обладнання, програмних засобів та приміщень

Амортизаційні відрахування розрахуємо за використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12} \quad (5.4)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання та програмних засобів, 1 місяць;

$Т_в$ – строк корисного використання обладнання, 5 років.

Враховуючи податковий вартісний критерій для ОЗ до амортизаційних нарахувань належить лише ноутбук, тому розрахуємо цей критерій за формулою.

$$А_{обл} = \frac{30000}{5} \cdot \frac{1}{12} = 500 \text{ (грн)}$$

5.3.6 Інші витрати

До інших витрати належать ті, що не відображені в попередніх статтях витрат і розраховуються як 50 % від суми основної заробітної плати працівників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%} = 24\,285{,}71 \cdot 0{,}5 = 12\,142{,}86 \text{ (грн)}$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат:

$$В_{заг} = З_о + З_{дод} + З_н + В_{спец} + В_{прг} + А_{обл} + І_в \quad (5.5)$$

$$В_{заг} = 24\,285{,}71 + 2\,428{,}57 + 5\,877{,}14 + 35\,076 + 2\,430{,}9 + 500 + 12\,142{,}86 = 82\,741{,}18 \text{ (грн)}$$

Загальні витрати на завершення науково-дослідної роботи розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} = 82\,741{,}18 \,/\, 0{,}9 = 91\,934{,}64 \text{ (грн)}$$

5.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Збільшення чистого прибутку у потенційного інвестора протягом декількох років розраховується за формулою:

$$\Delta П_і = (\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_і \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}) \quad (5.6)$$

де $\Delta Ц_о$ – зміна основного якісного показника від впровадження результатів науково-технічної розробки в аналізованому році;

N – основний кількісний показник, який визначає величину попиту на аналогічні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_о$ – основний якісний показник, який визначає ціну реалізації нової науково-технічної розробки в аналізованому році, $Ц_о = Ц_б \pm \Delta Ц_о$;

$Ц_б$ – основний якісний показник, який визначає ціну реалізації існуючої науково-технічної розробки у році до впровадження результатів;

ΔN – зміна основного кількісного показника від впровадження результатів науково-технічної розробки в аналізованому році;

λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. λ =0,8333;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. ρ = 0,3;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у ϑ = 18%.

В результаті впровадження наукової розробки покращується якість певного продукту, що дозволяє підвищити ціну його реалізації на 500 грн. Кількість користувачів збільшиться протягом першого року на 500, другого – 200, третього – 200. Реалізація продукції до впровадження результатів наукової розробки складала 2000 користувачів, а її ціна – 2000 грн.

Розрахуємо показник прибутку впродовж трьох років відносно базового.

$\Delta П_1 = (500 \cdot 2000 + (2000 + 500) \cdot 500) \cdot 0{,}8333 \cdot 0{,}3 \cdot (1–18/100) = 461\,231{,}55$ (грн)

$\Delta П_1 = (500 \cdot 2000 + (2000 + 500) \cdot 700) \cdot 0{,}8333 \cdot 0{,}3 \cdot (1–18/100) = 563\,727{,}45$ (грн)

$\Delta П_1 = (500 \cdot 2000 + (2000 + 500) \cdot 900) \cdot 0{,}8333 \cdot 0{,}3 \cdot (1–18/100) = 666\,223{,}35$ (грн)

Далі розрахуємо приведену вартість збільшення чистих прибутків, що їх може отримати потенційний інвестор від впровадження розробки за формулою:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t} \quad (5.7)$$

де Т – період часу, протягом якого очікується отримання позитивних результатів від впровадження науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,05…0,15;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{461\,231{,}55}{(1+0{,}05)^1} + \frac{563\,727{,}45}{(1+0{,}05)^2} + \frac{666\,223{,}35}{(1+0{,}05)^3} = 1\,526\,094{,}34 \text{ (грн)}$$

Далі розрахуємо величину початкових інвестицій, які інвестор має вкласти для впровадження розробки за формулою:

$$PV = k_{інв} * ЗВ \quad (5.8)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. $k_{інв} = 2$.

Отже, PV = 2 · 91 934,64 = 183 869,28 (грн).

Тоді чистий приведений дохід для інвестора становитиме:

$$Е_{абс} = ПП - PV = 1\,526\,094{,}34 - 183\,869{,}28 = 1\,342\,225{,}06 \text{ (грн)}$$

Додатний показник свідчить про потенційну зацікавленість інвесторів у впровадженні розробки, проте не є достатнім. Для остаточного рішення потрібно розрахувати внутрішню економічну дохідність за формулою:

$$\text{Е}_\text{в} = \sqrt[\text{Т}_\text{ж}]{1 + \frac{\text{Е}_\text{абс}}{PV}} - 1 = \sqrt[3]{1 + \frac{1\,342\,225{,}06}{183\,869{,}28}} - 1 = 1{,}02 \text{ або } 102\%$$

Далі визначимо бар'єрну ставку дисконтування $\tau_{мін}$. Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ визначається за формулою:

$$\tau_{\text{мін}} = d + f \tag{5.9}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; d = 0,09;

f – показник, що характеризує ризикованість вкладення інвестицій; f =0,15.

Отже, $\tau_{\text{мін}} = 0{,}09 + 0{,}15 = 0{,}24$ або 24 %.

Оскільки величина $E_в > \tau_{мін}$, то потенційний інвестор може бути зацікавлений у фінансуванні впровадження науково-технічної розробки.

Далі розрахуємо період окупності інвестицій за формулою:

$$T_{\text{ок}} = \frac{1}{E_{\text{в}}} \tag{5.10}$$

де $\text{Е}_\text{в}$ – внутрішня економічна дохідність вкладених інвестицій.

Отже, період окупності буде наступний:

$$T_{\text{ок}} = \frac{1}{1{,}02} = 0{,}98 \text{ року}$$

Оскільки період окупності менше трьох років, це свідчить про комерційну привабливість науково-технічної розробки.

5.5 Висновки

Таким чином, в даному розділі проаналізовано економічну складову науково-дослідної роботи за темою «Розробка інтерактивного середовища для вивчення шаблонів проєктування». Проведено науковий аудит, за результатами досліджень створюване ПЗ має характеристику «відносно нове» за ступенем наукової новизни, що означає в роботі присутні елементи новизни в методах дослідження.

Проведено комерційний та технологічний аудит НДР. В результаті дослідження розробки трьома незалежними експертами сформовано таблицю оцінювання (таблиця 5.1) комерційного потенціалу. За результатами дослідження з’ясувалося, що розробка має високий рівень комерційного потенціалу і науково-технічного рівня. Такий рівень показника досягнуто в результаті впровадження нових методів вивчення шаблонів проєктування, що беруть за основу методи практикування та візуалізації, які не використовуються в аналогах. Також підвищено продуктивність програмного продукту за рахунок удосконалення методів трансляції і зберігання текстур тривимірних моделей.

Здійснено прогнозування витрат на виконання НДР. До цієї категорії враховано наступні показники: витрати на оплату праці (основну та додаткову), відрахування на соціальні заходи, спец-устаткування та програмне забезпечення для наукових робіт, амортизація обладнання та інші витрати. На основі вище згаданих показників розраховано загальні витрати на завершення НДР, що становить 91934,64 грн.

Далі розраховано показник ефективності вкладених інвестицій. Величина цього показника складає 102 %, що більше за 24 %, тобто мінімальний поріг, що визначає потенційну можливість інвестування. Також розраховано період окупності, який дорівнює 0,98 року. Такий термін свідчить про комерційну привабливість проєкту.

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи створено інтерактивне середовище для вивчення шаблонів проєктування GoF (Gang of Four).

Проведено порівняльний аналіз методів вивчення складного матеріалу, до якого належать шаблони проєктування за наступними критеріями: складність сприйняття інформації, швидкість роботи, незалежність від інших методів та відсоток засвоєння. На основі аналізу визначено, що метод практики та навчання інших мають найвищі показники, а тому їх необхідно використати у додатку для вивчення шаблонів проєктування.

В результаті проведення порівняльного аналізу аналогів засобів вивчення шаблонів проєктування визначено, що розробка власного додатку є актуальним завданням. При порівнянні враховано наступні критерії: тестова та графічна складова, безкоштовність ресурсу, підключення до мережі, підтримка псевдо-мови.

Проведено аналіз методів створення трансляторів для мов програмування та визначено, що є потреба у розробці власного методу, який підлаштовується під створену мову програмування. Також розглянуто особливості програмних засобів середовища програмування з метою визначення найважливіших з них для подальшого впровадження у розроблений додаток.

На основі аналізу методів вивчення складної інформації розроблено комбінований метод навчання, що є найбільш ефективним рішенням. Даний метод базується на основі інтерактивного середовища, що поєднує текстові, графічні дані та практичні завдання для підвищення відсотку засвоєння інформації.

Таким чином для реалізації практичної частини інтерактивного середовища удосконалено метод трансляції, що використовується під спеціально розроблену псевдо-мову програмування. Алгоритм трансляції базується на особливостях компілятора та транслятора, а саме запам’ятовує

виконувані інструкції в оперативній пам’яті, а не в окремому файлі. Таким чином підвищується швидкість виконання коду та заощаджується пам’ять.

Для візуалізації шаблонів використовується тривимірна графіка. Звичайні методи текстурування використовують по одній основній та декілька додаткових мап для фарбування кожної моделі. Тому розроблено метод, що використовує одну основну та декілька додаткових для текстурування усіх створених моделей. Таким чином даний спосіб заощаджує пам’ять системи, підвищує швидкодію додатку та прискорює роботу з тривимірними моделями.

Проведено аналіз засобів реалізації інтерактивного середовища. Основний додаток створено на основі середовища розробки Unity. Тривимірні моделі розроблялися в програмному пакеті Blender. В якості мови програмування обрано C#. На основі обраних програмних засобах розроблено псевдо-мову програмування, консоль розробника для користувача та реалізовано інтерактивне середовище вивчення шаблонів проєктування.

Проведено аналіз методів тестування ПЗ та здійснено тестування розробленого програмного додатку. Протестовано його основну функціональність, а саме коректність виконання користувацького коду, зв’язок між модулем програмування та візуалізацією. Усі тести пройдено успішно, додаток працює належним чином.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Архітектура та проектування ПЗ [Електронний ресурс] URL: http://surl.li/dyblw
2. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design patterns : elements of reusable object-oriented software. N.-Y.:Addison-Wesley, 1994. 416
3. Навіщо знати патерни [Електронний ресурс] URL: https://refactoring.guru/uk/design-patterns/why-learn-patterns
4. Шаблони проєктування. Початок. [Електронний ресурс]. URL: https://travelscode.com/shabloni-proektuvannya-pochatok/
5. Васянович Є., Катєльніков Д. Використання шаблонів проектування у сучасному підході до розробки програмного забезпечення. / Матеріали конференції «L Науково-технічна конференція підрозділів Вінницького національного технічного університету (2021)», Вінниця, 2021. [Електронний ресурс]. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/author/submission/12552
6. Васянович Є., Ліщинська Л. Загальна характеристика підходів до вивчення складного матеріалу на прикладі шаблонів проєктування. / Матеріали конференції міжнародна науково-практична інтернет-конференція «Електронні інформаційні ресурси: створення, використання, доступ», Вінниця, 2022
7. Як і для чого використовувати візуалізацію даних? [Електронний ресурс]. URL: http://eidos.org.ua/novyny/yak-i-dlya-choho-vykorystovuvaty-vizualizatsiyu-danyh/
8. Дизайн-патерни – просто, як двері. [Електронний ресурс]. URL:https://learn.ztu.edu.ua/pluginfile.php/632/mod_resource/content/1/Design Patterns_AndriyBuday.pdf
9. Візуалізація або текст? Засоби сприйняття інформації [Електронний ресурс]. URL: https://kibit.edu.ua/vizualizaciya-ili-tekst/

