**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені Ігоря Сікорського»**

**Факультет прикладної математики**

**Кафедра прикладної математики**

«До захисту допущено»

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_ О. Р. Чертов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 р.

**Дипломна робота**

**на здобуття ступеня бакалавра**

з напряму підготовки 6.040301 «Прикладна математика» на тему: «Автоматизована система генерації емоційно забарвлених реплік»

Виконав: студент IV курсу, групи КМ-61

|  |  |  |
| --- | --- | --- |
|  | Мозговий Микита Андрійович | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Керівник | асистент  Громова Вікторія Вікторівна | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Консультант зі спеціальних питань | асистент  Борисенко П. Б. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Консультант із нормоконтролю | старший викладач  Мальчиков В. В. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Рецензент | ГречкоАнастасія Валеріївна | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Засвідчую, що в цій дипломній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент \_\_\_\_\_\_\_\_\_\_\_\_

Київ — 2020**Національний технічний університет України  
«Київський політехнічний інститут  
імені Ігоря Сікорського»**

Факультет прикладної математики

Кафедра прикладної математики

Рівень вищої освіти — перший (бакалаврський)

Напрям підготовки 6.040301 «Прикладна математика»

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_\_\_\_ О. Р. Чертов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 р.

**ЗАВДАННЯ**

**на дипломну роботу студенту**

Мозговому Микиті Андрійовичу

1. Тема роботи: «Автоматизована система генерації емоційно забарвлених реплік»,

керівник роботи Громова Вікторія Вікторівна, асистент,

затверджені наказом по університету від .

2. Термін подання студентом роботи:

3. Вихідні дані до роботи: розроблена система повинна мати можливість масштабування на інші задачі генерації.

4. Зміст роботи: виконати аналіз існуючих методів розв’язання задачі, вибрати метод генерації тексту, спроектувати автоматизовану систему генерації емоційно забарвлених реплік, здійснити програмну реалізацію розробленої системи, провести тестування розробленої системи.

5. Перелік ілюстративного матеріалу: архітектурні графи ланцюгів Маркова, блок-схеми розроблених алгоритмів, схема взаємодії модулів системи, знімки екранних.

6. Консультанти розділів роботи:

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
| завдання видав | завдання прийняв |
| Розділ 4. Програмне забезпечення | Борисенко П. Б., асистент |  |  |

7. Дата видачі завдання:

Календарний план

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітка |
| 1 | Огляд літературних джерел та аналіз наявних програм та систем. | 19.11.2019 |  |
| 3 | Висування загальних вимог до програмного забезпечення та математичної моделі. | 25.12.2019 |  |
| 4 | Вибір моделі та обґрунтування вибору на основі якісних оцінок моделей. | 02.01.2020 |  |
| 5 | Проектування та створення математичної моделі на основі вибраного методу. | 03.02.2020 |  |
| 6 | Проектування та створення ПЗ на основі вибраного методу. | 04.04.2020 |  |
| 7 | Формування пояснювальної записки. | 31.06.2020 |  |

АНОТАЦIЯ

Дипломну роботу присвячено розробці математичних та програмних засобів для розв’язання задачі генерації емоційно забарвлених реплік.

У роботі проведено аналіз існуючих рішень указаної задачі, виконано їх порівняння з погляду використання ресурсів та адаптації до різних умов генерації.

Для розв’язання задачі в роботі використано метод ланцюгів Маркова. Користувач описує схему ланцюга Маркова, де станом може бути як текст, так і інший ланцюг Маркова. Розроблено автоматизовану систему, що реалізує обраний метод. У ході її тестування було виявлено, що система досить ефективна с точки зору витрати ресурсів .

Роботу виконано на 40 аркушах, вона містить 3 додатки та перелік посилань на використані джерела з 9 найменувань. У роботі наведено 12 рисунків та 3 таблиці.

Ключові слова: емоція, генерація тексту, ланцюги Маркова.

ABSTRACT

This thesis deals with developing of the mathematical and software tools for solving the problem of generating emotional text.

In the thesis, the comparative analysis of existing solutions in terms of the resource consumption and adaptability to generating different kinds of texts.

Markov chain approach is used to solve the task. User creates scheme for Markov chain, in which each node represents either statement, or another Markov chain.

The automated system implementing the chosen method is developed. During testing the system, it was found that the system has good performance .

The thesis is presented in 40 pages. It contains 3 appendixes and bibliography of 9 references. 12 figures and 3 tables are given in the thesis.

Keywords: emotion, text generating, Markov chain.

Зміст

[Вступ 9](#_Toc41753821)

[1 Постановка задачі 10](#_Toc41753822)

[2 Аналіз існуючих методів генерації 11](#_Toc41753823)

[2.1 Видiлення критеріїв оцінки методів 11](#_Toc41753824)

[2.2 Математичні методи 12](#_Toc41753825)

[2.2.1 Лінгвістичні моделі 12](#_Toc41753826)

[2.2.2 Моделювання з блоків 14](#_Toc41753827)

[2.2.3 Використання міток або тегів 15](#_Toc41753828)

[2.2.4 Генерація з використанням ієрархій 16](#_Toc41753829)

[2.2.5 Марківські моделі 17](#_Toc41753830)

[2.3 Висновки 18](#_Toc41753831)

[3. Математичне забезпечення 20](#_Toc41753832)

[3.1 Принцип моделювання ланцюгами Маркова 20](#_Toc41753833)

[3.2 Формування моделі 24](#_Toc41753834)

[3.3 Висновки 26](#_Toc41753835)

[4. Програмне забезпечення 27](#_Toc41753836)

[4.1 Архітектура програми 27](#_Toc41753837)

[4.2 Опис розроблених алгоритмів 29](#_Toc41753838)

[4.3 Формат вхiдних даних 33](#_Toc41753839)

[4.3.1 Формат схеми ланцюга 33](#_Toc41753840)

[4.3.2 Формат файлу лінкування 33](#_Toc41753841)

[4.4 Формат результуючих даних 34](#_Toc41753842)

[4.5 Результати тестування 35](#_Toc41753843)

[4.6 Висновки 37](#_Toc41753844)

[Висновки 39](#_Toc41753845)

[Перелiк посилань 40](#_Toc41753846)

[Додаток А 41](#_Toc41753847)

[Iлюстративний матерiал 41](#_Toc41753848)

# Вступ

У наш час попиту набувають системи обробки великої кількості даних. Разом із тим, все більше уваги приділяється суміжній сфері генерації даних за певним критерієм.

Ця сфера допомагає нам не лише генерувати реалістичні текстові приклади, а й створювати нову корисну інформацію. Інформаційні системи, що здатні інтерполювати велику кількість інформації здатні змінити наш підхід до перегляду фільмів, пошукових запитів і генерації нової інформації. Зараз більша кількість інформації генерується людиною, що звужує можливість аналізу і збільшує можливість повторення. Ця проблема стосується як малювання, яке із приходом зручних засобів навчання, створення та розповсюдження контенту, змінює фокус у напрям сюжетів у малюнках, так і науки, яка зараз вирішує проблему псевдо позитивних результатів, яка пов’язана не тільки з великою кількістю досліджень, але і з не публікацією усіх досліджень та стандартним критерієм *p < 0.05*. Можливість генерації та аналізу згенерованих даних дозволяє нам знайти можливі шляхи за якими пройшла генерація та по новому подивитися на старі критерії.

Вже зараз ми маємо можливість прослухати згенеровану музику, або власноруч згенерувати її онлайн. Все частіше з’являються нові алгоритми у цій сфері. Проте, вони мають обмеження, або потребують повного перерахунку моделі чи великої кількості ресурсів

У даній роботі, я проаналізував існуючі рішення для генерації текстових даних. Оскільки ця тема найчастіше зустрічається у ігровому просторі, більшість із прикладів будуть пов’язані із іграми. У даній сфері існує попит на генерацію текстів, які відрізняються за розміром, стилем та інтерактивністю. Я вважаю, що вирішення цієї задачі створює обмеження у витратах ресурсів та можливості створювати нову модель при кожному запуску.

# 1 Постановка задачі

Метою даної дипломної роботи є створення математичного та програмного забезпечення для генерації емоційно забарвленого тексту в умовах ігрового простору.

Множина емоцій не фіксована, але у наведених прикладах розглядаються наступні емоції:

а)щастя;

б)сум;

в)злість.

При розробленні відповідного забезпечення я планую розв’язати наступні завдання:

а) проведення порівняльного аналізу існуючих методів генерації тексту;

б)вибір реалізація одного з методів генерації;

в) розробка програмного забезпечення на базі вибраного математичного методу.

Реалізована система має задовольняти такі вимоги:

а) мати можливість адаптуватися до різних умов генерації, без перебудови моделі;

б) бути достатньо оптимальною с точки зору витрати ресурсів, для інтеграції у таку навантажену систему, як ігри;

в) мати мінімальні потреби у людських ресурсах.

# 2 Аналіз існуючих методів генерації

## 2.1 Видiлення критеріїв оцінки методів

Для порівняння згенерованого тексту існує низка критеріїв, але метою даної роботи є знаходження найкращого з точки зору інтеграції методу, а не з точки зору ефективності. Для конкретизації сфери генерації було обрано ігри, оскільки вони мають обмеження у ресурсах та віддають перевагу масштабованим методам.

Сучасні методи автоматизації генерації можна умовно віднести до потребуючих навчання, тобто великої кількості даних, або потребуючих людських ресурсів для проектування таких систем. Перші відзначаються перевагою у консистентності згенерованого тексту, але зазвичай витрачають багато ресурсів системи. Другий тип навпаки являє собою економний варіант, але потребує людських ресурсів на кожному етапі проектування, та погано масштабується.

Урахувавши особливості поставленої задачі та основні підходи до її розв’язання, можна виділити наступні критерії вибору методу:

а) витрати пам’яті на стадії ініціалізації;

б) витрати пам’яті на стадії праці;

в) витрати людських ресурсів на проектування та навчання;

г) можливість масштабування — можливість використати вже ініціалізовану систему для генерації нових даних.

## 2.2 Математичні методи

### 2.2.1 Лінгвістичні моделі

Описана у [1] модель являє собою оцінку ймовірності певного розподілу з множини прикладів , кожен з яких складається з послідовності символів різної довжини .

Оцінка ймовірності вираховується за формулою:

При такій реалізації модель вираховує ймовірність вихідних символів при умові вхідних. Наступним кроком є отримання завдання як параметра, тоді формулу автори умовно описують як .

