Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах”

Виконав здобувач вищої освіти
IV курсу, групи КБ-19
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Середа О.О.
«____» _____ 2023 р.

Керівник проекту
доктор технічних наук, професор
_____ Мелешко Є.В.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Середі Олександру Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах*

2. Керівник роботи *Мелешко Єлизавета Владиславівна,* д.т.н., *професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 12-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі соц. мережі	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів генерації та моделювання	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Мелешко Є.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Середа О.О.
(прізвище та ініціали)

АНОТАЦІЯ

Середа О.О. Програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

Метою розробки є програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

Результат роботи – програмна реалізація системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

В процесі роботи над програмною моделлю виконано аналіз існуючих програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на платформах з ОС Windows 10/11, а також у браузерах, які підтримують WebGL 1.0/2.0.

Програму розроблено в середовищі Unity3D 2019.4.40f1 із використанням мови програмування C#.

Ключові слова: інформаційні віруси, імітаційне моделювання, соціальні мережі

ABSTRACT

Sereda O.O. Software for simulation of the spread of information viruses in social networks. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the first (bachelor) level of higher education, software was developed, which is intended for a system of simulation modeling of the spread of information viruses in social networks.

The purpose of the development is the software of the simulation modeling system of the spread of information viruses in social networks.

The result of the work is the software implementation of a system for simulating the spread of information viruses in social networks.

In the process of working on the software model, an analysis of existing software tools was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on platforms with Windows 10/11 OS, as well as in browsers that support WebGL 1.0/2.0.

The program was developed in the Unity3D 2019.4.40f1 environment using the C# programming language.

Keywords: information viruses, simulation modeling, social networks

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

.NET	–	.NET Framework
ПЗ	–	Програмне забезпечення
ВА	–	Модель Барабаші-Альберт
ЗМІ	–	Засоби масової інформації
CLR	–	Common Language Runtime
FCL	–	Framework Class Library
XML	–	Extensible Markup Language
SDK	–	Software development kit
HDRP	–	High Definition Render Pipeline
URP	–	Universal Render Pipeline

ВСТУП

Актуальність теми. Сучасна парадигма управління розглядає людину як один з основних ресурсів, що значною мірою визначає успішність будь-якої діяльності. Даним ресурсом необхідно грамотно керувати, застосовуючи заходи інституційного, мотиваційного та інформаційного характеру.

Соціальні мережі є одним із ефективних каналів поширення інформації в Інтернеті. Вони наблизилися до традиційних ЗМІ у можливості розповсюдження інформації та стали потужним інструментом впливу на громадську думку з комерційною, суспільною та політичною метою. Інформація в Інтернеті, і особливо в соціальних мережах, часто поширюється швидше і має більше охоплення аудиторії, ніж при використанні традиційних ЗМІ. Соціальні мережі часто виступають як постачальники інформації для офіційних ЗМІ, які вчаться перевіряти її достовірність. Управління поширенням інформації у соціальних мережах стає дедалі актуальнішим.

Хоча дослідження поширення інформації у соціальних мережах є складним завданням, оскільки зв'язки між учасниками мережі випадкові і мережа постійно збільшується за рахунок нових користувачів, можна досліджувати поширення інформації не в самій мережі, а в її моделі. Для цього можна використовувати випадкові графи кращого зв'язування. Метод імітаційного моделювання дозволяє багаторазово запускати процес поширення інформації в моделі, щоб визначити ряд важливих характеристик цього процесу, таких як час охоплення всієї цільової аудиторії інформацією, що поширюється, або частка охоплення аудиторії інформацією за певний період часу.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

– Дослідження алгоритмів генерування соціальних графів.

– Дослідження алгоритмів розповсюдження інформації у суспільстві.

– Програмна реалізація системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно моделювати та аналізувати розповсюдження інформаційних вірусів у соціальних мережах.

Таким чином, виходячи з вищеперерахованого, Програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система імітаційного моделювання поширення інформаційних вірусів у соціальних мережах призначена для дослідження та аналізу процесу поширення вірусів, шкідливого контенту або інформаційного впливу в соціальних мережах. Вона дозволяє створювати віртуальні моделі соціальних мереж, де можна відтворити різні сценарії поширення інформації та аналізувати їх наслідки.

Основні призначення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах включають:

1. Вивчення поширення вірусів та інших видів шкідливого контенту в соціальних мережах з метою розуміння механізмів їх поширення та ефективності протишкідливих заходів.

2. Аналіз впливу інформаційного контенту на поведінку користувачів у соціальних мережах і визначення факторів, що сприяють поширенню інформації.

3. Випробування різних стратегій та сценаріїв протидії поширенню вірусів та шкідливої інформації в соціальних мережах для розробки ефективних методів захисту.