10. Піраміда навчання Едґара Дейла. [Електронний ресурс]. URL: https://fsp.kpi.ua/ua/piramida-navchannya-edgara-dejla/
11. Патерни проєктування [Електронний ресурс]. URL:https://refactoring.guru/uk/design-patterns
12. Основи патернів проєктування [Електронний ресурс]. URL:https://metanit.com/sharp/patterns/1.1.php
13. Design Patterns (GoF) in Java [Електронний ресурс]. URL:https://play.google.com/store/apps/details?id=com.jaypeesoft.dpad&hl=uk&gl=US
14. Design Patterns in C# [Електронний ресурс]. URL:https://play.google.com/store/apps/details?id=com.myapp.designpatternscs&hl=uk&gl=US
15. Транслятор [Електронний ресурс]. URL: http://www.ndu.edu.ua/liceum/php/php.pdf
16. Компілятор. Основні задачі компілятора. [Електронний ресурс]. URL: https://studfile.net/preview/358216/#2
17. Різниця між компілятором та інтерпретатором [Електронний ресурс]. URL: https://sites.google.com/site/erinaprogram/theory/01-programming-languages/1-3-compiler-interpreter
18. Препроцесор [Електронний ресурс]. URL: https://learn.microsoft.com/ru-ru/cpp/preprocessor/preprocessor?view=msvc-170
19. Лексичний аналіз та скінченні автомати [Електронний ресурс]. URL: https://csc-knu.github.io/sys-prog/lectures/tex/02.pdf
20. Синтаксичні аналізатори [Електронний ресурс]. URL: https://vuzlit.com/994431/sintaksichni_analizatori
21. Інтегроване середовище розробки [Електронний ресурс]. URL: http://um.co.ua/13/13-9/13-92003.html
22. Інтегроване середовище програмування [Електронний ресурс]. URL: https://studfile.net/preview/9236942/page:2/

23. Редактори коду [Електронний ресурс]. URL: https://uk.javascript.info/code-editors
24. Пометун О.І., Пироженко Л.В. Сучасний урок. Інтерактивні технології навчання: наук.-метод. посібн. Київ : Видавництво А.С.К., 2004. 192 с.
25. Інтерактивні методи навчання [Електронний ресурс]. URL: https://sites.google.com/site/nmcmyk/naukova-dialnist/interaktivni-metodi-navcanna
26. Візуалізація [Електронний ресурс]. URL: https://socialdata.org.ua/manual/manual5/
27. Текстура (тривимірна графіка) [Електронний ресурс]. URL: https://www.wiki.uk-ua.nina.az/Текстура_(трьохвимірна_графіка).html
28. How to Create Procedural Textures for Design & Engineering [Електронний ресурс]. URL: https://ntopology.com/blog/how-to-procedural-textures/
29. Текстурування. Частина 3. PBR та матеріали [Електронний ресурс]. URL: https://habr.com/ru/post/458696/
30. Normal Map. Практичний посібник [Електронний ресурс]. URL: https://www.school-xyz.com/normal-map-prakticheskoe-rukovodstvo
31. Що Таке UV Мапінг? [Електронний ресурс]. URL: https://3dcoat.com/ua/articles/article/what-is-uv-mapping/
32. Як зробити UV-розгортку в Blender [Електронний ресурс]. URL: https://you-hands.ru/2019/12/01/kak-sdelat-uv-razvertku-v-blender-2-8/
33. Основні принципи низькополігонального моделювання [Електронний ресурс]. URL: https://3dcoat.com/ua/articles/article/basic-principles-of-low-poly-modeling/
34. Unity [Електронний ресурс]. URL: https://unity.com
35. Blender [Електронний ресурс]. URL: https://www.blender.org
36. Історія появи мови програмування C# [Електронний ресурс]. URL: https://www.bestprog.net/uk/2022/05/22/c-the-history-of-the-emergence-of-the-c-programming-language-and-microsoft-net-technology-ua/

37. 10 найкращих IDE [Електронний ресурс]. URL: https://timeweb.com/ru/community/articles/5-luchshih-ide-1
38. Microsoft Visual Studio [Електронний ресурс]. URL: https://visualstudio.microsoft.com
39. Diagrams.net [Електронний ресурс]. URL: https://app.diagrams.net
40. Алфавіт і синтаксис мови програмування [Електронний ресурс]. URL: https://studfile.net/preview/9236942/page:3/
41. Syntax [Електронний ресурс]. URL: https://www.computerhope.com/jargon/s/syntax.htm
42. Псевдокод [Електронний ресурс]. URL: https://datascience.eu/ru/программирование/псевдокод/
43. Основні засади ООП [Електронний ресурс]. URL: https://training.epam.ua/News/Items/275?lang=ua
44. Деревоподібна структура [Електронний ресурс]. URL: https://jak.waykun.com/articles/derevopodibna-struktura-ce.html
45. Вступ до інтерфейсу командного рядка [Електронний ресурс]. URL: https://tutorial.djangogirls.org/uk/intro_to_command_line/
46. Ланцюжок обов'язків [Електронний ресурс]. URL: https://refactoring.guru/uk/design-patterns/chain-of-responsibility/
47. Канер Кем, Фолк Джек, Нгуен Енг Кек. Тестування програмного забезпечення. Фундаментальні концепції менеджменту бізснес-додатків. Київ : ДіаСофт, 2001. 544 с.
48. Що таке тестування програмного забезпечення та яке його значення [Електронний ресурс]. URL: https://www.quality-assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/
49. Козловський В. О., Лесько О. Й., Кавецький В. В. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. Вінниця : ВНТУ, 2021. 42 с.

ДОДАТКИ

Додаток А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
«16» вересня 2022 р.

Технічне завдання
на магістерську кваліфікаційну роботу
«Розробка методів і засобів для створення інтерактивного середовища вивчення шаблонів проєктування GoF»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
д.т.н., проф. каф. ПЗ Л.Б.Ліщинська
"15" вересня 2022 р.

Виконав:
студент гр.1ПІ-21м Васянович Є.А.
"15" вересня 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування GoF».

Галузь застосування - системи навчання.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 205-А ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності вивчення шаблонів проєктування за рахунок комбінації методів сприйняття інформації, а саме використання методів візуалізації інформації у поєднанні з виконанням задач на програмування програмування.

Призначення роботи – розробка методів і засобів створення інтерактивного середовища вивчення шаблонів проєктування.

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Пометун О.І., Пироженко Л.В. Сучасний урок. Інтерактивні технології навчання: наук.-метод. посібн. Київ : Видавництво А.С.К., 2004. 192 с.
2. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design patterns : elements of reusable object-oriented software. N.-Y.:Addison-Wesley, 1994. 416 p.
3. Сергей С.З. Конструирование компиляторов. Німеччина : LAP LAMBERT Academic Publishing, 2015. 571 с.

4. Технічні вимоги

Кольоровий режим – TrueColor; формат зберігання тривимірних моделей – «.fbx»; формат зберігання растрових зображень – «.png»; розмір основних текстур – 512x512; тип полігональної моделі – трикутна; середовище розробки – Unity; мова розробки – C#; операційна система – Windows 10 та вище; псевдомова програмування та транслятор до неї.

5. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування	17.09.2022-28.09.2022
2	Моделювання інтерактивного середовища вивчення шаблонів проєктування та проєктування програмних засобів	29.09.2022-25.10.2022
3	Програмна реалізація методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування	26.10.2022-20.11.2022

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
4	Тестування програмого додатку	21.11.2022-29.11.2022
5	Економічна частина	30.11.2022-8.12.2022

9. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

Додаток Б

Протокол перевірки

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: «Розробка методів і програмних засобів для створення інтерактивного середовища вивчення шаблонів проєктування GoF»

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 21м

Науковий керівник:

Unicheck	
Оригінальність	**96,1%**
Схожість	3,9%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Васянович Є. А.

Керівник роботи _____ Ліщинська Л. Б.

Додаток В

Лістинг коду

Лістинг програми, що відповідає за виконання вже готового програмного коду, клас Interpreter.

```
public void RunLines(int lineIndex, int stopIndex, List<BaseType> localVariables =
null)
    {
        if (currentContext.ReadyToUseCode[lineIndex].comands != null)
        {
            foreach (var item in currentContext.ReadyToUseCode[lineIndex]?.comands)
            {
                if (item is Condition condition)
                {
                    bool result = false;
                    if (item.Perform(ref contextVar, localVariables) != null)
                    {
                        result = true;
                    }
                    HandleCondition(lineIndex + 1, stopIndex, result, condition.parent
as ConditionBlock);
                    stopIndex = lineIndex;
                    break;
                }
                List<BaseType> parameters = null;
                if (localVariables != null)
                    item.Perform(ref contextVar, localVariables);
                else
                {
                    parameters = item.Perform(ref contextVar);
                }
                if (item is FunctionComand func)
                {
                    if (globalComands.TryGetValue(func.GetName(), out
Action<List<BaseType>> value))
                    {
                            value(parameters);
                    }
                    else
                    {
                        prevContexts.Push(currentContext);
                        if (func.context != null)
                        {
                            currentContext = func.context;
                            prevVarContexts.Push(contextVar);
                        }

                        FunctionBlock function =
currentContext.AllFunctions[func.GetName()];
                        HandleFunction(parameters, function.blockStartLineIndex,
function.blockEndLineIndex);

                        bool isUsed = false;
                        while (prevVarContexts.Count > 0)
                        {
                            ReferenceType temp = prevVarContexts.Pop();
```

```
                            if (temp != contextVar)
                            {
                                contextVar = temp;
                                isUsed = true;
                                break;
                            }
                        }
                        if (!isUsed)
                            contextVar = null;

                        currentContext = prevContexts.Pop();

                        if (function.ReturnValue != null)
                        {
                            localVariables = new List<BaseType>();
                            localVariables.Add(function.ReturnValue);
                            function.ReturnValue = null;
                        }
                    }
                }
            }
        }

        if (lineIndex + 1 < stopIndex)
            RunLines(lineIndex + 1, stopIndex);
    }
    public void HandleFunction(List<BaseType> parameters, int lineIndex, int
stopIndex)
    {
        RunLines(lineIndex, stopIndex, parameters);
    }
    public void HandleLoop(int lineIndex, int stopIndex, int repeatNum)
    {
        int blockEndLineIndex = lines[lineIndex + 1].parent.blockEndLineIndex;
        for (int i = 0; i < repeatNum; i++)
        {
            RunLines(lineIndex + 1, blockEndLineIndex, false);
        }
        RunLines(blockEndLineIndex, stopIndex, false);
    }
    public void HandleCondition(int lineIndex, int stopIndex, bool result,
ConditionBlock block)
    {
        int blockEndLineIndex = block.blockEndLineIndex;
        int chainBlockEndLineIndex = block.chainBlockEndLineIndex;
        if (result)
        {
            RunLines(lineIndex + 1, blockEndLineIndex);
            RunLines(chainBlockEndLineIndex, stopIndex);
        }
        else
            RunLines(blockEndLineIndex, stopIndex);
          }
```

Лістинг програми, що відповідає за перетворення початкового коду у готовий до виконання, клас CodeAnalizator.