Деталі цього метода можна знайти у [5]. Суть цього процесу складається у отриманні контексту, питання та відповіді представлених у послідовності міток. Потім з цього формуються відповідні матриці, де рядок — відповідна репрезентація мітки у символах. Після навчання отримуємо матриці контексту та питання.

Така модель, при зміні відповідних ймовірностей символів, дозволяє реалізувати систему, яка залежить можем приймати завдання як вхідний параметр. Основною метою цих систем є набуття стійкою відносно цього параметру.

Ця модель була протестована на таких завданнях, як:

* моделювання мови;
* переклад;
* розуміння прочитаного;
* відповідь на питання по тексту.

Отримані результати наведено у таблиці 2.1

Таблиця 2.1- результати тестування лінгвістичної моделі

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Датасет | Завдання | Результат |
| Моделювання мови | LAMBADA | Передбачити останнє слово у реченні | 52.66% та 63.24% після урахування проблем з моделюванням останнього слова |
| Моделювання мови | Winograd Schema Challenge | Визначення значення слова у реченні | 70.70% |
| Розуміння прочитаного | CoQA | Відповідь на запитання з обробленого діалогу | 55 за метрикою ROUGE F1, без збору відповідних даних |
| Переклад | - | Англійська-Французька | 5 BLEU |
| Переклад | - | Французька -Англійська | 11.5 BLEU |
| Відповідь на питання по тексту | SQUAD | Відповідь на питання | 4.1% |

Основною вадою цього методу є розмір датасету «WebText», на якому проходило тренування - 40 GB. Обробка цього датасету генерує модель з розміром словника 50,257 слів. Робота з таким словником може потребувати великої кількості ресурсів.

### 2.2.2 Моделювання з блоків

Підхід моделі [2] будується на розбиті тексту на блоки і побудову ациклічного графу з цих блоків. Блоки без вхідних дуг, називаються початковими блоками. Блоки без вихідних дуг – кінцевими. Генерація відбувається шляхом вибору випадкового шляху між початковим і кінцевим блоком.

Кожен наступний блок бере потрібну інформацію за попередніх блоків. Таким чином, один з блоків має можливість вказати локацію і ключові предмети, а всі наступні блоки використовують цю інформацію.

Така структура потребує щоб всі можливі шляхи мали потрібну інформацію, тобто щоб не було згенеровано шлях до блок потребуючого локацію, через блоки які не містять цю інформацію.

Кожен блок також пов’язаний з усіма потрібними йому активами, що дозволяє економити ресурси та використовувати такі блоки в інших місцях.

Такий підхід дозволяє використовувати статичну генерацію, тобто економний с точки зору ресурсів системи, але , як зазначили автору цієї моделі, потребує великої кількості планування та навичок проектування таких систем.

Для тестування цієї моделі було створено гру на Unity Engine та створено 8 квестів. Порівняння квестів з вручну підібраних блоків (Hs), згенерованих (Gs) та абсолютно випадкових (Rs) виявило, що різниця між Gs та Rs невелика, хоча різниця між Hs там Gs приблизно такого ж масштабу.

### 2.2.3 Використання міток або тегів

Підхід з використанням міток заснований на принципі створення деякого контенту, та розмітки його відповідними тегами. Алгоритм генерації полягає у запиті за тегами, знаходженні усіх сутностей з відповідними тегами, та вибором випадкової сутності з знайдених.

Модифікація цього методу описана у документі [3] полягає в наступних моментах:

* Система перевірки запиту на генерацію, яка видає помилку при неможливості згенерувати контент за запитом
* Опціональні теги, тобто теги які можливо відкинути у запиті на генерацію
* Система виключення тегів, яка полягає у розмітці контенту тегами, які потребують явно вказаної властивості у запиті
* Колоди – всі теги спочатку розміщуються у випадково згенеровані колоди і коли системі потрібно випадкового вибрати тег, вона витягує його з верху колоди. Така система позволяє проаналізувати результати генерації до їх використання, і, за потреби, виключити деякі випадки.

Така система набула широкого використання у багатьох іграх та сферах, тому її дуже легко розробити, підтримувати та використати у різних сферах без значних змін, але така схема потребує значної кількості людського планування так часу і погано масштабується із збільшенням кількості тегів.

### 2.2.4 Генерація з використанням ієрархій

Модель [4] побудована спеціально для використання у ігровій сфері. Вона полягає у декомпозиції ігрового завдання на під-завдання та події у вигляді графа. Обробка цього графу відбувається за допомогою системи планування і моніторингу.

Згідно цієї моделі завдання можна представити у вигляді проблеми планування

де *P* — атомарні символи, *O* — доступні операції планування, — початковий стан, — список цілей

Система планування генерує таке завдання, користувач виконує одну з доступних операцій змінюючи стан системи, а система моніторингу постійно перевіряє поточний стан на відповідність очікуваному і при заходженні різниці робить запит системі планування на генерацію нового завдання.

Така модель, згідно проведених у [4] досліджень має невелику потребу у ресурсах, але потребує змін при кожному рішенні користувача, та швидко набуває потреб у ресурсах при збільшенні вкладеності плану. Хоча на мою думку така система не є достатнім рішенням проблеми генерації тексту, але вона може виступати системою моніторингу абстрактного стану систем генерації. Зараз алгоритми для генерації полігонів Вороного використовуються для генерації початкового стану деякої мапи місцевості, беручи у увагу, недолік варіанту [1] з великим словником є можливість комбінації цих двох методів, який дозволить використати вже натреновану першу модель, але без використання всього словника.

Метод з [4] може виступати розміткою потрібного словника і здійснювати його постійний моніторинг протягом інтеракції. Нажаль, знайти результати використання такої схеми мені не вдалося, але вона залишає за собою недолік у 40 Gb початкового датасету та постійних змін у структурі.

### 2.2.5 Марківські моделі

Описаний у [6] підхід заснований на генерації емоції за допомогою Марківських моделей, базується на ймовірностях переходу між станами, асоційованими з певною емоцією під дією сигналу, але описує лише процес генерації емоції.

Даний підхід може бути застосований, для вибору потрібної емоції і передачі управління наступному ланцюгу Маркова, який за отриманою емоцією буде генерувати текст. Нажаль, у такому вигляді, система не дає можливості описати умовну схему, тому не має можливості адаптації, але не використовує великої кількості ресурсів.

## 2.3 Висновки

У таблиці 2.2 наведена оцінка по кожному з критеріїв, від 1 — погано, до 5 -дуже добре. Оскільки всі критерії рівні за вагою можемо оцінити метод сумою балів за кожним критерієм.

Таблиця 2.2 — Порівняння математичних методів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Витрати пам’яті (ініціалізація) | Витрати пам’яті (генерація) | Людські ресурси | Масштабування | ∑ |
| Лінгвістичні моделі | 1 | 3 | 5 | 5 | 14 |
| Марківські моделі | 4 | 4 | 3 | 2 | 13 |
| Моделювання з блоків | 4 | 5 | 2 | 2 | 13 |
| Використання міток або тегів | 3 | 5 | 1 | 1 | 10 |
| Генерація з використанням ієрархій | 3 | 3 | 2 | 1 | 9 |

На основі проведеного порівняльного аналізу можна зробити висновок, що основними методами можуть виступати лінгвістичні моделі, Марківські моделі та моделювання з блоків. Кожен з них має свої вагомі недоліки, але модифікація лінгвістичної моделі, с точки зору використання ресурсів, зараз не здається можливою. У аналізі Марківських моделей був запропонований метод, який дозволить вирішити проблему адаптації.

Адаптацію методу Марківських моделей можна отримати, якщо дозволити вкладеність ланцюгів, та замість опису конкретного ланцюга, описувати його схему, за якою вже буде створюватися ланцюг Маркова. Цей підхід схожий до моделювання блоків і потребує значно більших навичок проектування таких систем ніж звичайні Марківські моделі.

# 3. Математичне забезпечення

## 3.1 Принцип моделювання ланцюгами Маркова

У класичній інтерпретації ланцюга Маркова, згідно визначення з [9] - це випадковий процес ,у якого область визначення T – дискретна множина точок , а простір станів являє собою дискретну множину точок .

В момент часу можливі зміни стану(тобто перехід з одного стану в інший). Такий випадковий процес можна розглядати як послідовність дискретних випадкових величин яка називається дискретною випадковою послідовністю.

Тоді простим ланцюгом Маркова називається дискретна випадкова послідовність, для якої одновимірна умовна функція розподілу:

|  |  |
| --- | --- |
|  | (1) |

З такого визначення випливає, що ланцюг Маркова складається з множини станів та переходів. Переходи характеризуються ймовірністю, а стани відповідають поточному результату. У таких моделях випадкова величина описується станом.

У моїй моделі випадкова величина описується переходом, а стан характеризується лише можливими з нього переходами.

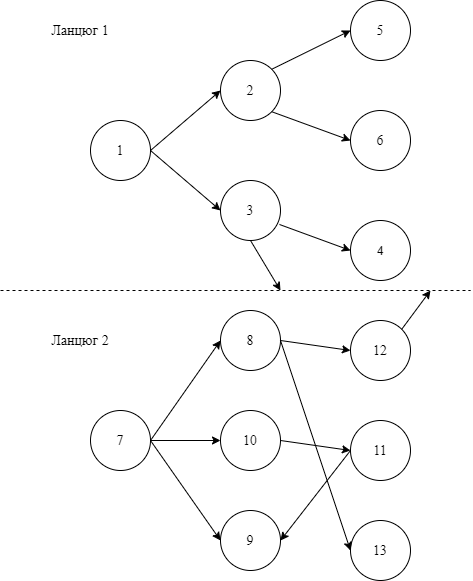
Розглянемо архітектуру такої моделі (рис 3.1).

Рисунок 3.1 – Архітектура вкладеного ланцюга Маркова

При переході між ланцюгами ми потрапляємо в той стан другого ланцюга, з якого було зроблено переміщення. При закінченні роботи одного з ланцюгів — керування передається ланцюгу який його викликав у місце виклику.

Ця модель базується на принципі ланцюгів Маркова, але, у загальному випадку, не відповідає визначенню ланцюга Маркова (1):

де ,

Для доведення цього твердження потрібно ввести декілька позначень.

Загальним ланцюгом назвемо пару, яка складається з множин пар , де S- множина станів, P- множина переходів, - початковий стан, . Позначимо таку множину .