4. Прогнозування та аналіз потенційних наслідків поширення вірусів та шкідливої інформації для планування заходів з кібербезпеки та контрмір.

1.2 Область застосування

Застосування системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах має декілька аспектів і може бути корисним у різних областях, включаючи:

1. Кібербезпека та захист від вірусів: Система імітаційного моделювання

дозволяє випробувати різні сценарії поширення вірусів у соціальних мережах та оцінити їх вплив на користувачів та інфраструктуру. Це допомагає розробникам антивірусного програмного забезпечення та кібербезпековим експертам вдосконалити методи виявлення, захисту та реагування на загрози вірусів у соціальних мережах.

2. Соціальні дослідження: За допомогою системи імітаційного моделювання можна вивчити вплив різних факторів на поширення інформації у соціальних мережах. Дослідники можуть аналізувати вплив алгоритмів рекомендацій, структури мережі, характеристик користувачів та властивостей вмісту на поширення вірусів та інформаційного впливу. Це дозволяє зрозуміти соціальні тенденції, механізми впливу та розробити ефективні стратегії маркетингу або впливу на громадську думку.

3. Планування кризового управління: Використання системи імітаційного моделювання дозволяє оцінити можливі наслідки поширення вірусів та шкідливої інформації у соціальних мережах. Це може бути корисним для організацій, урядових установ та інших суб'єктів, які планують заходи кризового управління.

4. Ефективність рекламних кампаній: Система імітаційного моделювання дозволяє оцінити ефективність рекламних кампаній у соціальних мережах, враховуючи їх вплив на поширення інформації та залучення цільової аудиторії. Це допомагає рекламодавцям оптимізувати свої стратегії та бюджети, максимізувати результативність рекламних кампаній та залучення нових клієнтів.

5. Перевірка ефективності політик інформаційної безпеки: Система імітаційного моделювання дозволяє оцінити ефективність політик інформаційної безпеки у соціальних мережах. Вона дозволяє аналізувати вразливості системи, виявляти слабкі місця та розробляти стратегії захисту, що сприяють покращенню загальної безпеки даних та протидії вірусам та зловмисному впливу.

6. Вивчення поведінки користувачів: Система імітаційного моделювання може бути використана для вивчення поведінки користувачів у соціальних мережах під впливом різних факторів. Це дозволяє розуміти, які фактори

спонукають користувачів до поширення інформації та взаємодії, що може бути корисним для розробки персоналізованих рекомендацій, соціальної аналітики та стратегій залучення аудиторії.

В цілому, система імітаційного моделювання поширення інформаційних вірусів у соціальних мережах допомагає вивчати та аналізувати процеси поширення інформації, виявляти потенційні загрози інформаційній безпеці.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Візуалізація соціальної мережі - це графічне зображення мережі з використанням точок та зв'язків між ними. Це дає змогу легко сприймати та аналізувати зв'язки та структуру мережі.

Існує багато інструментів, які допомагають візуалізувати соціальні мережі. Деякі з них - це Gephi, Cytoscape, Pajek та NodeXL.

Зазвичай візуалізація соціальної мережі відображається у вигляді графа, де вузлами є користувачі соціальної мережі, а зв'язки - це взаємодії між ними. Розмір та колір вузла можуть вказувати на важливість користувача або кількість зв'язків, які він має. Зв'язки можуть мати різний тип та вагу, що відображає взаємодію між користувачами.

Одним з прикладів візуалізації соціальної мережі є графічне зображення Facebook, де користувачі зображаються як кола, а їх зв'язки - як лінії, які їх з'єднують.

Візуалізація соціальної мережі є потужним інструментом для аналізу та розуміння структури та зв'язків в мережі. Це може допомогти вивчити взаємодію між користувачами, ідентифікувати групи користувачів, відслідковувати поширення інформації, виявляти впливових користувачів та багато іншого.

Системи моделювання поширення інформації в Інтернеті призначені для вивчення та аналізу процесів розповсюдження інформації в соціальних мережах та інших онлайн-середовищах. Вони використовують математичні моделі та комп'ютерні симуляції для дослідження характеристик цього процесу, таких як

час охоплення всієї цільової аудиторії інформацією, що поширюється, частку охоплення аудиторії інформацією за встановлений час та інші показники.

Як моделі дослідження поширення інформації в Інтернеті часто використовуються випадкові графи кращого зв'язування, які дозволяють враховувати випадковий характер зв'язків між учасниками мережі та динамічний характер зміни розміру мережі.