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

```
using UnityEngine;
using System.Linq;
using System.Text.RegularExpressions;

public static class CodeAnalizator
{
    static ClassObject classObject;
    static string[] specialSingleSymbols = { "(", ")", "+", "-", "*", "/", ">",
"<" };
    // prepare code for separating on comands
    public static ClassObject ReadCode(string code)
    {
        // remove white strings and move '{' to the new string
        // resolve parenthesis like this (x+ 12 *aor > 1 or(1 > abs(2))) -> ( x
+ 12 * aor > 1 or ( 1 > abs ( 2 + 3 ) ) )
        List<string> resultCode = code.Split('\n').ToList();
        for (int i = 0; i < resultCode.Count; i++)
        {
            if (resultCode[i].Contains('\t') || resultCode[i].Contains('\r'))
            {
                resultCode[i] = resultCode[i].Replace("\t", string.Empty);
                resultCode[i] = resultCode[i].Replace("\r", string.Empty);
            }
            if (specialSingleSymbols.Any(x => resultCode[i].Contains(x)))
            {
                resultCode[i] = ResolveMathStringWithWhiteSpaces(resultCode[i]);
            }
            if (resultCode[i].Contains('{'))
            {
                resultCode[i] = resultCode[i].Replace("{", string.Empty);
                resultCode.Insert(i + 1, "{");
                i++;
            }
        }
        resultCode.RemoveAll(x => string.IsNullOrWhiteSpace(x));
        return PreProcessCode(resultCode.ToArray());
    }
    static string ResolveMathStringWithWhiteSpaces(string str)
    {
        string testSymbol = "";
        int startIndex = 0, endIndex = 0;
        for (int i = 0; i < str.Length; i++)
        {
```

```
                testSymbol += str[i];
                endIndex = i;
                if (str[i] == ' ')
                {
                    testSymbol = "";
                    startIndex = i + 1;
                    continue;
                }
                // if isn't "or|and" continue
                if   ((testSymbol.Contains("||")   &&   testSymbol   !=   "||"   ||
testSymbol.Contains("&&") && testSymbol != "&&") && str[endIndex + 1] != ' ' &&
str[endIndex + 1] != '(')
                {
                    continue;
                }
                // if operator create new string
                if (specialSingleSymbols.Any(x => x == str[i].ToString()))
                {
                    testSymbol = str[i].ToString();
                    startIndex = i;
                    endIndex = i;
                    if ((str[i] == '>' || str[i] == '<') && str[i + 1] == '=')
                    {
                        i++;
                        testSymbol += str[i];
                        endIndex = i;
                    }
                }
                // insert spaces between operators
                if (specialSingleSymbols.Any(x => x == testSymbol) || (testSymbol ==
"||" || testSymbol == "&&") && (startIndex == 0 || str[startIndex - 1] != ' ' ||
str[startIndex - 1] != ')') && (str[endIndex + 1] != ' ' || str[endIndex + 1] != '('))
                {
                    if (endIndex + 1 != str.Length)
                    {
                        if (str[endIndex + 1] != ' ')
                        {
                            str = str.Insert((endIndex + 1) == str.Length ? endIndex
: endIndex + 1, " ");
                        }
                    }
                    if (startIndex == 0 || str[startIndex - 1] != ' ')
                    {
```

```
                        str = str.Insert(startIndex == 0 ? startIndex + 1 :
startIndex, " ");
                    }
                }
            }
            return str;
        }
        // separate code on comands
        static ClassObject PreProcessCode(string[] code)
        {
            List<Line> lines = new List<Line>();
            string parentName = GetParrentName(code[0]);

            ICodeBlock currentParent = new ClassBlock(0, code.Length - 1);
            classObject = new ClassObject(parentName, currentParent);
            if (code[0].Contains(':'))
            {
                classObject.HasParent = true;
                classObject.ParentName = code[0].Split(':')[1].Trim();
            }

            for (int i = 0; i < code.Length; i++)
            {
                bool isNotNeedGetValue = false;
                bool isNotNeedMath = false;
                bool isNeedAssignment = true;
                List<Comand> currentComands = new List<Comand>();
                List<Comand> mathExpression = null;
                string conditionPatern = @"((\s*if\s+\(|\s*else\s*))";

                if (Regex.IsMatch(code[i], conditionPatern))
                {
                    ConditionBlock block = new ConditionBlock();
                    if (code[i].Contains("else"))
                    {
                        block.prevChainElement = (ConditionBlock)lines[i - 1].parent;
                    }
                    block.blockStartLineIndex = i;
                    ICodeBlock prevParent = currentParent;
                    currentParent = block;
                    block.parent = prevParent;
                }
                if (code[i].Contains("method"))
```

```
                {
                    currentComands.Add(HandleFunctionDeclaration(ref  currentParent,
code[i], i));
                    lines.Add(new Line(code[i], i, currentComands, currentParent));
                    continue;
                }
                if (code[i].Contains("}"))
                {
                    currentParent.blockEndLineIndex = i;
                    if (currentParent is ConditionBlock condition)
                    {
                        condition.chainBlockEndLineIndex = i;
                        ConditionBlock conditionParent = condition.prevChainElement;
                        while (conditionParent != null)
                        {
                            conditionParent.chainBlockEndLineIndex = i;
                            conditionParent = conditionParent.prevChainElement;
                        }
                    }
                    lines.Add(new Line(code[i], i, currentComands, currentParent));
                    currentParent = currentParent?.parent;
                    continue;
                }

                string creatingRefTypePatern = @"((\W|\s+)new\s+)";
                bool isNeedFuncCall = true;
                if (Regex.IsMatch(code[i], creatingRefTypePatern))
                {
                    isNeedFuncCall = false;
                    isNotNeedGetValue = true;
                    currentComands.Add(HandleRefTypeCreation(currentParent,
code[i]));
                }


                if (Regex.IsMatch(code[i], conditionPatern))
                {
                    isNeedFuncCall = false;
                    isNotNeedMath = true;
                    isNotNeedGetValue = true;
                    isNeedAssignment = false;
                    Comand com = HandleLogicCondition(code[i], currentParent);
                    if (com != null)
```

```
                    currentComands.Add(com);
                }

                string functionCallPatern = @"(\w+\s*\(\w*|\s*\))";
                if (Regex.IsMatch(code[i], functionCallPatern) && !isNotNeedGetValue
&& isNeedFuncCall)
                {
                    isNotNeedMath = true;
                    isNotNeedGetValue = true;
                    Comand com = HandleFunctionCall(code[i], currentParent);
                    if (com != null)
                        currentComands.Add(com);
                }

                string mathPatern = @"(\+|\-|\*|\/)";
                if (Regex.IsMatch(code[i], mathPatern) && !isNotNeedMath)
                {
                    isNotNeedGetValue = true;
                    mathExpression = CreateMathList(code[i], currentParent);
                    while (mathExpression.Count > 1)
                    {
                        mathExpression = HandleMath(currentParent, mathExpression);
                    }
                    currentComands.Add(mathExpression[0]);
                }

                string returnPatern = @"(\s*return\s+)";

                if (Regex.IsMatch(code[i], returnPatern))
                {
                    if (currentComands.Count > 0)
                        currentComands.Add(HandleReturn(code[i],      currentParent,
currentComands[0]));
                    else
                        currentComands.Add(HandleReturn(code[i],      currentParent,
null));
                }

                string getValuePatern = @"(\d|\w)";
                if (Regex.IsMatch(code[i], getValuePatern) && !isNotNeedGetValue)
                {
                    if (code[i].Contains("="))
```

```
                        currentComands.Add(GetValue(currentParent,
code[i].Split('=')[1]));
                }

                string assignmentPatern = @"(\s*)=(\s*)";
                if       ((Regex.IsMatch(code[i],       assignmentPatern)       ||
code[i].Contains("float") || code[i].Contains("string")) && isNeedAssignment)
                {
                    Comand prev = currentComands.LastOrDefault();
                    if (prev is FunctionComand || prev is RefTypeCreationComand)
                    {
                        currentComands.Add(HandleAssignment(code[i],  currentParent,
null));
                    }
                    else
                    {
                        currentComands.Remove(prev);
                        currentComands.Add(HandleAssignment(code[i],  currentParent,
prev));
                    }
                }
                lines.Add(new Line(code[i], i, currentComands, currentParent));
            }
            classObject.ReadyToUseCode = lines;
            return classObject;
        }
        static List<Comand> CreateMathList(string line, ICodeBlock parent)
        {
            List<Comand> mathExpression = new List<Comand>();
            string expression = null;
            if (line.Contains("="))
            {
                expression = line.Split('=')[1].Trim();
            }
            else
            {
                if (line.Contains('('))
                    expression          =          line.Substring(line.IndexOf("("),
line.LastIndexOf(")") - (line.IndexOf("(") + 1));
                else
                    expression = line.Trim();
            }
            string[] tokens = expression.Split(' ');
```

```
            for (int i = 0; i < tokens.Length; i++)
            {
                if (tokens[i] != "return")
                    mathExpression.Add(GetValue(parent, tokens[i]));
            }

            return mathExpression;
        }
        static Comand HandleFunctionDeclaration(ref ICodeBlock parent, string expression, int lineIndex)
        {
            string methodName = expression.Split("method")[1].Split("(")[0].Trim();
            FunctionBlock function = new FunctionBlock();
            function.parent = parent;
            function.blockStartLineIndex = lineIndex;
            if (expression.Contains("override") || expression.Contains("virtual"))
                function.IsOverride = true;
            classObject.AllFunctions.Add(methodName, function);

            Assignment assignment;
            string[] vars = expression.Substring(expression.IndexOf('(') + 1, expression.IndexOf(')') - (expression.IndexOf('(') + 1)).Split(',');
            if (vars.Length > 0 && !string.IsNullOrWhiteSpace(vars[0]))
            {
                foreach (var item in vars)
                {
                    AddVariableToParent(function, item, out BaseType var, true);
                }
                assignment = new Assignment(function, "id", true);
            }
            else
            {
                assignment = new Assignment(function, "null", true);
            }
            parent = function;
            return assignment;
        }
        static Comand HandleFunctionCall(string expression, ICodeBlock parent)
        {
            string[] parameters = expression.Substring(expression.IndexOf('(') + 1, expression.IndexOf(')') - (expression.IndexOf('(') + 1)).Split(',');

            GetValueComand currentComand = null;
```

```
            if (parameters.Length > 0 && !string.IsNullOrWhiteSpace(parameters[0]))
            {
                foreach (var item in parameters)
                {
                    string mathPatern = @"(\+|\-|\*|\/)";
                    if (Regex.IsMatch(item, mathPatern))
                    {
                        List<Comand> mathExpression = CreateMathList(item, parent);
                        while (mathExpression.Count > 1)
                        {
                            mathExpression = HandleMath(parent, mathExpression);
                        }
                        if (currentComand != null)
                            currentComand = new GetValueComand(parent, item,
mathExpression[0], currentComand);
                        else
                            currentComand = new GetValueComand(parent, item,
mathExpression[0], null);
                    }
                    else
                    {
                        if (currentComand != null)
                            currentComand = new GetValueComand(parent, item, null,
currentComand);
                        else
                            currentComand = new GetValueComand(parent, item, null,
null);
                    }
                }
            }
            string[] strings = expression.Split('(')[0].Trim().Split(' ');
            FunctionComand comand = new FunctionComand(parent, currentComand,
strings[strings.Length - 1]);
            return comand;
        }
        static Comand HandleLogicCondition(string expression, ICodeBlock parent)
        {
            string patern = @"(else)";
            if (Regex.IsMatch(expression, patern) && !expression.Contains("if"))
            {
                return new Condition(parent, expression, false);
            }
                return new Condition(parent, "(" + expression.Split('(')[1], true);
```

```
        }
        static    List<Comand>    HandleMath(ICodeBlock    parent,    List<Comand>
mathExpression)
        {
            Comand first = null;
            Comand second = null;
            string mathOperator = null;
            List<string> currentExpression = new List<string>();
            int deepLevel = 0;
            int highestDeepLevel = 0;
            bool needToAddElement = true;
            int startIndex = 0;
            // find the deepest expression (x + (y + 1)) -> y + 1
            for (int i = 0; i < mathExpression.Count; i++)
            {
                string value = "null";
                if (mathExpression[i] is GetValueComand temp)
                {
                    value = temp.value;
                }

                if (value == "(")
                {
                    deepLevel++;
                    if (highestDeepLevel < deepLevel)
                    {
                        startIndex = i;
                        needToAddElement = true;
                        highestDeepLevel = deepLevel;
                        currentExpression.Clear();
                    }
                }
                if (value == ")")
                {
                    currentExpression.Add(value);
                    needToAddElement = false;
                    deepLevel--;
                }
                if (needToAddElement)
                    currentExpression.Add(value);
            }

            // check highest priority operation
```

```
int operationPriority = 0;
for (int i = 0; i < currentExpression.Count; i++)
{
    if (currentExpression[i] == "*" || currentExpression[i] == "/")
    {
        if (operationPriority < 1)
            operationPriority = 1;
    }
}
// create operation
for (int i = 0; i < currentExpression.Count; i++)
{
    if (operationPriority == 1)
    {
        if (currentExpression[i] == "*" || currentExpression[i] == "/")
        {
            startIndex = startIndex + i;
            first = mathExpression[startIndex - 1];
            second = mathExpression[startIndex + 1];
            mathOperator = currentExpression[i];
            break;
        }
    }
    else if (operationPriority == 0)
    {
        if (currentExpression[i] == "+" || currentExpression[i] == "-")
        {
            startIndex = startIndex + i;
            first = mathExpression[startIndex - 1];
            second = mathExpression[startIndex + 1];
            mathOperator = currentExpression[i];
            break;
        }
    }
}

Math mathComand = new Math(parent, first, second, mathOperator);
mathExpression[startIndex - 1] = mathComand;
mathExpression.RemoveAt(startIndex + 1);
mathExpression.RemoveAt(startIndex);

// resolve parentheses
List<int> numberOfTockens = new List<int>();
```

```
            List<int> indexesOfFirstParenthesis = new List<int>();
            deepLevel = 0;
            for (int i = 0; i < mathExpression.Count; i++)
            {
                string value = "null";
                if (mathExpression[i] is GetValueComand temp)
                {
                    value = temp.value;
                }

                if (value == "(")
                {

                    if (indexesOfFirstParenthesis.Count <= deepLevel)
                        indexesOfFirstParenthesis.Add(i);
                    else
                        indexesOfFirstParenthesis[deepLevel] = i;

                    deepLevel++;
                }
                if (value == ")")
                {
                    if (numberOfTockens[deepLevel] < 2)
                    {
                        mathExpression.RemoveAt(i);
                        mathExpression.RemoveAt(indexesOfFirstParenthesis[deepLevel
- 1]);

                        i -= 2;
                    }
                    deepLevel--;
                }
                if (value != "(" && value != ")")
                {
                    while (numberOfTockens.Count <= deepLevel)
                    {
                        numberOfTockens.Add(0);
                    }
                    numberOfTockens[deepLevel]++;
                }

            }
            return mathExpression;
        }
```

```
        static string GetParrentName(string line)
        {
            return line.Trim().Split(' ')[1];
        }
        static string AddVariableToParent(ICodeBlock parent, string code, out
BaseType var, bool isArguments)
        {
            string varDeclaration = code.Split('=')[0].Trim();
            string varType;
            string varName;
            var = null;
            if (varDeclaration.Split(' ').Length > 1)
            {
                varType = varDeclaration.Split(' ')[0];
                varName = varDeclaration.Split(' ')[1];
                switch (varType)
                {
                    case "float":
                        var = new BaseType(varName, VarType.Floating);
                        break;
                    case "string":
                        var = new BaseType(varName, VarType.String);
                        break;
                    default:
                        var = new ReferenceType(varName, varType, true);
                        break;
                }
                if (parent != null)
                {
                    if (parent.variables == null)
                    {
                        parent.variables = new Dictionary<string, BaseType>();
                        if (isArguments && ((FunctionBlock)parent).arguments == null)
                        {
                            ((FunctionBlock)parent).arguments = new
Dictionary<string, BaseType>();
                        }
                    }
                    if (isArguments)
                    {
                        ((FunctionBlock)parent).arguments.Add(varName, var);
                    }
                    parent.variables.Add(varName, var);
```

```
                }
            }
            else
            {
                varName = varDeclaration;
            }
            return varName;
        }
        static Comand HandleAssignment(string codeLine, ICodeBlock parent, Comand
prevComand)
        {
            string varName = AddVariableToParent(parent, codeLine, out BaseType var,
false);

            if (parent != null)
            {
                Assignment assignment = new Assignment(parent, varName, false,
prevComand);
                return assignment;
            }
            return null;
        }
        static Comand HandleReturn(string codeLine, ICodeBlock parent, Comand prev)
        {
            string value = codeLine.Replace("return", "").Trim();
            GetValueComand getValue = new GetValueComand(parent, value, null, prev);

            return new ReturnValueComand(parent, getValue);
        }
        static Comand GetValue(ICodeBlock parent, string expression)
        {
            GetValueComand comand = new GetValueComand(parent, expression.Trim());
            return comand;
        }
        static Comand HandleRefTypeCreation(ICodeBlock parent, string codeLine)
        {
            string varDeclaration = codeLine.Split('=')[0].Trim();
            string secondVarType = codeLine.Split("new")[1].Split('(')[0].Trim();
            string varType = varDeclaration.Split(' ')[0];
            string varName = varDeclaration.Split(' ')[1];
            RefTypeCreationComand comand = new RefTypeCreationComand(parent, varName,
varType, secondVarType);
            return comand;
```

```
    }}
```