Множину переходів можна представити у вигляді пари матриць Де, - ймовірність переходу із стану в стан , - випадкова величина отримана при переході із стану в стан

Під-ланцюгом назвемо пару з множини .

Ланкою назвемо пару , де ,

Для забезпечення кінцевої генерації, сума ймовірностей переходів із стану менше або рівна 1, та наявна умова відсутності циклічних під-ланцюгів та ланок в яких не входить вершина з сумою ймовірностей переходів з неї менше 1:

|  |  |
| --- | --- |
|  | (2) |

|  |  |
| --- | --- |
|  | (3) |

Ймовірність — назвемо ймовірністю виходу з ланцюга.

Побудуємо початковий ланцюг , який складається і підрахуємо ймовірності виходу: , . За умови наявності переходу між двома ланцюгами, виконаємо роботу одного з них. Нам неважлива кількість таких переходів, та їх розташування, доки виконуються умови (1) і (2), оскільки один з ланцюгів першим закінчить виконання своєї роботи і керування перейде до іншого. Будемо вважати, що перший ланцюг закінчив роботу першим, тоді підрахуємо ймовірність виходу для другого ланцюга:

|  |  |
| --- | --- |
|  | (4) |

Оскільки між двома ланцюгами існує хоча б один перехід, остання ймовірність не дорівнює 0. Тоді, при умові, що — вихід, отримуємо

## 3.2 Формування моделі

Моделювання генерації тексту за допомогою таких ланцюгів складається з трьох основних частин: схема, опис станів, опис тексту.

Схемою назвемо ланцюг Маркова першого рівня. Такий ланцюг описує структуру згенерованого речення. Кожен стан такого ланцюга відповідає за місце в тексті, а кожен перехід представляє собою ланцюг Маркова. У таких ланцюгах нульова ймовірність виходу.

Рисунок 3.2 – Схема тексту

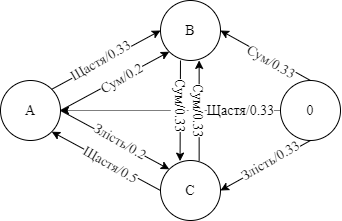
Кожен з цих ланцюгів представляє собою два рівні: опис станів та опис тексту. Перший рівень описує ланцюга для емоцій. Він містить початковий стан, з якого відбувається переміщення у стан відповідної емоції. При переході з початкового стану виконується генерація відповідного тексту на другому рівні. Інші переходи між станами описують ймовірність зміни емоції, а також ненульову ймовірність виходу.

Рисунок 3.3 – Перший рівень у вигляді ланцюга Маркова

Стан «A» відповідає за щастя, «B»- сум, «C» - злість а стан «0»- початковий стан.

Рівень опису тексту, включає в себе дерево ймовірних реплік. При змінні стану генерується відповідна репліка. Також у таких станів може бути ненульова ймовірність виходу і запуску ланцюга емоцій( ланцюга із другого рівня).

Схеми використаних ланцюгів наведено у Додатку В.

## 3.3 Висновки

Розроблено математичне забезпечення системи генерації емоційно забарвлених текстів. Для цього було спроектовано дворівневий вкладений ланцюг Маркова, де перший рівень відповідає за обранні емоції, а другий — за текст для відповідних емоції. У ході генерації відбувається зміна станів першого рівня як через переходи у другому, так і після кінця генерації у відповідному рівні.

За зупинку генерації відповідає ймовірність зупинки, яка розраховується за формулою:

# 4. Програмне забезпечення

## 4.1 Архітектура програми

Програмне забезпечення для розв’язання задачі складається з трьох частин: Builder, Linker та Chain. Перші дві компоненти відповідають за побудову ланцюга із схеми, а Chain — за реалізацію переміщення по ланцюгу.

Для роботи ПЗ отримує на вхід схему ланцюга. Схема описує переходи між станами та складається з полів From, To, Chance, Type та String. Поля From і To відповідають за напрям переходу. Chance відповідає за ймовірність переходу, Type — перехід між ланцюгами, або між станами, String — рядок для генерації, або шлях до схеми ланцюга.

У рядку для генерації можуть знаходитись змінні, які відповідають синтаксису: @variable\_name@. Змінні введені для надання можливості описувати шаблони у схемах, замість самих схем.

Детальніше структуру вхідних даних розглянуто у розділі 4.3.

На рисунку 4.1. зображено схему роботи та взаємодії цих компонент.

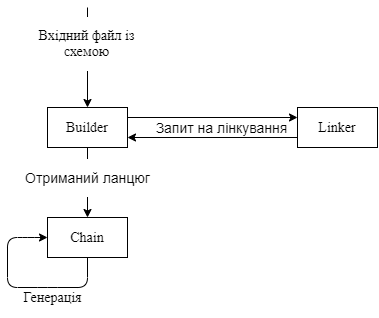


Рисунок 4.1 — Схема взаємодії модулів розробленого програмного забезпечення

Робота програми складається з двох етапів: побудова ланцюга, використання ланцюга для генерації.

Компонента Builder виконує рекурсивну побудову ланцюга. Стадія побудови кожного ланцюга складається з pre-build та build етапів.

Pre-build відповідає за валідацію вхідного файлу, та розрахунок потрібної пам’яті. Валідація складається з перевірки формату та розміру файлу. Якщо файл не знайдено, або він пустий — виконується зупинку процесу Build.

Build стадія складається з рекурентного зчитування файлу та розпізнавання вказаної схеми. Якщо у зчитуваній схемі вказано, що стану відповідає ланцюг — builder відкриває відповідний файл та починає побудову вказаного ланцюга. Якщо у схемі вказано вже існуючий ланцюг — builder виконає зв’язок з цим ланцюгом. У разі отримання переходу для побудови речення builder проаналізує це речення на наявність змінних і, за потреби, зробить запит на лінкування.

За стадію лінкування відповідає модуль Linker. Він отримує потрібний рядок і знаходить у ньому зміну, яку потім замінює потрібним значенням. Така модель представляє собою статичне лінкування, потрібне для підтримки можливості опису схем. У розробленому програмному забезпеченні підтримана лише функція декларації змінної.

Після закінчення стадії Build, керування передається компоненті Chain. Робота компоненти Chain складається із вибору наступного переходу та реалізації переходу між станами. Вибір наступного стану відбувається шляхом генерації випадкового значення із рівномірного розподілу та порівняння його із ймовірністю активації переходу + ймовірність перевірених переходів.

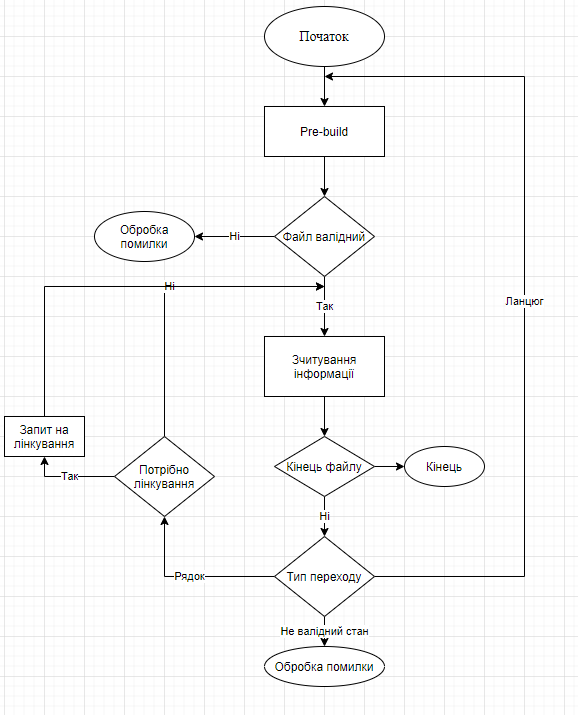
Після отримання потрібного переходу відбувається його активація, шляхом генерації потрібного рядка, або запуском вкладного ланцюга. Після активації відбувається пошук нового стану, доки не буде досягнуто стан виходу.

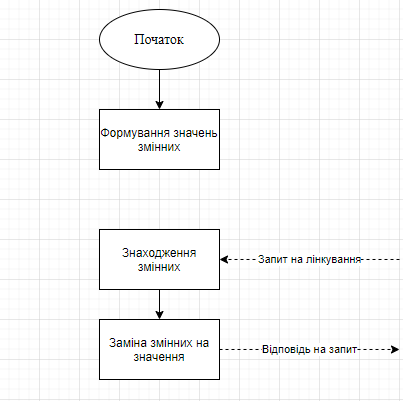
## 4.2 Опис розроблених алгоритмів

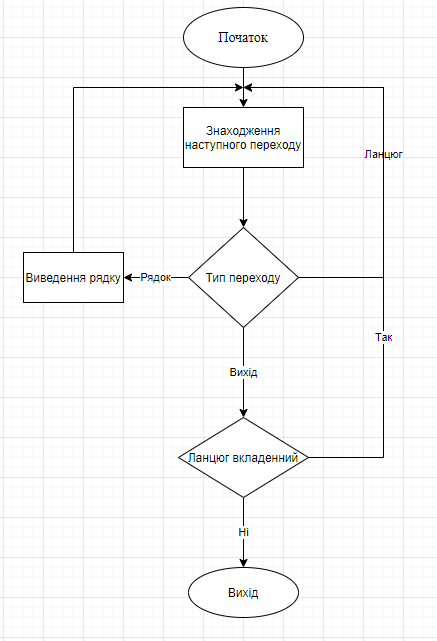
На рисунку 4.2 представлено схему алгоритму компоненти Build.

Блок-схема 4.3 описує алгоритм лінкування змінних.

Алгоритм зміни станів наведено на рисунку 4.4

Рисунок 4.2 — Блок-схема алгоритму компоненти Build

Рисунок 4.3 — Блок-схема алгоритму компоненти Link

Рисунок 4.4 — Блок-схема алгоритму компоненти Chain

## 4.3 Формат вхiдних даних

### 4.3.1 Формат схеми ланцюга

Вхідними даними для ініціалізації системи є текстова схема вкладеного ланцюга Маркова із відповідними параметрами. Така схема описує переходи між станами. Для опису переходу використовують наступний параметри:

From — з якого стану відбувається перехід

To — у який стан відбувається перехід

Chance — ймовірність активації переходу

Type — тип переходу. “Chain” — відповідає ланцюгу, “String” — відповідає рядку

String — рядок для генерації, або шлях до файлу схеми ланцюга. Якщо тип переходу “String”, то рядок може містити зміні. Місце для вставки значення змінної відповідає синтаксису @variable\_name@

### 4.3.2 Формат файлу лінкування

Для ініціалізації модуль Linker потребує значень змінних, які описані у файлі Globals.txt. У цьому файлі перечисленні змінні у наступному вигляді:

* Операція над змінною. Наразі підтримується тільки операція define, яка декларує зміну
* Ім’я змінної
* Значення змінної

## 4.4 Формат результуючих даних

Результатом генерації емоційно забарвлених реплік є 10 згенерованих реплік (рисунок 4.5). Така кількість потрібна для показу можливостей схеми генерувати різні тексти.

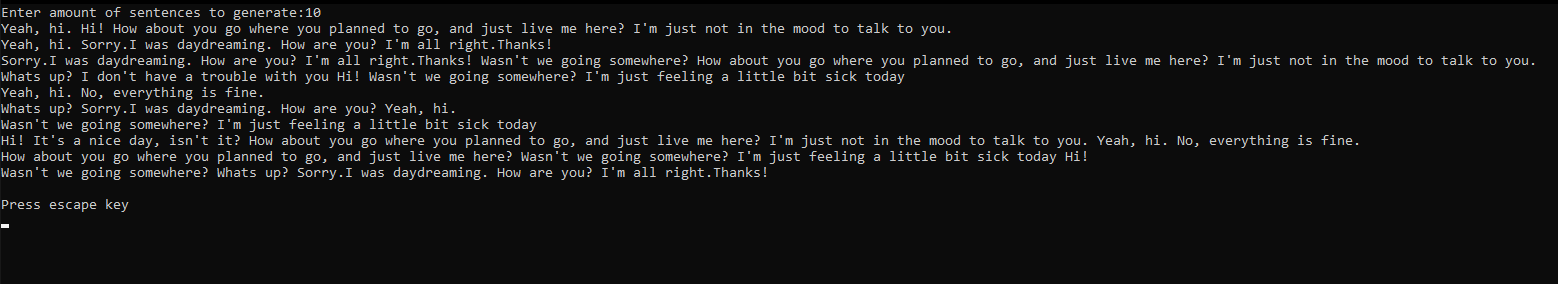
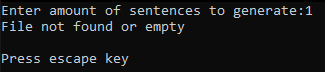
Під час спроби завантажити невалідну схему ланцюга в програму, остання виведе повідомлення про помилку ( рисунок 4.6).

Рисунок 4.5 — Приклад згенерованих речень

Рисунок 4.6 — Повідомлення про помилку

## 4.5 Результати тестування

Система збудувала ланцюг за схемою наведеною у додатку В. Після цього було згенеровано 10 речень. Через складну рекурентну структуру ланцюг потрібно було створювати для кожної ітерації тестування, цей недолік внесено до плану майбутніх покращень.

У таблиці 4.1 наведено показники оцінки за критеріями витрат ресурсів.

|  |  |  |
| --- | --- | --- |
| Стадія | Ініціалізація | Генерація |
| Ресурси процесора | 4ms | 1ms |
| Ресурси пам’яті | 19kb | 19kb |

Результати отриманні завдяки вбудованим функціям моніторингу ресурсів у VisualStudio 19. На рисунках 4.7 та 4.8 зображено екрани моніторингу.

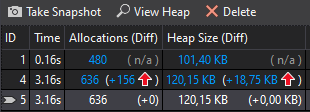


Рисунок 4.7 — Heap monitoring. 1- початок роботи, 4- після ініціалізації, 5 -під час роботи

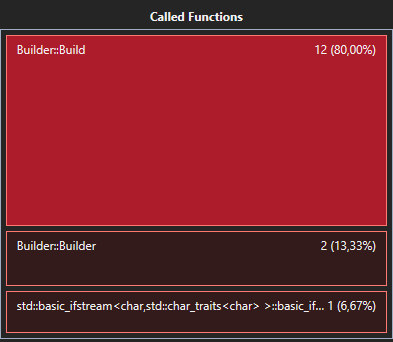


Рисунок 4.8 – Моніторинг ресурсів процесора

На рисунку 4.9 наведено результат генерації та нижче описано текстовий варіант

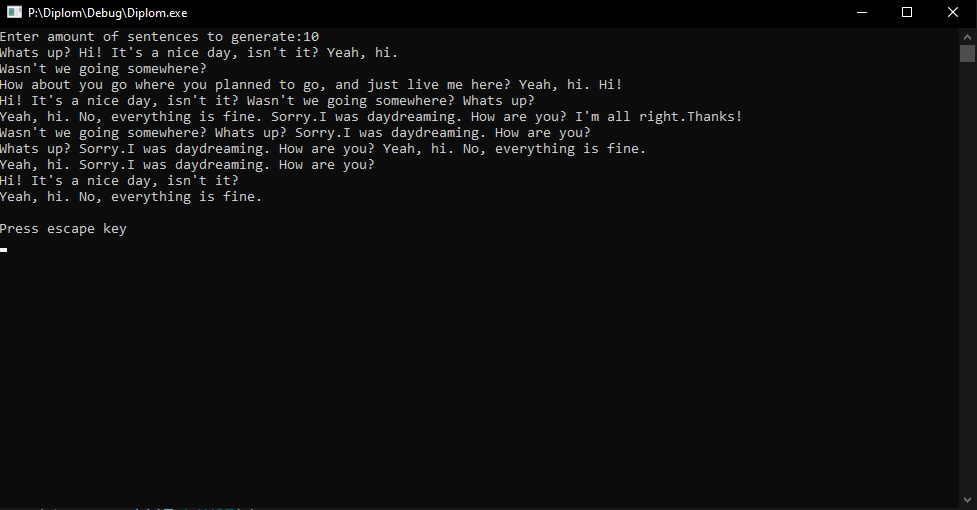


Рисунок 4.9– Результат генерації

“- Whats up? Hi! It's a nice day, isn't it? Yeah, hi.

* Wasn't we going somewhere?
* How about you go where you planned to go, and just live me here? Yeah, hi. Hi!
* Hi! It's a nice day, isn't it? Wasn't we going somewhere? Whats up?
* Yeah, hi. No, everything is fine. Sorry.I was daydreaming. How are you? I'm all right.Thanks!
* Wasn't we going somewhere? Whats up? Sorry.I was daydreaming. How are you?
* Whats up? Sorry.I was daydreaming. How are you? Yeah, hi. No, everything is fine.
* Yeah, hi. Sorry.I was daydreaming. How are you?
* Hi! It's a nice day, isn't it?
* Yeah, hi. No, everything is fine.”

## 4.6 Висновки

Розроблено програмне забезпечення для автоматизованої системи генерації емоційно забарвленого тексту. Реалізована система отримує схему вкладеного ланцюга на вхід і генерує відповідний текст.

Отриманні результати демонструють низьку консистентість речень, такий результат може бути через невдало описану архітектуру емоційного рівня, яка включає в себе зв’язки між емоційними станами, кожен з яких генерує речення з початку і до кінця. Для покращення результату потребується більш складна схема, тобто більше людських ресурсів і алгоритмів, для складання недетермінованих шляхів.

Однак згідно результатів моніторингу така система економна с точки зору потреби у ресурсах. Модуль Linker надає можливості масштабування, але потребує вирішення проблеми з поверненням ланцюга у початковий стан

# Висновки

У роботі розглянуто основні підходи до генерації тексту: лінгвістичні моделі, Марківські ланцюги, з використанням ієрархій, моделювання з блоків та за допомогою тегів. У результаті проведеного порівняльного аналізу за наперед визначеними критеріями для вирішення поставленої задачі обрано гібридний варіант між моделюванням з блоків та Марківськими ланцюгами.

Гібридний варіант представляє собою ланцюг Маркова з ймовірністю виходу, який не задовольняє основній властивості Марківських ланцюгів. Підхід з моделювання з блоків застосовано на стадії створення такого ланцюга.

Спроектоване математичне забезпечення реалізовано програмно. Розроблена відповідна програмна система та схеми ланцюгів.

# Перелiк посилань

1. Language Models are Unsupervised Multitask Learners / Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever — 2019.
2. Generating Side Quests from Building Blocks / Tomas Hromada, Martin Cerny, Michal Bıda, and Cyril Brom — 2015
3. Procedural Level and Story Generation Using Tag-Based Content Selection / Jurie Horneman
4. Hierarchical Generation of Dynamic and Nondeterministic Quests in Games/ Edirlei Soares de Lima, Bruno Feijó, Antonio L. Furtado — 2015
5. McCann, B., Keskar, N. S., Xiong, C., and Socher, R. The natural language decathlon: Multitask learning as question answering - 2018.
6. A Computational Architecture to Model Human Emotion / Arun Chandra/ - 2009
7. GPT-2: 1.5B Release [Електронний ресурс]. — Режим доступу: <https://openai.com/blog/gpt-2-1-5b-release/>
8. GPT-2 GitHub page [Електронний ресурс]. — Режим доступу: <https://github.com/openai/gpt-2>
9. Горбань І. І. Теорія ймовірностей і математична статистика для наукових працівників та інженерів / Ігор Ілліч Горбань. – Київ, 2003. – 239 с.