Лістинг програми, що виконує функціонал консолі розробника, клас Console.

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using System.Linq;
using System.Text.RegularExpressions;
using System.Threading;
using System;

public class Console : MonoBehaviour
{
    public GameObject holder;
    public GameObject mainCamera;
    public GameObject consoleCamera;
    public TextAsset[] classes;
    public TMP_Dropdown dropdown;
    public TextMeshProUGUI numbersText;
    public TMP_InputField inputText;
    public TextMeshProUGUI consoleText;
    public ColorTheme theme;
    public Queue<ConsoleFunction> consoleComands = new Queue<ConsoleFunction>();

    Dictionary<string, string> classesCode = new Dictionary<string, string>();
    string selectedClassName = "";
    bool isTextChangedForcibly = false;
    int prevCurretPosition = 0;
    string nonColorCode = "";
    void Start()
    {
        AddCommonFunctions();
        dropdown.ClearOptions();
        List<string> classesList = new List<string>();
        foreach (var item in classes)
        {
            classesList.Add(item.name);
            classesCode.Add(item.name, item.text);
        }
        selectedClassName = classesList[0];
        inputText.text = classesCode[selectedClassName];
        dropdown.AddOptions(classesList);
        dropdown.value = 0;
        holder.SetActive(false);
    }
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Tab))
        {
            mainCamera.SetActive(!mainCamera.activeSelf);
            consoleCamera.SetActive(!consoleCamera.activeSelf);
            holder.SetActive(consoleCamera.activeSelf);
        }

        if (consoleComands.Count > 0)
        {
            ConsoleFunction func = consoleComands.Dequeue();
            func.func(func.vars);
        }
```

```
    }
    private void LateUpdate()
    {
        if (Input.GetKeyDown(KeyCode.LeftArrow))
        {
            if (prevCurretPosition > 0)
                prevCurretPosition--;
        }
        if (Input.GetKeyDown(KeyCode.RightArrow))
        {
            if (prevCurretPosition < nonColorCode.Length)
                prevCurretPosition++;
        }
        inputText.caretPosition = prevCurretPosition;
    }
    public void OnRunBtnClick()
    {
        List<ClassObject> classObjects = new List<ClassObject>();
        foreach (var item in classes)
        {
            ClassObject classVar =
CodeAnalizator.ReadCode(UncolorizeCode(classesCode[item.name]));
            classObjects.Add(classVar);
        }
        Interpreter interpreter = new Interpreter(classObjects);
        ClassObject mainClass = classObjects.First(x => x.Name == "Program");
        interpreter.startLineIndex =
mainClass.AllFunctions["Main"].blockStartLineIndex;
        interpreter.endLineIndex = mainClass.AllFunctions["Main"].blockEndLineIndex;
        Thread thread = new Thread(interpreter.RunLines);
        thread.Start();
    }
    public void OnTextChanged(string s)
    {
        HandleLineIndexes(s);
        if (!isTextChangedForcibly)
        {
            classesCode[selectedClassName] = inputText.text;
            prevCurretPosition = inputText.caretPosition;
            ColorizeCode(s);
        }
        isTextChangedForcibly = false;
    }
    void HandleLineIndexes(string code)
    {
        numbersText.text = "";
        string[] lines = code.Split('\n');
        for (int i = 1; i < lines.Length + 1; i++)
        {
            numbersText.text += i + " |\n";
        }
    }
    string FormatCode(string code)
    {
        int blockIndex = 0;
        string[] lines = code.Split('\n');
        string newCode = "";
        for (int i = 0; i < lines.Length; i++)
        {
            string spacing = new string('\t', blockIndex);
            for (int index = 0; index < lines[i].Length; index++)
            {
                if (lines[i][index] == '\t' || lines[i][index] == ' ')
                {
```

```
                    if (lines[i][index] == '\t')
                    {
                        lines[i] = lines[i].Remove(index, 1);
                    }
                }
                else
                {
                    break;
                }
            }
            lines[i] = spacing + lines[i];
            if (lines[i].Contains('{'))
                blockIndex++;
            if (lines[i].Contains('}'))
                blockIndex--;
            newCode += lines[i];
            if (i != lines.Length - 1)
                newCode += "\n";
        }
        return newCode;
    }
    string UncolorizeCode(string code)
    {
        string[] lines = code.Split('\n');
        Regex regex;

        string colorPattern = @"<color=#\w*>|</color>";
        regex = new Regex(colorPattern);
        return regex.Replace(code, "");
    }
    void ColorizeCode(string code)
    {
        string keyWordsPattern = @"\s*if\s*|\s*else\s*|\s*method\s*";
        string classKeywordPattern =
@"\s+class\W+|^class$|^class\W+|\s+class$|\s*interface\s*";
        string varKeywordPattern = @"\s*float\s*|\s*string\s*";

        string coloredCode = "";
        //<color=#ff0000ff>colorfully</color>
        string[] lines = code.Split('\n');
        Regex regex;
        MatchCollection matches;
        nonColorCode = "";
        for (int i = 0; i < lines.Length; i++)
        {
            string colorPattern = @"<color=#\w*>|</color>";
            regex = new Regex(colorPattern);
            string tempString = regex.Replace(lines[i], "");
            nonColorCode += tempString;

            regex = new Regex(keyWordsPattern);
            matches = regex.Matches(tempString);
            if (matches.Count > 0)
            {
                foreach (Match match in matches)
                {
                    regex = new Regex(match.Value);
                    tempString = regex.Replace(tempString,
$"<color=#{ColorTheme.GetHexColor(theme.keywords)}>{match.Value}</color>");
                }
            }

            regex = new Regex(classKeywordPattern);
            matches = regex.Matches(tempString);
```

```
            if (matches.Count > 0)
            {
                foreach (Match match in matches)
                {
                    regex = new Regex(match.Value);
                    tempString = regex.Replace(tempString,
$"<color=#{ColorTheme.GetHexColor(theme.classKeyword)}>{match.Value}</color>");
                }
            }
            regex = new Regex(varKeywordPattern);
            matches = regex.Matches(tempString);
            if (matches.Count > 0)
            {
                foreach (Match match in matches)
                {
                    regex = new Regex(match.Value);
                    tempString = regex.Replace(tempString,
$"<color=#{ColorTheme.GetHexColor(theme.varType)}>{match.Value}</color>");
                }
            }

            coloredCode += tempString;
            if (i != lines.Length - 1)
            {
                nonColorCode += "\n";
                coloredCode += "\n";
            }

        }
        if (coloredCode != code)
        {
            isTextChangedForcibly = true;
        }
        inputText.text = coloredCode;
    }

    public void PrintConsoleThisThread(List<BaseType> variables)
    {
        foreach (var item in variables)
        {
            consoleText.text += item.Value;
        }
        consoleText.text += "\n";
    }
    public void PrintConsole(List<BaseType> variables)
    {
        ConsoleFunction func = new ConsoleFunction(PrintConsoleThisThread, variables);
        consoleComands.Enqueue(func);
    }
    public void AddCommonFunctions()
    {
        Interpreter.globalComands.Add("print", PrintConsole);
    }
    public void OnDropdownChanged(int s)
    {
        string selectedOption = dropdown.options[s].text;
        classesCode[selectedClassName] = inputText.text;
        selectedClassName = selectedOption;
        inputText.text = classesCode[selectedOption];
    }
    public class ConsoleFunction
    {
        public Action<List<BaseType>> func;
        public List<BaseType> vars;
```

```
        public ConsoleFunction(Action<List<BaseType>> func, List<BaseType> vars)
        {
            this.vars = new List<BaseType>();
            this.func = func;
            foreach (var item in vars)
            {
                BaseType newVar = new BaseType(item.Name, item.Type);
                newVar.Value = item.Value;
                this.vars.Add(newVar);
            }
        }
    }

}
```

Лістинг класу Assignment.

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

public class Assignment : Comand
{
    string varName;
    Comand prev;
    bool isArguments;
    public Assignment(ICodeBlock parent, string varName, bool isArguments, Comand prev
= null) : base(parent)
    {
        this.varName = varName;
        this.prev = prev;
        this.isArguments = isArguments;
    }
    public override List<BaseType> Perform(ref ReferenceType context, List<BaseType>
variables = null)
    {
        BaseType var;
        if (variables != null)
        {
            if (isArguments)
            {
                int i = 0;
                foreach (var v in ((FunctionBlock)parent).arguments)
                {
                    CopyValues(parent.variables[v.Key], variables[i]);
                    i++;
                }
            }
            else
            {
                BaseType value = variables[0];
                var = FindVar(varName, parent, context);
                var.Value = value?.Value;
            }
        }
        else if (!isArguments)
        {
            if (varName == "null")
                return null;
            if (prev != null)
            {
                BaseType value = prev.Perform(ref context)[0];
                var = FindVar(varName, parent, context);
```

```
                CopyValues(var, value);
                variables = new List<BaseType>();
                variables.Add(var);
            }
        }
        return variables;
    }

}
```

Лістинг класу BaseType.