## Додаток А

Iлюстративний матерiал



Рисунок А.1 – Слайд 1

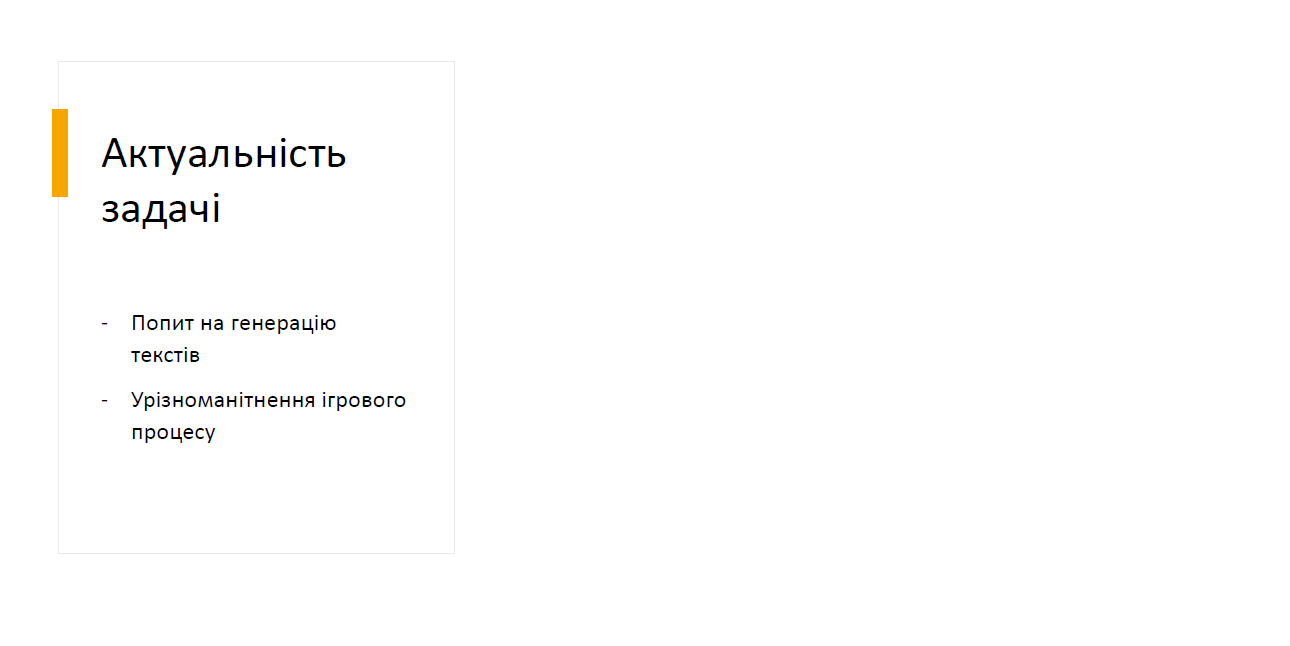


Рисунок А.2 -Слайд 2

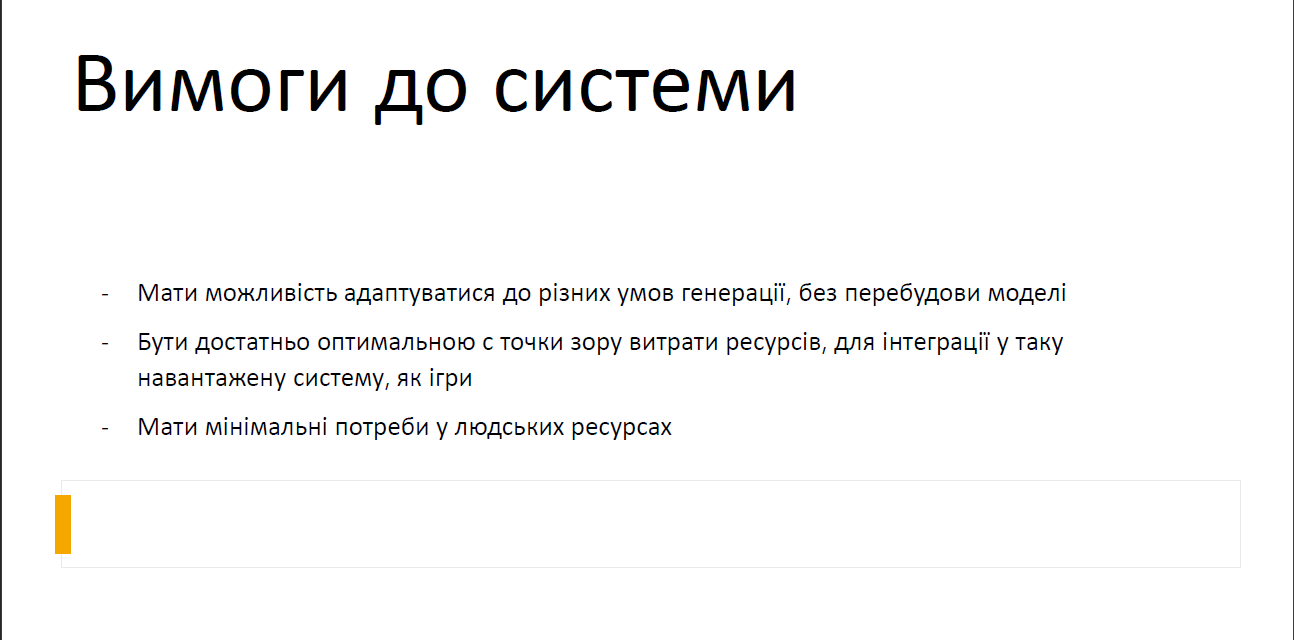


Рисунок А.3 – Слайд 3

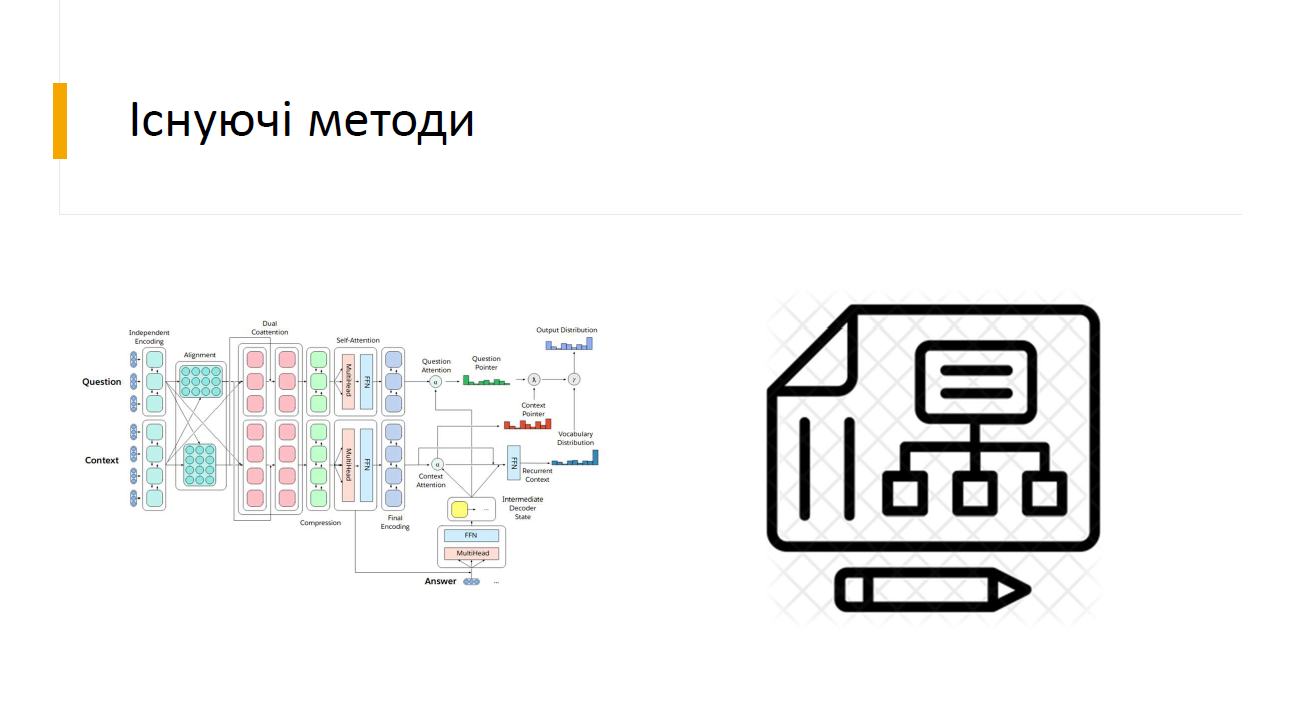


Рисунок А.4 – Слайд 4



Рисунок А.5 – Слайд 5

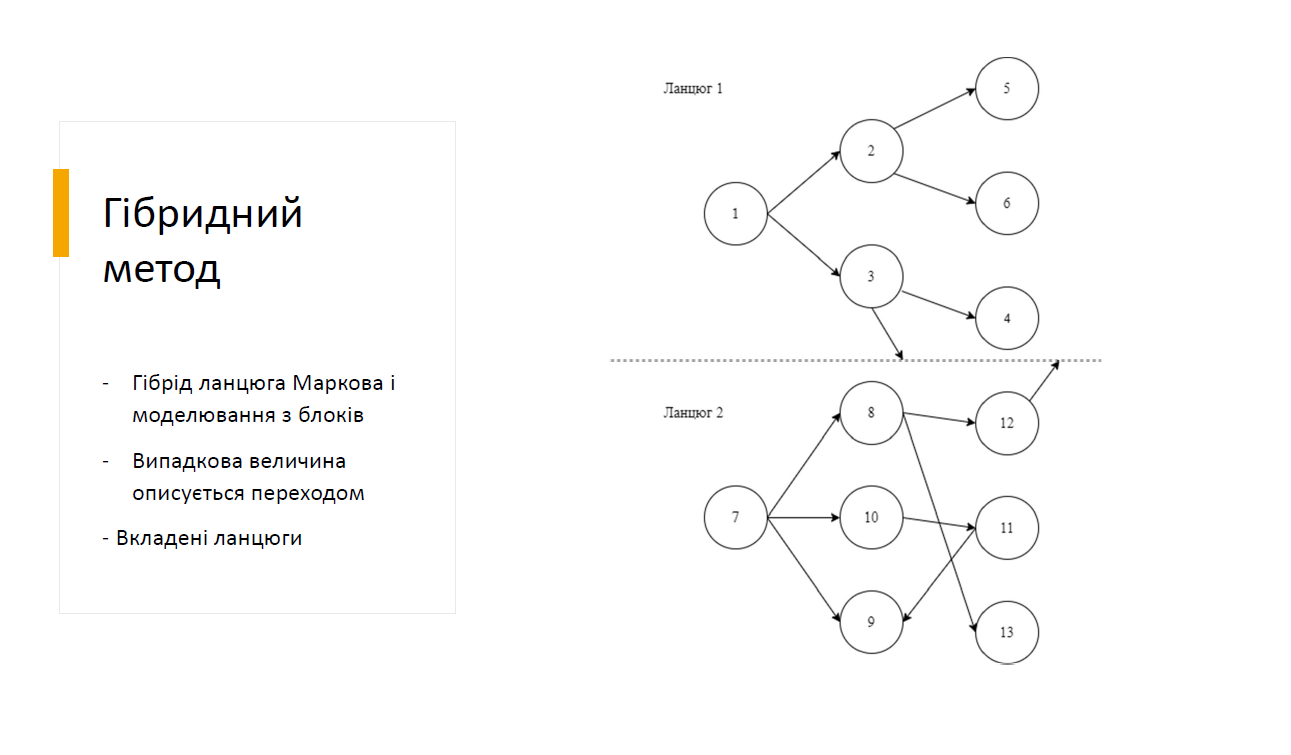


Рисунок А.6 -Слайд 6

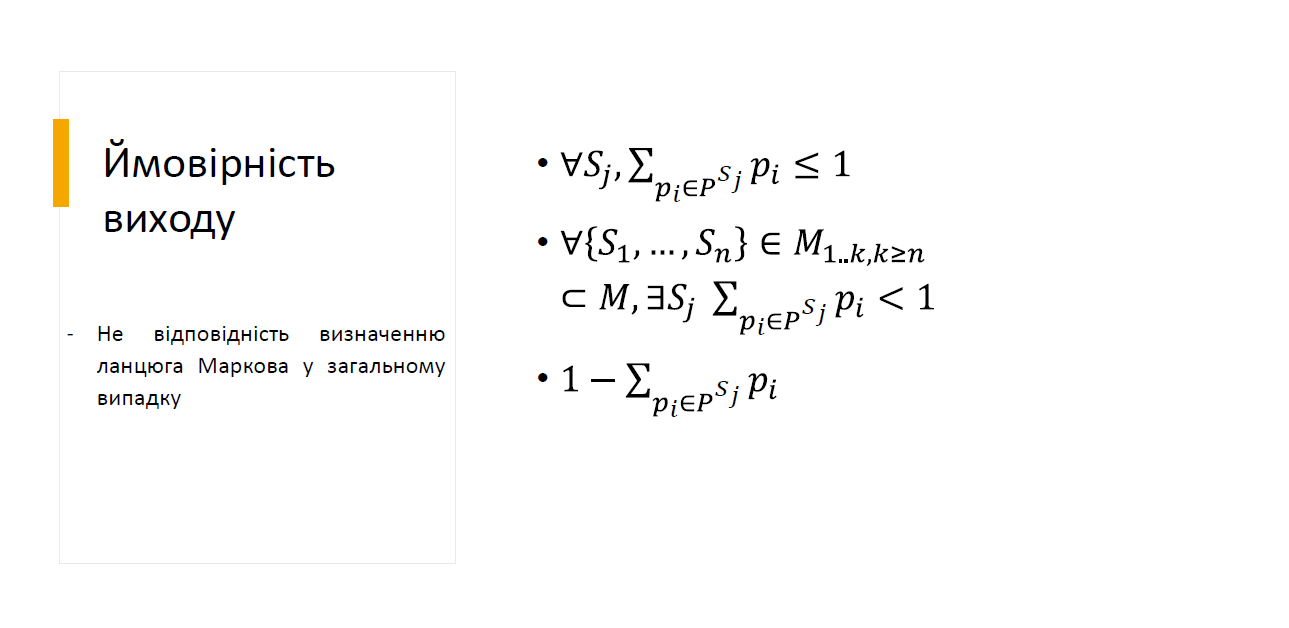


Рисунок А.7 - Слайд 7

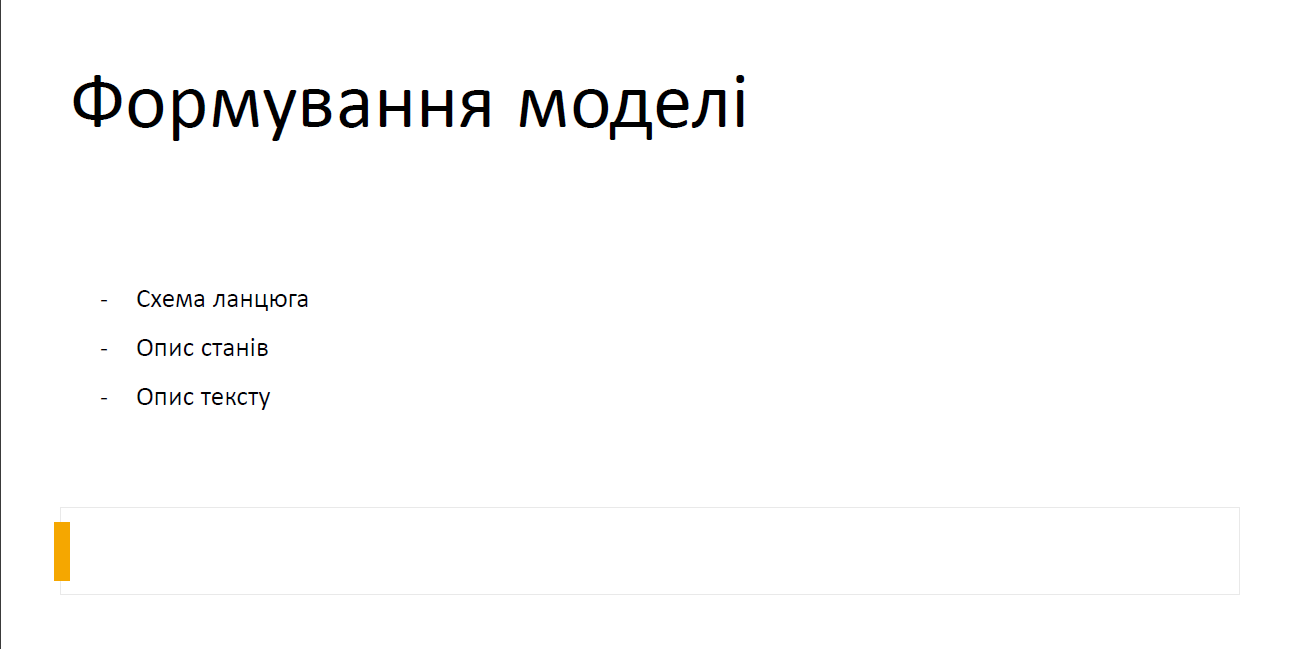


Рисунок А.8 – Слайд 8

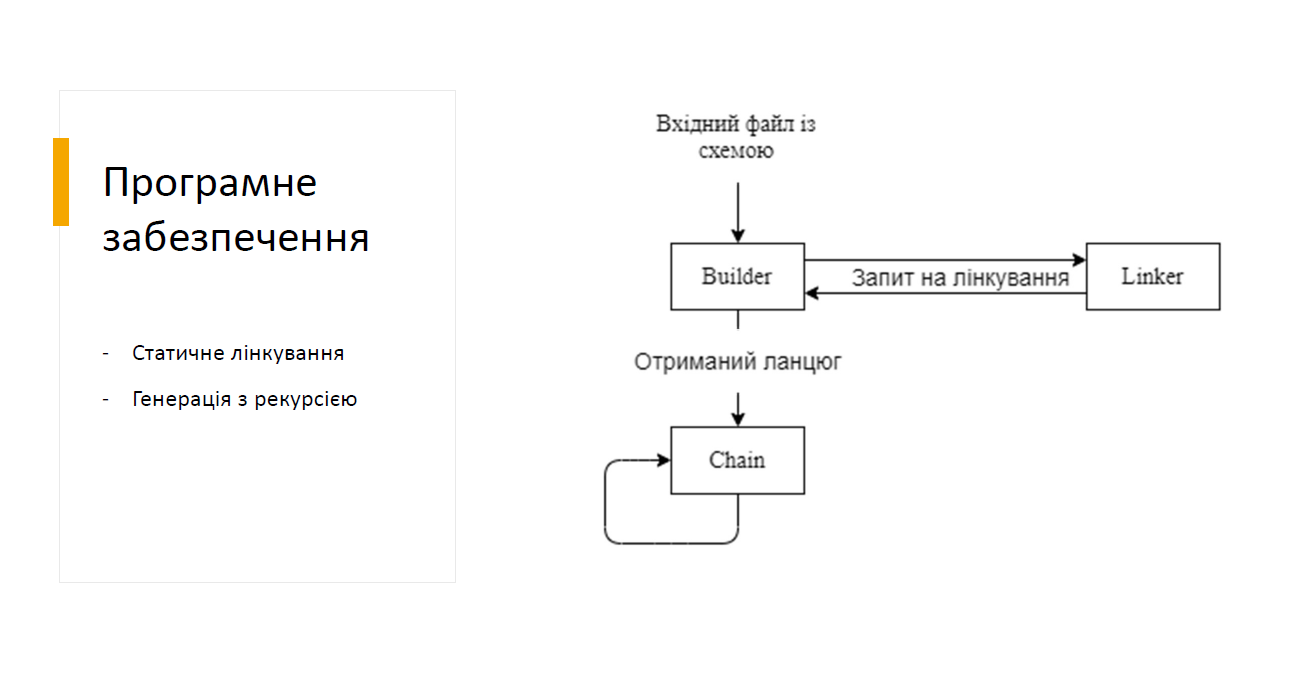


Рисунок А.9 – Слайд 9



Рисунок А.10 – Слайд 10

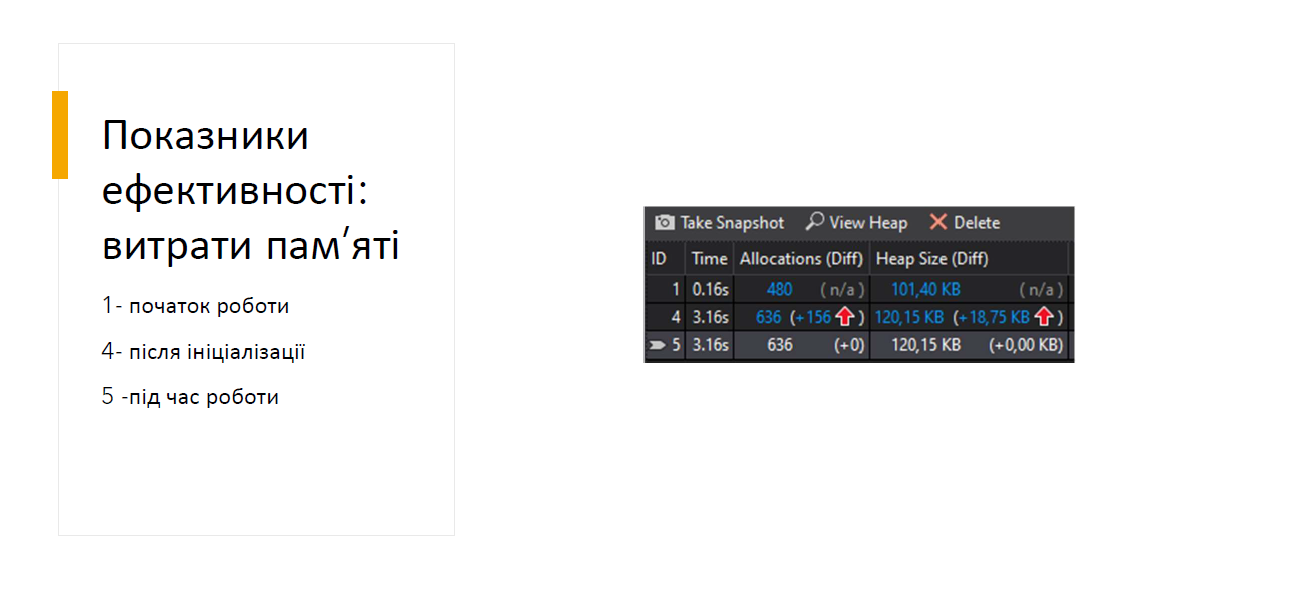