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

public enum VarType
{
    Floating,
    String,
    Reference
}
public class BaseType
{
    public string Name { get; set; }
    public VarType Type { get; set; }
    public string Value { get; set; }

    public BaseType(string name, VarType type)
    {
        Name = name;
        Type = type;
    }
}
```

Лістинг класу ClassBlock.

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

public class ClassBlock : ICodeBlock
{
    public ICodeBlock parent { get; set; } = null;
    public int blockStartLineIndex { get; set; }
    public int blockEndLineIndex { get; set; }
    public Dictionary<string, BaseType> variables { get; set; }

    public ClassBlock()
    {
    }

    public ClassBlock(int blockStartLineIndex, int blockEndLineIndex, ICodeBlock
parent = null)
    {
        this.parent = parent;
        this.blockStartLineIndex = blockStartLineIndex;
```

```
        this.blockEndLineIndex = blockEndLineIndex;
    }
}
```

Лістинг класу ClassObject.

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

public class ClassObject
{
    public ICodeBlock classBlock { get; set; }
    public string Name { get; set; }
    public bool HasParent { get; set; }
    public string ParentName { get; set; }
    public List<Line> ReadyToUseCode { get; set; }
    public Dictionary<string, FunctionBlock> AllFunctions { get; set; }

    public ClassObject(string name, ICodeBlock classBlock)
    {
        AllFunctions = new Dictionary<string, FunctionBlock>();
        Name = name;
        this.classBlock = classBlock;
        HasParent = false;
    }
}
```

Лістинг класу ColorTheme.

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

[CreateAssetMenu(menuName = "Color Theme", fileName = "Color Theme")]
public class ColorTheme : ScriptableObject
{
    public Color varType;
    public Color classKeyword;
    public Color keywords;

    public static string GetHexColor(Color color)
    {
        string hexColor = ToHex(Mathf.Lerp(0, 255, color.r)) + ToHex(Mathf.Lerp(0,
255, color.g)) + ToHex(Mathf.Lerp(0, 255, color.b)) + ToHex(Mathf.Lerp(0, 255, 1));
        return hexColor;
    }
    static string ToHex(float value)
    {
        int intPart = (int)(value / 16);
        int floatPart = (int)(value % 16);
        string result = "";

        if (value < 16)
        {
            result += "0";
        }
```

```
        result += GetHexValue(intPart);
        if (result.Length < 2)
            result += GetHexValue(floatPart);
        return result;
    }
    static string GetHexValue(int value)
    {
        switch (value)
        {
            case 10:
                return "A";
            case 11:
                return "B";
            case 12:
                return "C";
            case 13:
                return "D";
            case 14:
                return "E";
            case 15:
                return "F";
            default:
                return value.ToString();
        }
    }
}
```

Лістинг класу Comand.

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

public abstract class Comand
{
    public ICodeBlock parent;
    protected Comand(ICodeBlock parent)
    {
        this.parent = parent;
    }
    public abstract List<BaseType> Perform(ref ReferenceType context, List<BaseType>
variables = null);
    public BaseType FindVar(string varName, ICodeBlock parent, ReferenceType context)
    {
        if (context != null)
        {
            if (context.variables.ContainsKey(varName))
                return context.variables[varName];
            else if (context.Parent != null)
            {
                if (context.Parent.variables.ContainsKey(varName))
                    return context.Parent.variables[varName];
            }
        }
        ICodeBlock currentparent = parent;
        while (currentparent != null)
        {
            if (currentparent.variables != null && !varName.Contains("."))
            {
                if (currentparent.variables.ContainsKey(varName))
                    return currentparent.variables[varName];
```

```
            }
            else
            {
                string[] names = varName.Split('.');
                if (currentparent.variables != null &&
currentparent.variables.ContainsKey(names[0]))
                {
                    ReferenceType refType =
(ReferenceType)currentparent.variables[names[0]];
                    if (refType.variables.ContainsKey(names[1]))
                        return refType.variables[names[1]];
                    else if (refType.Parent != null)
                    {
                        return refType.Parent.variables[names[1]];
                    }
                    else if (refType.Child != null) // тимчасово
                    {
                        return refType.Child.variables[names[1]];
                    }
                }
            }
            currentparent = currentparent.parent;
        }

        return null;
    }
    public void CopyValues(BaseType to, BaseType from)
    {
        if (from is ReferenceType refType)
        {
            ((ReferenceType)to).variables = refType.variables;
            ((ReferenceType)to).Parent = refType.Parent;
            ((ReferenceType)to).Child = refType.Child;
            ((ReferenceType)to).referenceType = refType.referenceType;
            ((ReferenceType)to).IsNull = false;
        }
        else if (from.Value == "null")
        {
            ((ReferenceType)to).IsNull = true;
        }
        else
        {
            to.Value = from.Value;
        }
    }
}
```

Лістинг класу Condition.

```
using System.Collections;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using UnityEngine;
using System.Linq;

public class Condition : Comand
{
    bool isNeedCalculation = true;
    string expression;
    public Condition(ICodeBlock parent, string expression, bool isNeedCalculation) :
base(parent)
```

```
    {
        this.expression = expression;
        this.isNeedCalculation = isNeedCalculation;
    }

    public override List<BaseType> Perform(ref ReferenceType context, List<BaseType>
variables = null)
    {
        if (!isNeedCalculation)
            return new List<BaseType>();

        List<string> list = new List<string>();
        string[] array = expression.Trim().Split(" ");
        string someString = "";
        bool switcher = false;
        for (int i = 0; i < array.Length; i++)
        {
            if (array[i].StartsWith("\"") && array[i].EndsWith("\""))
            {
                list.Add(array[i]);
                continue;
            }
            if (array[i].Contains("\""))
            {
                switcher = !switcher;
            }
            if (switcher)
            {
                someString += array[i] + " ";
                continue;
            }
            if (!switcher && someString != "")
            {
                someString += array[i];
                list.Add(someString);
                someString = "";
                continue;
            }
            list.Add(array[i]);
        }
        if (list.Count == 0)
            return null;
        LogicOperator[] logicOperators = new LogicOperator[8] { new
LogicOperator("==", 1), new LogicOperator("!=", 1), new LogicOperator(">=", 1), new
LogicOperator("<=", 1), new LogicOperator(">", 1), new LogicOperator("<", 1), new
LogicOperator("&", 2), new LogicOperator("|", 2) };
        List<LogicOperator> operators = new List<LogicOperator>();
        List<string> operands = new List<string>();
        Regex operatorsPatern = new Regex(@"(==|!=|>=|<=|>|<|&|\|)");
        MatchCollection matches = operatorsPatern.Matches(expression);
        foreach (Match match in matches)
        {
            operators.Add(logicOperators.First(x => x.name == match.Value));
        }

        for (int i = 0; i < list.Count; i++)
        {
            if (list[i].Contains("\""))
            {
                operands.Add(list[i].Substring(1, list[i].Length - 2));
                continue;
            }
            if (float.TryParse(list[i], out float result))
                operands.Add(list[i]);
```

```
            else
            {
                BaseType temp = FindVar(list[i], parent, context);
                if (temp is ReferenceType type)
                {
                    if (type.IsNull)
                    {
                        operands.Add("null");
                    }
                    else
                    {
                        operands.Add("notnull");
                    }
                }
                else if (temp != null)
                    operands.Add(temp.Value);
            }
        }

        List<LogicOperand> operandsT2 = new List<LogicOperand>();
        List<LogicOperator> operatorsT2 = new List<LogicOperator>();
        for (int i = 0; i < operators.Count; i++)
        {
            if (operators[i].priority == 1)
            {
                LogicOperand op = new LogicOperand(Calculate(operators[i].name,
operands[i], operands[i + 1]));
                operandsT2.Add(op);
            }
            if (operators[i].priority == 2)
            {
                operatorsT2.Add(operators[i]);
            }
        }

        int index;
        for (int k = 0; k < 2; k++)
        {
            index = 0;
            for (int i = 0; i < operatorsT2.Count; i++)
            {
                if (k == 0)
                {
                    if (operatorsT2[i].name == "&")
                    {
                        if (operandsT2[index].isTrue && operandsT2[index + 1].isTrue)
                        {
                            operandsT2[index].isTrue = true;
                        }
                        else
                        {
                            operandsT2[index].isTrue = false;
                        }
                        operandsT2.RemoveAt(index + 1);
                        operatorsT2.RemoveAt(i);
                    }
                    else
                        index++;
                }
                else
                {
                    if (operandsT2[index].isTrue || operandsT2[index + 1].isTrue)
                    {
                        operandsT2[index].isTrue = true;
```

```
                    }
                    else
                    {
                        operandsT2[index].isTrue = false;
                    }
                    operandsT2.RemoveAt(index + 1);
                }
            }
        }
        if (operandsT2[0].isTrue)
            return new List<BaseType>();
        return null;
    }

    bool Calculate(string op, string left, string right)
    {
        if (left == "null" || right == "null" || left == "notnull" || right ==
"notnull" || !float.TryParse(left, out float result))
        {
            if (op == "==")
                return left == right;
            else
                return left != right;
        }
        float leftF = float.Parse(left);
        float rightF = float.Parse(right);
        switch (op)
        {
            case "==":
                return leftF == rightF;
            case "!=":
                return leftF != rightF;
            case ">=":
                return leftF >= rightF;
            case "<=":
                return leftF <= rightF;
            case ">":
                return leftF > rightF;
            case "<":
                return leftF < rightF;
            default:
                return false;
        }
    }
}
abstract class LogicToken
{

}
class LogicOperand : LogicToken
{
    public bool isTrue { get; set; }

    public LogicOperand(bool isTrue)
    {
        this.isTrue = isTrue;
    }
}
class LogicOperator : LogicToken
{
    public string name { get; private set; }
    public int priority { get; private set; }
    public LogicOperator(string name, int priority)
    {
```

```
        this.name = name;
        this.priority = priority;
    }
}
```

Лістинг класу ConditionBlock.

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

public class ConditionBlock : ICodeBlock
{
    public ICodeBlock parent { get; set; }
    public int blockStartLineIndex { get; set; }
    public int blockEndLineIndex { get; set; }
    public Dictionary<string, BaseType> variables { get; set; }
    public int chainBlockEndLineIndex { get; set; }
    public ConditionBlock prevChainElement { get; set; } = null;
}
```

Лістинг класу FunctionBlock.

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

public class FunctionBlock : ICodeBlock
{
    public ICodeBlock parent { get; set; } = null;
    public int blockStartLineIndex { get; set; }
    public int blockEndLineIndex { get; set; }
    public Dictionary<string, BaseType> variables { get; set; }
    public Dictionary<string, BaseType> arguments { get; set; }
    public BaseType ReturnValue { get; set; } = null;
    public bool IsOverride { get; set; } = false;
}
```

Лістинг класу FunctionComand.

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

public class FunctionComand : Comand
{
    Comand parameters;
    string name;
    string shortName;
    public ClassObject context { get; private set; }
    public FunctionComand(ICodeBlock parent, Comand parameters, string name) :
base(parent)
    {
        this.parameters = parameters;
```

```
        this.name = name;
        shortName = name;
    }

    public override List<BaseType> Perform(ref ReferenceType context, List<BaseType>
variables = null)
    {
        if (name.Contains('.'))
        {
            string varName = name.Split('.')[0];
            shortName = name.Split('.')[1];
            if (varName == "base")
            {
                context = context.Parent;
                foreach (var item in Interpreter.Classes)
                {
                    if (item.Name == context.referenceType)
                    {
                        this.context = item;
                        break;
                    }
                }
            }
            else
            {
                ReferenceType var = (ReferenceType)FindVar(varName, parent, context);
                context = var;
                foreach (var item in Interpreter.Classes)
                {
                    if (item.Name == context.referenceType)
                    {
                        if (item.AllFunctions[shortName].IsOverride && var.Child !=
null)
                        {
                            context = var.Child;
                        }
                        break;
                    }
                }

                foreach (var item in Interpreter.Classes)
                {
                    if (item.Name == context.referenceType)
                    {
                        if (item.classBlock.variables != null)
                        {
                            foreach (var variable in item.classBlock.variables.Keys)
                            {
                                if (var.variables.ContainsKey(variable))
                                    item.classBlock.variables[variable].Value =
var.variables[variable].Value;
                                else
                                    item.classBlock.variables[variable].Value =
var.Child.variables[variable].Value;
                            }
                        }
                        this.context = item;
                        break;
                    }
                }
            }

        }
        return parameters?.Perform(ref context);
```

```
    }
    public string GetName()
    {
        return shortName;
    }
}
```

Лістинг класу GetValueComand.