Рисунок А.11 – Слайд 11

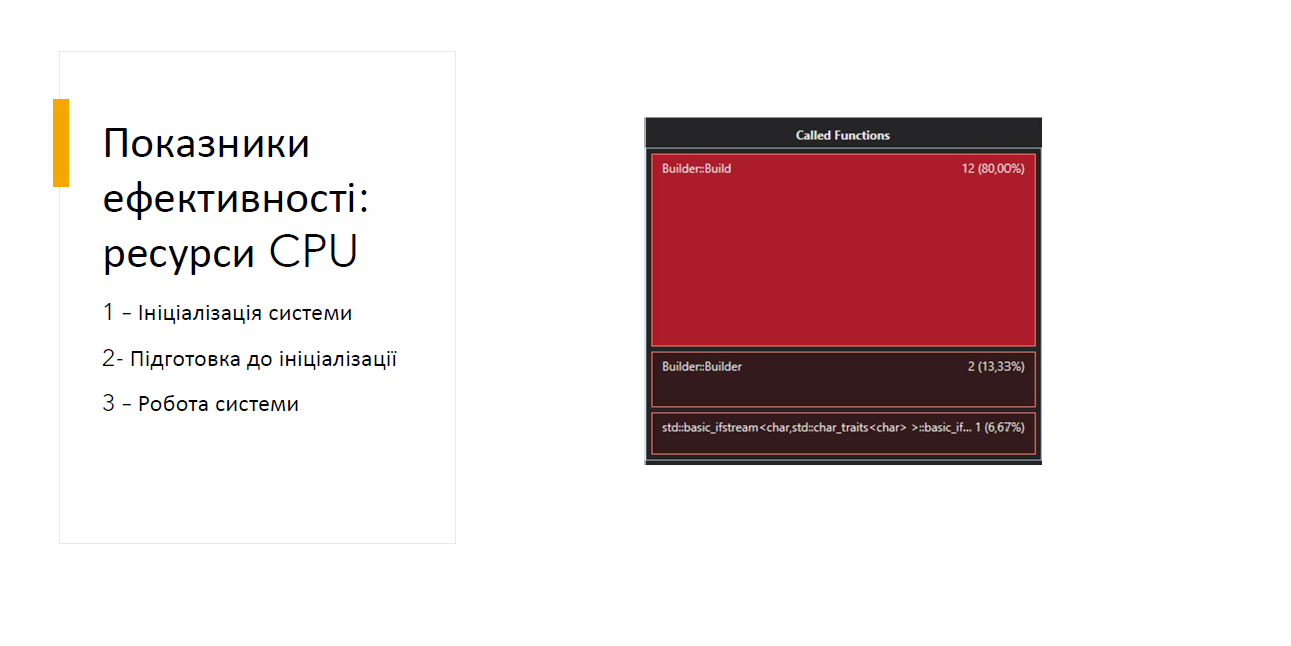


Рисунок А.12 – Слайд 12

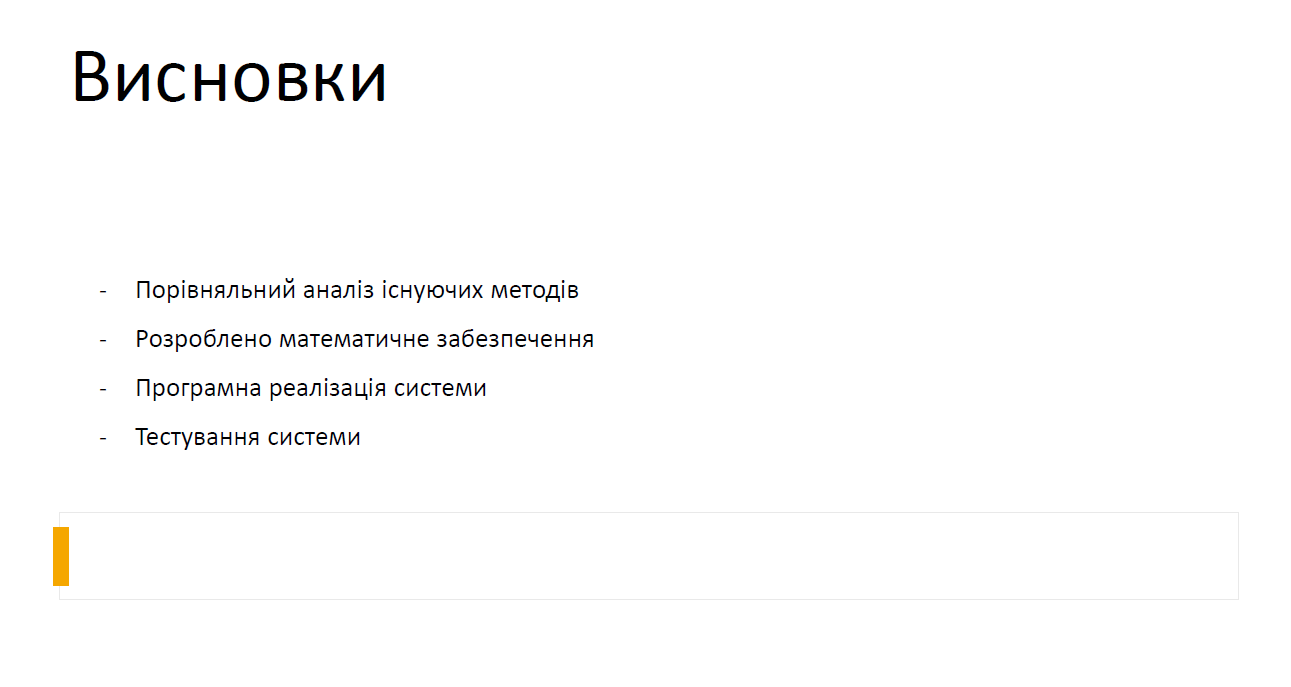


Рисунок А.13- Слайд 13

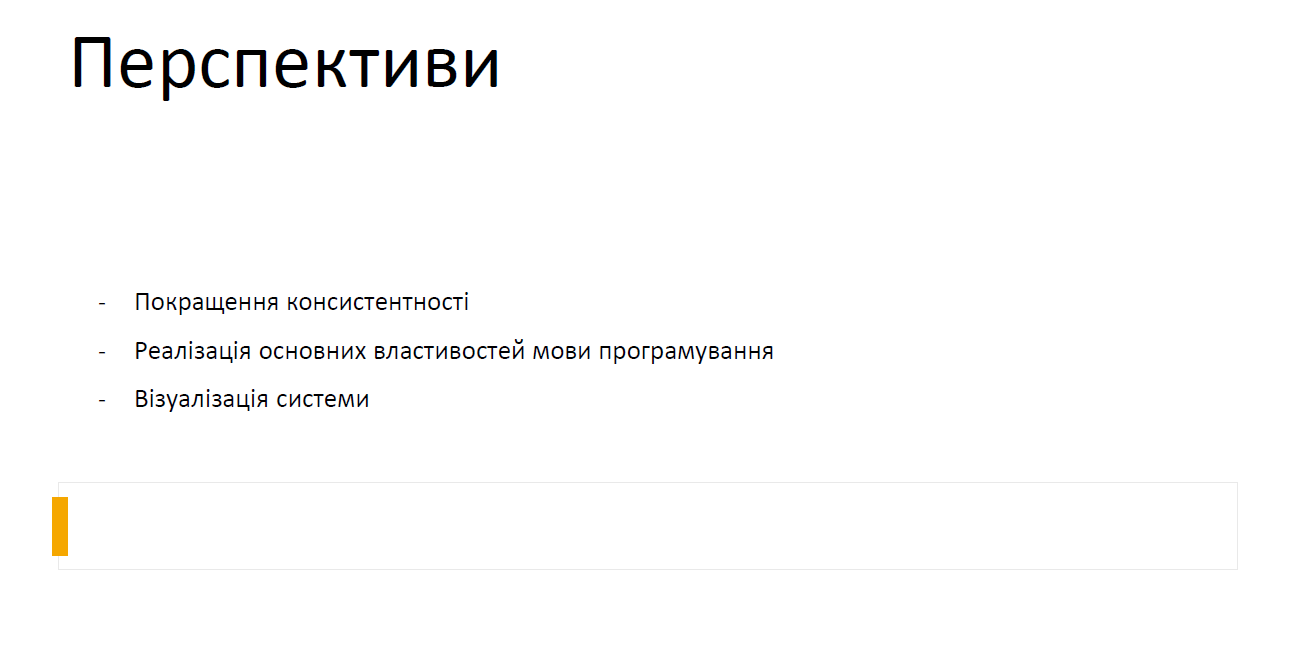


Рисунок А.14 – Слайд 14

## Додаток Б

Програмний код логiки роботи системи

Лiстинг файлу Arc.h – інтерфейс для переходів

#pragma once

#include "IRunnable.h"

#include <memory>

///

///@brief Connects Nodes in Markov chain

///

struct Arc

{

int m\_from, m\_to;

double m\_chance;

bool visited = false;

std::shared\_ptr<IRunnable> m\_iRunnable;

};

Лiстинг файлу Builder.h – компонента Builder

#pragma once

#include "Chain.h"

#include "StringConcat.h"

#include "RegexWrapper.h"

#include "Linker.h"

#include <fstream>

#include <unordered\_map>

///

///@brief Builds chain from a file

///

class Builder

{

public:

Builder();

///

///@brief Builds chain from a file

///

///@param file file

///@param head Chain starting node

///

void Build(std::ifstream &file, const std::shared\_ptr<Chain> &head);

private:

///

///@brief Calculates arcs amount

///

///@param file file

///@return int arcs amount

///

size\_t PreBuild(std::ifstream &file) const;

static std::string ParsePath(const std::string& path);

RegexWrapper m\_variableRegex{"@\\w+@"};

std::unordered\_map<std::string, std::shared\_ptr<Chain>> m\_chainList{};

std::shared\_ptr<Linker> m\_linker;

};

Лiстинг файлу Builder.cpp – компонента Builder

#include "Builder.h"

#include "StringConcat.h"

#include <string>

static constexpr char ResourcePath[] = "Chains/";

Builder::Builder()

{

try

{

m\_linker = std::make\_shared<Linker>("Chains/Global.txt");

}

catch (const char\* msg)

{

throw msg;

}

}

size\_t Builder::PreBuild(std::ifstream& file) const

{

if (!file)

{

return 0;

}

std::string header;

std::getline(file, header);

const auto backup = file.tellg();

const size\_t size = std::count(std::istreambuf\_iterator<char>(file),

std::istreambuf\_iterator<char>(), '\n');

file.clear();

file.seekg(backup);

return size;

}

std::string Builder::ParsePath(const std::string& path)

{

if (path.find('/') == std::string::npos)

{

return ResourcePath + path;

}

return path;

}

void Builder::Build(std::ifstream& file, const std::shared\_ptr<Chain>& head)

{

const auto size = PreBuild(file);

if (size == 0)

{

throw "File not found or empty";

}

head->m\_arcs.reserve(size);

while (!file.eof())

{

auto tmpArc = std::make\_shared<Arc>();

std::string type, string;

file >> tmpArc->m\_from >> tmpArc->m\_to >> tmpArc->m\_chance >> type;

file.get();

std::getline(file, string);

if (type == "Chain")

{

auto file\_path = ParsePath(string);

const auto iPos = m\_chainList.find(string);

if (iPos != m\_chainList.end())

{

tmpArc->m\_iRunnable = iPos->second;

}

else

{

auto chainT = std::make\_shared<Chain>();

m\_chainList.insert({ string, chainT });

tmpArc->m\_iRunnable = chainT;

std::ifstream newFile(file\_path);

Build(newFile, chainT);

}

}

else if (type == "String")

{

if (m\_variableRegex.IsMatch(string))

{

string = m\_linker->Link(string);

}

const auto stringRunner = std::make\_shared<StringConcat>(string);

tmpArc->m\_iRunnable = stringRunner;

}

else

throw "Wrong IRunnable type";

head->m\_arcs.push\_back(tmpArc);

}

file.close();

}

Лiстинг файлу Chain.h – компонента Chain

#pragma once

#include "IRunnable.h"

#include "Arc.h"

#include <vector>

#include <random>

///

///@brief Markov chain

///

class Chain final : public IRunnable

{

public:

explicit Chain();

///

///@brief Finds next node and executes arc action

///

void Run() override;

///

///@brief Chech if state with @State number is present in chain

///

bool IsPresent(unsigned state);

std::vector<std::shared\_ptr<struct Arc>> m\_arcs{};

private:

bool visited = false;

int m\_currentArc;

};

Лiстинг файлу Chain.cpp – компонента Chain

#include "Chain.h"

#include "RandomGenerator.h"

#include <iostream>

Chain::Chain() : m\_currentArc(0) {}

void Chain::Run()

{

while (true)

{

char isFound = 0;

const auto chance = RandomGenerator::Generate();

auto sumChance(0.0);

for (const auto &arc : m\_arcs)

{

if (arc->m\_from == m\_currentArc)

{

sumChance += arc->m\_chance;

if (sumChance - chance > 0)

{

m\_currentArc = arc->m\_to;

arc->m\_iRunnable->Run();

isFound = 1;

break;

}

}

}

if (!isFound)

{

//m\_currentArc = 0;

break;

}

}

}

bool Chain::IsPresent(unsigned state)

{

for (auto s: m\_arcs)

{

if(s->m\_from == state || s->m\_to==state)

{

return true;

}

}

return false;

}

Лiстинг файлу IRunnable.h – інтерфейс для опису дії при переході

#pragma once

class IRunnable

{

public:

virtual void Run() = 0;

virtual ~IRunnable() = default;

};

Лiстинг файлу Linker.h – компонента Linker

#pragma once

#include <string>

#include <unordered\_map>

#include "RegexWrapper.h"

class Linker

{

public:

explicit Linker(const std::string& fileName);

~Linker() = default;

///

///@brief Replaces string with value from globals

///

///@param string String to link

///@return std::string Linked string

/// Puf-puf

std::string Link(const std::string& string);

private:

std::unordered\_map<std::string,std::string> m\_map;

RegexWrapper m\_variableRegex{"(@\\w+@)"};

};

Лiстинг файлу Linker.cpp – компонента Linker

#include "Linker.h"

#include <fstream>

#include <iostream>

Linker::Linker(const std::string& fileName)

{

std::ifstream file(fileName);

while (!file.eof())

{

std::string operation, arg1, arg2;

file >> operation >> arg1 >> arg2;

if (operation == "define")

{

m\_map.insert({ "@"+arg1+"@",arg2 });

}

else

{

throw "Wrong operation type";

}

}

file.close();

}

std::string Linker::Link(const std::string& string)

{

try

{

const auto replacement= m\_map.at(m\_variableRegex.FindMatch(string));

return m\_variableRegex.ReplaceMatches(string,replacement);

}

catch (std::exception& e)

{

std::cerr << e.what() << '\n';

return "";

}

}

Лiстинг файлу StringConcat.h – виводить відповідний рядок при переході

#pragma once

#include "IRunnable.h"

#include <iostream>

///

///@brief Prints sentence after execution

///

///

class StringConcat final : public IRunnable

{

public:

///

///@brief Construct a new @c StringConcat object

///

///@param sentence sentence to print

///

explicit StringConcat(std::string sentence);

///

///@brief Prints sentence

///

void Run() override;

private:

std::string m\_sentence;

};

Лiстинг файлу StringConcat.cpp – виводить відповідний рядок при переході

#include "StringConcat.h"

StringConcat::StringConcat(std::string sentence) : m\_sentence(std::move(sentence)) {}

void StringConcat::Run()

{

std::cout << m\_sentence << ' ';

}

Лiстинг файлу RandomGenerator.h – генератор випадкових чисел за рівномірним розподілом

#pragma once

#include <random>

///

///@brief Generates random float number

///

class RandomGenerator

{

public:

///

///@brief Generates random double number via uniform distribution

///

///@return double generated number

///

static double Generate();

private:

static std::random\_device m\_randomDevice;

static std::default\_random\_engine m\_generator;

static std::uniform\_real\_distribution<> m\_distribution;

};

Лiстинг файлу RandomGenerator.cpp – генератор випадкових чисел за рівномірним розподілом

#include "RandomGenerator.h"

std::uniform\_real\_distribution<> RandomGenerator::m\_distribution{0.0, 1.0};

std::random\_device RandomGenerator::m\_randomDevice{};

std::default\_random\_engine RandomGenerator::m\_generator{m\_randomDevice()};

double RandomGenerator::Generate()

{

return m\_distribution(m\_generator);

}

Лiстинг файлу RegexWrapper.h – інтерфейс для роботи з регулярними виразами

#pragma once

#include <regex>

///

///@brief Wrapper around std::regex

///

class RegexWrapper

{

public:

explicit RegexWrapper(const std::string& regex);

///

///@brief Checks if regex found match in string

///@param string string to check

///@return true match found

///@return false match not found

///

bool IsMatch(const std::string& string);

///

///@brief Finds first match in string

///@param string string to check

///@return std::string first match

///

std::string FindMatch(const std::string& string);

///

///@brief Finds all matches in string

///@param string string to check

///@return std::shared\_ptr<std::cmatch> matches sptr

///

std::shared\_ptr<std::cmatch> FindMatches(const std::string& string);

[[nodiscard]] std::string ReplaceMatches(const std::string& string,const std::string& replacer) const;

private:

std::regex m\_regex;

std::cmatch m\_match;

};

Лiстинг файлу RegexWrapper.cpp – інтерфейс для роботи з регулярними виразами

#include "RegexWrapper.h"

#include <iostream>

RegexWrapper::RegexWrapper(const std::string& regex)

: m\_regex(regex)

{

}

bool RegexWrapper::IsMatch(const std::string& string)

{

std::regex\_search(string.c\_str(),m\_match,m\_regex);

return !m\_match.empty();

}

std::string RegexWrapper::FindMatch(const std::string& string)

{

std::regex\_search(string.c\_str(),m\_match,m\_regex);

if(m\_match.empty())

{

return "";

}

return m\_match[0];

}

std::shared\_ptr<std::cmatch> RegexWrapper::FindMatches(const std::string& string)

{

std::regex\_search(string.c\_str(),m\_match,m\_regex);

return std::make\_shared<std::cmatch>(m\_match);

}

std::string RegexWrapper::ReplaceMatches(const std::string& string,const std::string& replacer) const

{

return std::regex\_replace(string,m\_regex,replacer);

}

## Додаток В

Схема використаного ланцюга

Лiстинг файлу TestChain.txt – початковий файл генерації

F T Ch Type String

0 1 0.33 Chain Happy.txt

0 2 0.33 Chain Sad.txt

0 3 0.33 Chain Angry.txt

1 2 0.33 Chain Happy.txt

1 3 0.2 Chain Angry.txt

1 2 0.2 Chain Sad.txt

2 3 0.33 Chain Sad.txt

3 2 0.33 Chain Sad.txt

3 1 0.5 Chain Happy.txt

Лiстинг файлу Sad.txt – схема ланцюга суму

F T Ch Type String

0 1 0.5 String Yeah, hi.

0 2 0.5 String Wasn't we going somewhere?

1 3 0.5 String No, everything is fine.

1 5 0.5 Chain Happy.txt

2 3 0.5 String I'm just feeling a little bit sick today

2 6 0.33 Chain Angry.txt

Лiстинг файлу Happy.txt – схема ланцюга щастя

F T Ch Type String

0 1 0.5 String Hi!

0 2 0.5 String Sorry.I was daydreaming. How are you?

1 3 0.5 String It's a nice day, isn't it?

1 5 0.5 Chain Angry.txt

2 3 0.5 String I'm all right.Thanks!

2 6 0.33 Chain Sad.txt

Лiстинг файлу Angry.txt – схема ланцюга злості

F T Ch Type String

0 1 0.5 String Whats up?

0 2 0.5 String How about you go where you planned to go, and just live me here?

1 3 0.5 String I don't have a trouble with you

1 5 0.5 Chain Happy.txt

2 3 0.5 String I'm just not in the mood to talk to you.

2 6 0.33 Chain Sad.txt