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

public class GetValueComand : Comand
{
    public string value { get; private set; }
    public Comand prev { get; private set; }
    private Comand comandValue;
    public GetValueComand(ICodeBlock parent, string value, Comand comandValue = null,
Comand prev = null) : base(parent)
    {
        this.value = value.Trim();
        this.prev = prev;
        this.comandValue = comandValue;
    }

    public override List<BaseType> Perform(ref ReferenceType context, List<BaseType>
variables = null)
    {
        if (comandValue != null)
        {
            value = comandValue.Perform(ref context)[0].Value;
        }

        List<BaseType> resultList = new List<BaseType>();
        if (prev != null)
        {
            resultList = prev.Perform(ref context);
        }
        BaseType type;
        if (float.TryParse(value, out float result))
        {
            type = new BaseType("id", VarType.Floating);
            type.Value = result.ToString();
        }
        else
        {
            if (value.Contains("\""))
            {
                type = new BaseType("id", VarType.String);
                type.Value = value.Substring(1, value.Length - 2);
            }
            else
            {
                type = FindVar(value, parent, context);
            }
        }
        resultList.Add(type);
        return resultList;
    }
```

```
}
```

Лістинг класу ICodeBlock.

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

public interface ICodeBlock
{
    public ICodeBlock parent { get; set; }
    public int blockStartLineIndex { get; set; }
    public int blockEndLineIndex { get; set; }
    public Dictionary<string, BaseType> variables { get; set; }

}
```

Лістинг класу Line.

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


public class Line
{
    public List<Comand> comands { get; private set; }
    public string codeLine { get; private set; }
    public int currentIndex { get; private set; }
    public ICodeBlock parent { get; private set; }

    public Line(string line, int currentIndex, List<Comand> comands, ICodeBlock
parent)
    {
        codeLine = line;
        this.comands = comands;
        this.currentIndex = currentIndex;
        this.parent = parent;
};

}
```

Лістинг класу Math.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Text.RegularExpressions;
using System.Linq;

public class Math : Comand
{
    string mathOperator;
```

```
    Comand first;
    Comand second;

    //string expression;
    //string digitPatern = @"(\d)";
    //string letterPatern = @"(\w)";
    public Math(ICodeBlock parent, Comand first, Comand second, string mathOperator) :
base(parent)
    {
        this.first = first;
        this.second = second;
        this.mathOperator = mathOperator;
    }
    public override List<BaseType> Perform(ref ReferenceType context, List<BaseType>
variables = null)
    {
        BaseType var1 = first.Perform(ref context)[0];
        BaseType var2 = second.Perform(ref context)[0];
        float value1 = float.Parse(var1.Value);
        float value2 = float.Parse(var2.Value);

        BaseType result = new BaseType("id", VarType.Floating);
        switch (mathOperator)
        {
            case "+":
                result.Value = (value1 + value2).ToString();
                break;
            case "-":
                result.Value = (value1 - value2).ToString();
                break;
            case "*":
                result.Value = (value1 * value2).ToString();
                break;
            case "/":
                result.Value = (value1 / value2).ToString();
                break;
            default:
                break;
        }

        return new List<BaseType>(new BaseType[] { result });

    }
}
```

Додаток Г

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ ДЛЯ СТВОРЕННЯ ІНТЕРАКТИВНОГО СЕРЕДОВИЩА ВИВЧЕННЯ ШАБЛОНІВ ПРОЄКТУВАННЯ GOF

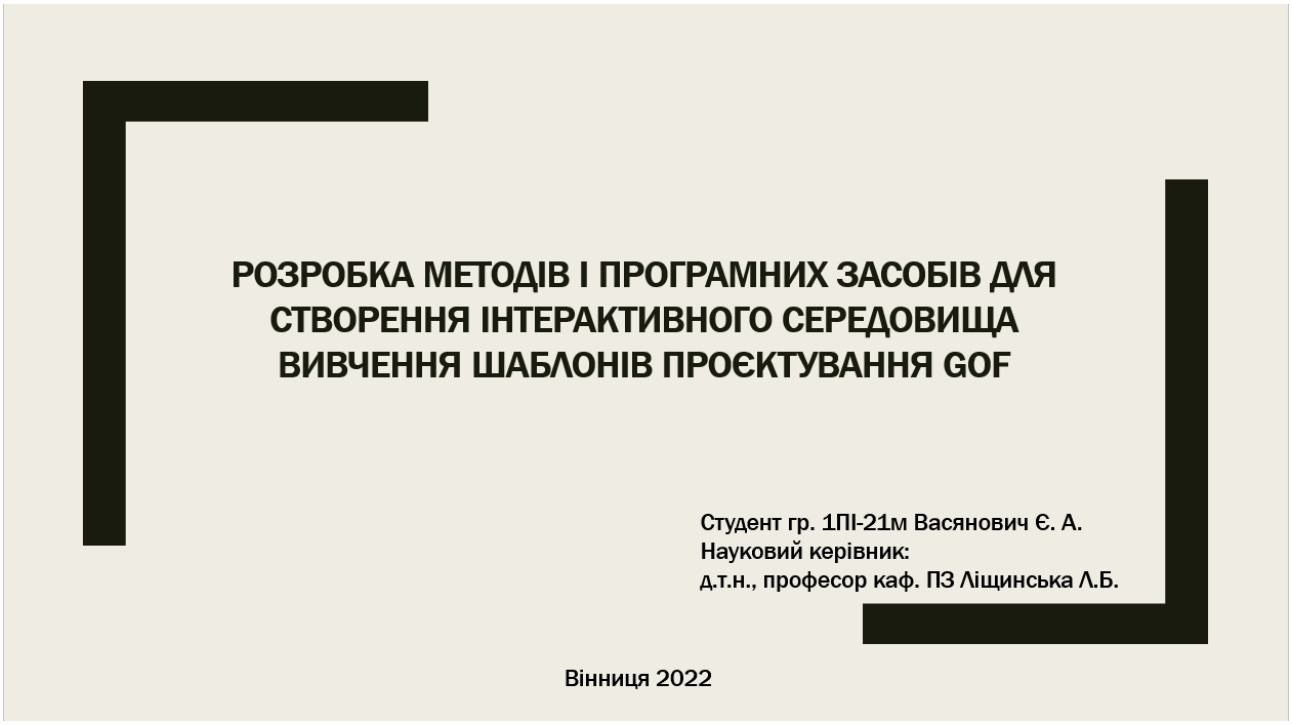


Рисунок Г.1 – Слайд презентації №1

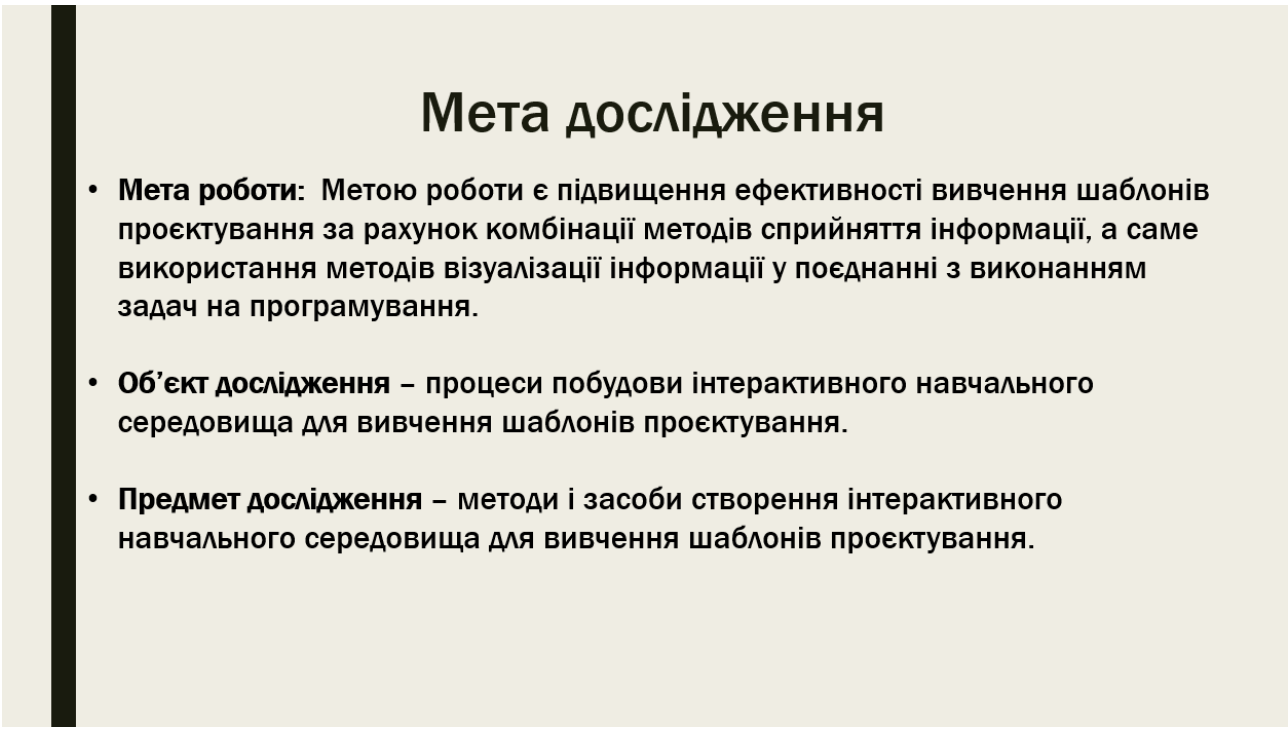


Рисунок Г.2 – Слайд презентації №2

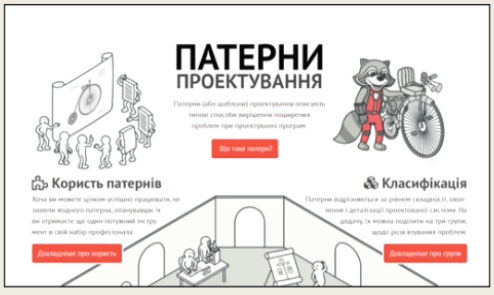
Завдання дослідження

- провести аналіз методів вивчення складної та комплексної інформації з метою покращення ефективності вивчення шаблонів проєктування;
- проаналізувати основні аналоги програмних засобів вивчення шаблонів проєктування;
- удосконалити метод вивчення шаблонів проєктування;
- вдосконалити метод створення трансляторів для мов програмування;
- запропонувати метод текстурування з метою підвищення ефективності зберігання графічної інформації;
- розробити псевдомову програмування для виконання практичних задач;
- розробити інтерактивне середовище з візуалізацією для вивчення шаблонів;
- провести тестування розробленого додатку.

Рисунок Г.3 – Слайд презентації №3

Аналоги засобів вивчення шаблонів проєктування

- Refactoring guru
- Metanit
- GoF Design Patterns
- Design Patterns in C#



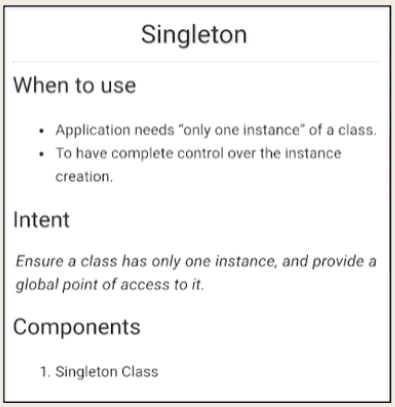
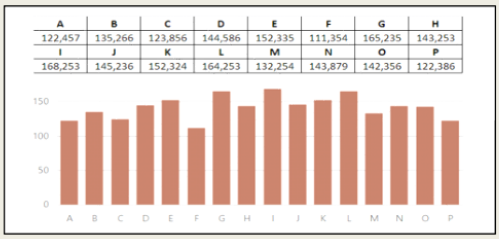
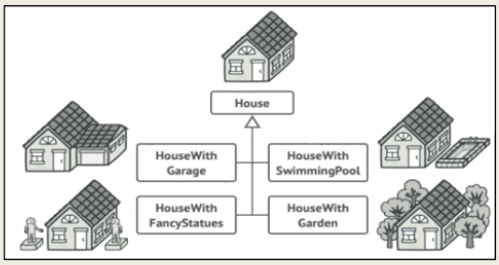


Рисунок Г.4 – Слайд презентації №4

Огляд методів подання складної інформації





Методи вивчення інформації:

- Лекції.
- Читання.
- Відео/аудіо матеріали.
- Практика через дію.
- Навчання інших, застосування практики.

Рисунок Г.5 – Слайд презентації №5

Порівняльний аналіз аналогів

Критерій	Refactoring guru	Metanit	GoF Design Patterns	Design Patterns in C#	Власний додаток
Графічна складова	4/5	2/5	1/5	1/5	5/5
Текстова складова	3/3	2/3	2/3	1/3	2/3
Безкоштовна ліцензія	3/3	3/3	2/3	2/3	3/3
Підключення до мережі	1/2	1/2	2/2	2/2	2/2
Підтримка псевдомови	3/3	1/3	1/3	1/3	3/3
Загальна оцінка ефективності	14/16	9/16	8/16	7/16	15/16

Рисунок Г.6 – Слайд презентації №6

Рисунок Г.7 – Слайд презентації №7

Рисунок Г.8 – Слайд презентації №8

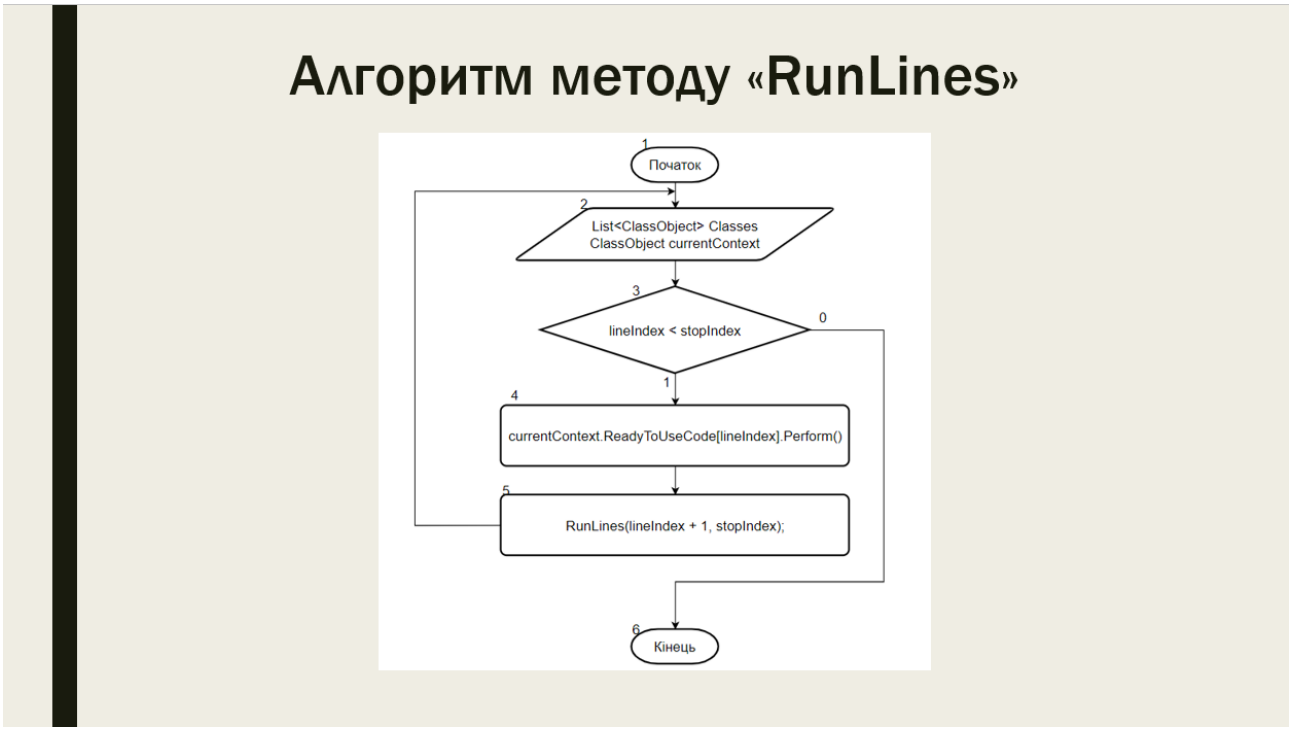


Рисунок Г.9 – Слайд презентації №9

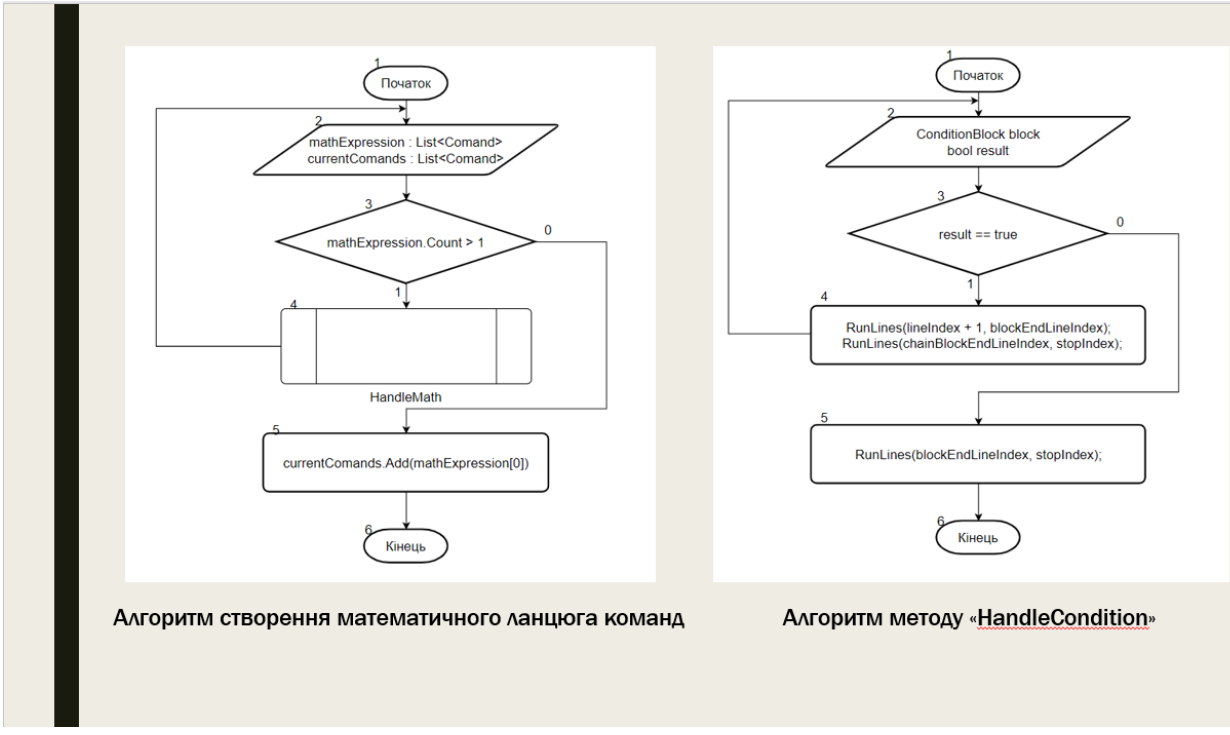


Рисунок Г.10 – Слайд презентації №10

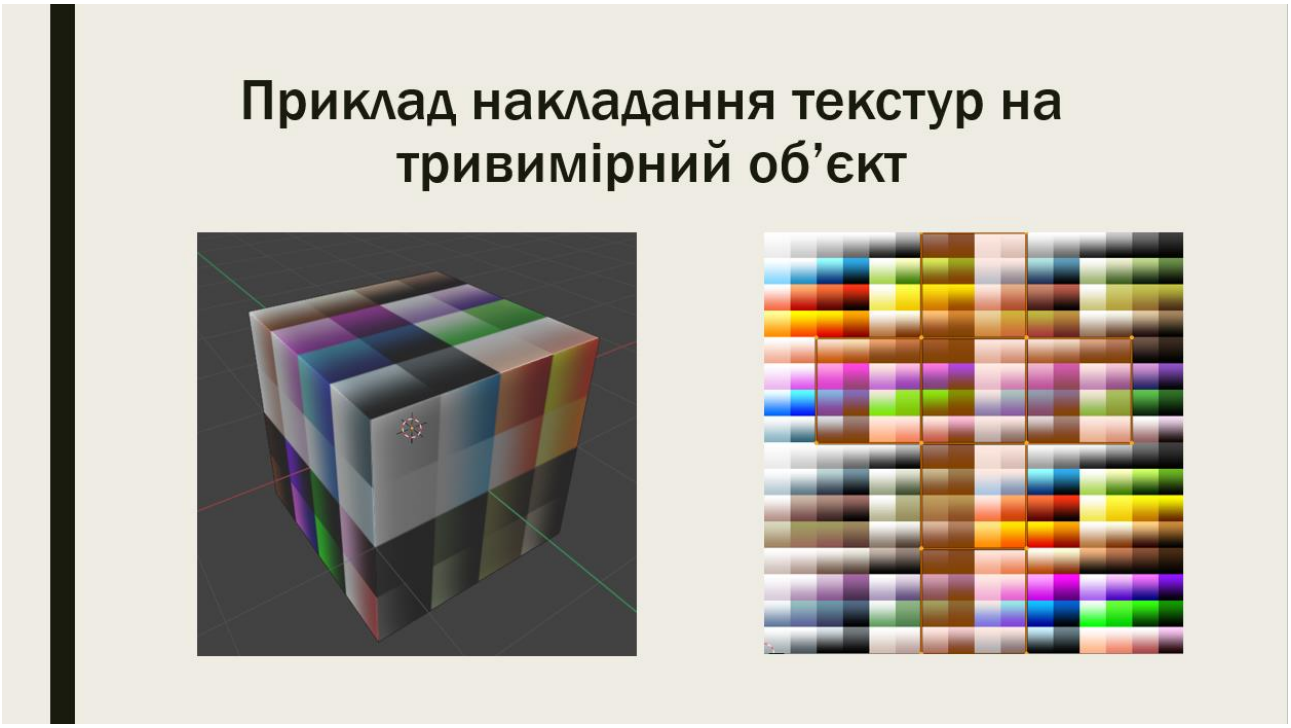


Рисунок Г.11 – Слайд презентації №11

Рисунок Г.12 – Слайд презентації №12

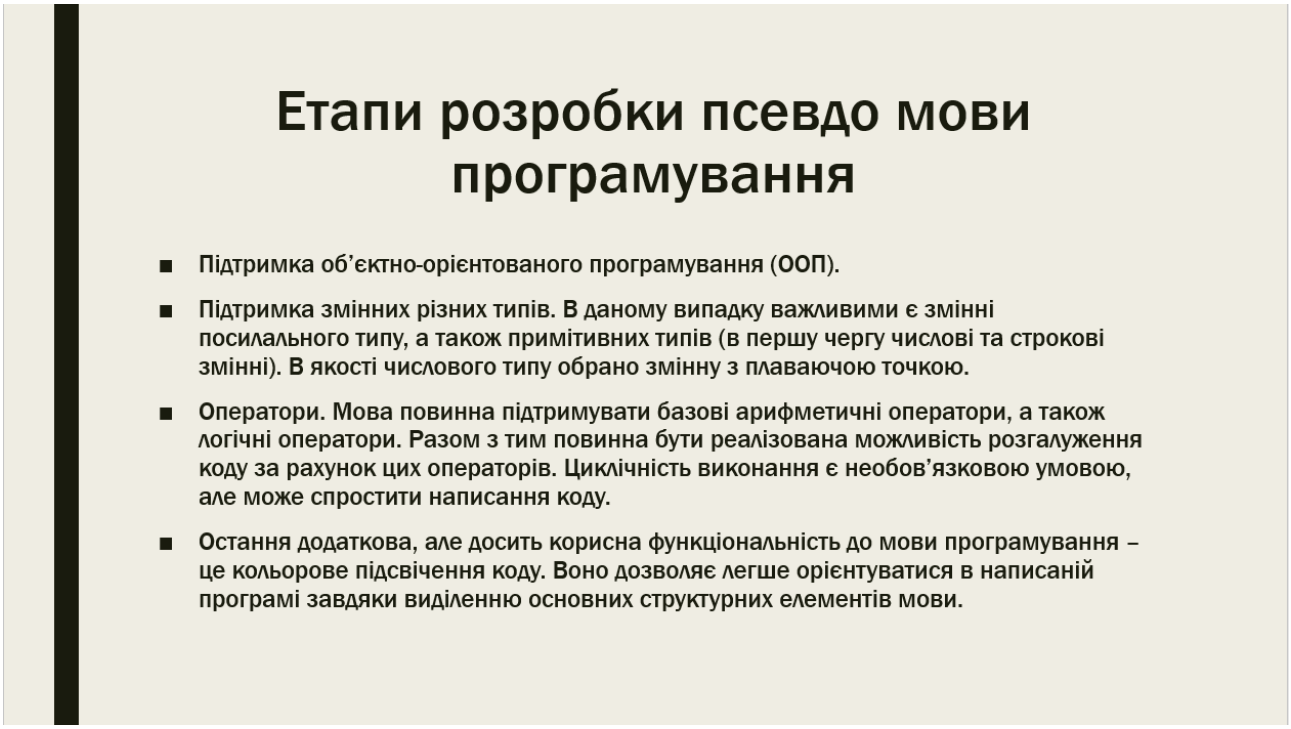


Рисунок Г.13 – Слайд презентації №13

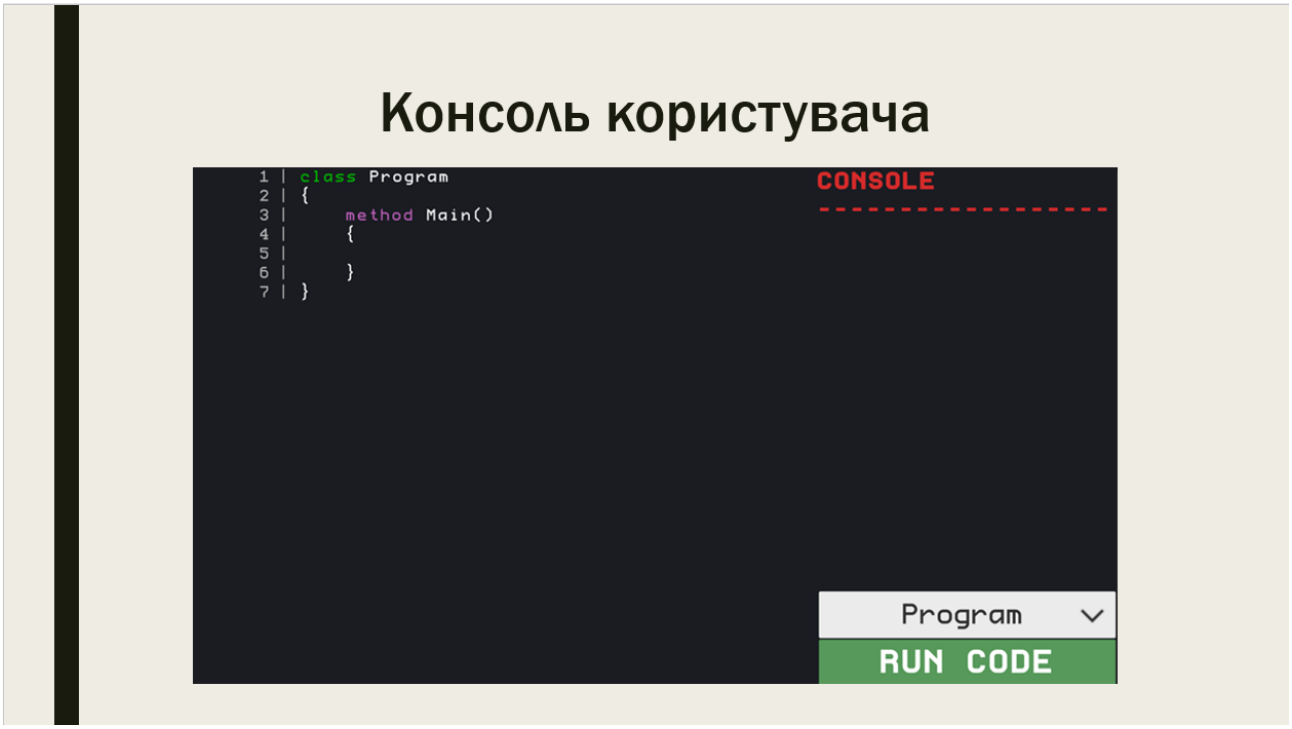


Рисунок Г.14 – Слайд презентації №14

Приклад реалізації шаблону «ланцюжок обов'язків»

Рисунок Г.15 – Слайд презентації №15

Наукова новизна отриманих результатів

- Удосконалено метод вивчення шаблонів проєктування на основі інтерактивного середовища, що дозволить підвищити ефективність їх вивчення, за рахунок використання засобів візуалізації, а також інтегрованого середовища розробки для практичного тренування навичок створення шаблонів.
- Запропоновано комбінований метод трансляції мови програмування, що використовує переваги компілятора та інтерпретатора і дозволяє уникнути зайвих трансляції програмного коду, при цьому не зберігаючи виконуваний код в окремий файл.
- Запропоновано метод зберігання текстур для тривимірних низько-полігональних моделей на основі однієї основної текстури та декількох додаткових мап, що дозволяє заощадити використання пам'яті додатком на 25%.

Рисунок Г.16 – Слайд презентації №16

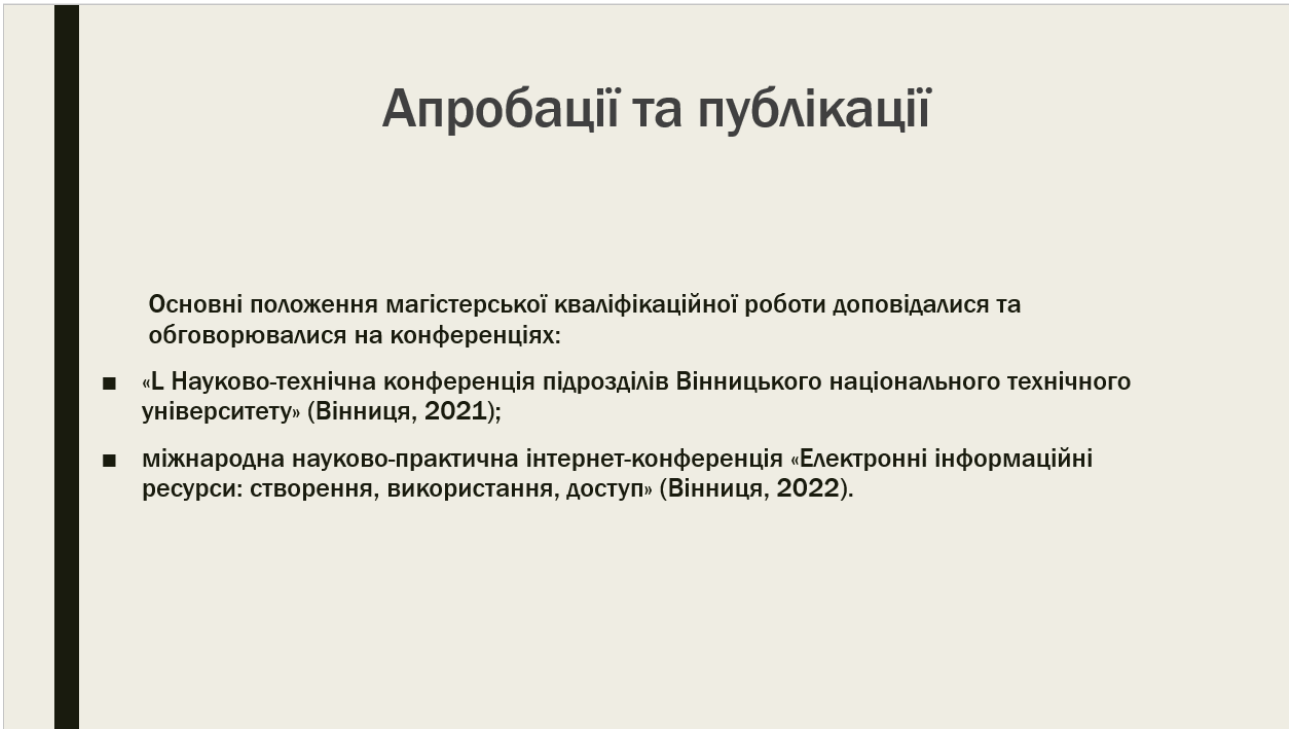


Рисунок Г.17 – Слайд презентації №17

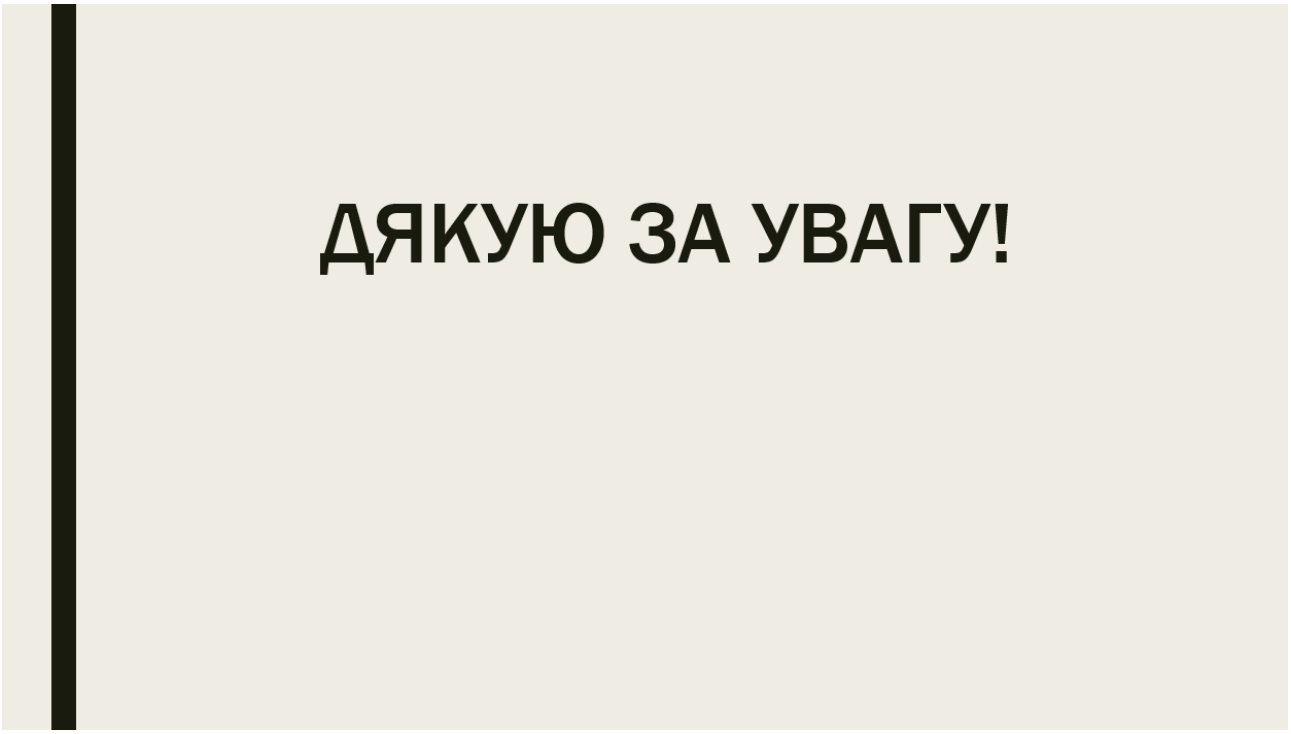


Рисунок Г.18 – Слайд презентації №18