Системи моделювання розповсюдження інформації можуть бути використані для різних цілей, таких як визначення оптимальних стратегій розповсюдження інформації в соціальних мережах, оцінка ефективності рекламних кампаній в Інтернеті, аналіз впливу соціальних мереж на думку і т.д. Вони можуть також допомогти у плануванні та оптимізації маркетингових та комунікаційних стратегій компаній в Інтернеті.

Gephi

Gephi - це пакет програмного забезпечення для аналізу та візуалізації мережі з відкритим кодом, написаний мовою Java на платформі NetBeans.

Історія

Спочатку розроблений студентами Комп'єнського технологічного університету (UTC) у Франції, Gephi був обраний для Google Summer of Code у 2009, 2010, 2011, 2012 та 2013 роках.

Його остання версія, 0.10.1, була запущена в 2023 році. Попередні версії: 0.6.0 (2008), 0.7.0 (2010), 0.8.0 (2011), 0.8.1 (2012), 0.8.2 (2013) , 0.9.0 (2015), 0.9.1 (2016) і 0.9.2 (2017).

Консорціум Gephi, створений у 2010 році, є французькою некомерційною корпорацією, яка підтримує розробку майбутніх версій Gephi. Членами є SciencesPo, Linkfluence, WebAtlas і Quid. Gephi також підтримується великою спільнотою користувачів, структурованих у дискусійній групі та форумі, які створюють численні публікації в блогах, документи та навчальні посібники.

Використання

Gephi використовувався в ряді дослідницьких проектів в академічних колах, журналістиці та інших сферах, наприклад, для візуалізації глобального зв'язку вмісту New York Times і вивчення мережевого трафіку Twitter під час соціальних заворушень разом із більш традиційними темами аналізу мережі. Gephi широко використовується в цифрових гуманітарних науках (в історії, літературі, політичних науках тощо), спільноті, до якої залучено багато її розробників.

Gephi надихнув LinkedIn InMaps і використовувався для візуалізації мережі для Truthy.

Cytoscape

Cytoscape - біоінформатична платформа з відкритим вихідним кодом, призначена для візуалізації мереж молекулярних взаємодій та біологічних шляхів з можливістю використання додаткових даних, таких як функціональна анотація, інформація про рівень експресії генів та інших. Незважаючи на те, що початково Cytoscape був розроблений для біологічних досліджень, зараз він широко використовується для вирішення різних задач аналізу мереж і їх візуалізації.

Історія

Cytoscape був розроблений в Інституті системної біології в Сіетлі у 2002 році. Перша версія програми, Cytoscape 0.8, була випущена в липні 2002 року, наступні релізи 0.9 і 1.0 відбулися в листопаді 2002 і березні 2003 відповідно. Серія 2.0 оновлювалася з 2004 до 2012 року, після чого в 2013 році було започатковано серію 3.xx. Останнє велике оновлення програми вийшло у 2014 році з виходом Cytoscape v3.3, в якому було реорганізовано основні моделі внутрішніх даних та значно покращено підтримку сторонніх додатків та плагінів.

До версії 3.3 Cytoscape складався з Ядра та додаткових плагінів. Починаючи з версії 3.3 деякі функції Ядра були виділені в основні Плагіни. Ядро – це код, який організовує, відображає, зчитує та записує мережі, але не містить біологічних функцій. Основні Плагіни - це встановлені модулі, що виконують

функції, відмінні від Ядра, але необхідні для використання Cytoscape. На відміну від попередніх версій Cytoscape, функціонал основних плагінів можна оновлювати, не випускаючи нового великого оновлення Cytoscape. Також є додаткові плагіни (доступні в App Store), що дозволяють виконувати певні завдання. У версії 3.6.1 Cytoscape автоматично встановлює актуальну версію Java для роботи.

Наразі підтримку програми здійснює Міжнародний консорціум розробників програм із відкритим кодом. На сьогоднішній день останньою версією Cytoscape є версія 3.8. Вона містить покращення продуктивності та можливостей мережевого рендерингу та покращення інтеграції сервісу NDEx. Нова версія отримала можливості використання для високопродуктивного аналізу, мультимасштабних мереж, мереж білок-білкових взаємодій, доступного аналізу оміки.

Одним із довгострокових планів є тісна інтеграція Cytoscape та NDEx, щоб користувачі природно завантажували та зберігали свої мережі, використовуючи сервіс NDEx.

В даний час Cytoscape експортує мережу у вигляді файлу, який можна використовувати як додатковий матеріал для статті в журналі. Інтернет-засіб для перегляду статей видавця журналу може вільно відображати мережу в інтерактивному режимі перегляду, що дозволяє користувачеві збільшувати масштаб мережі та переміщувати вузли.

Застосування

У біології:

– Завантаження наборів даних молекулярних та генетичних взаємодій у різних форматах.

– Потужні можливості візуалізації даних.

– Проектування та інтегрування наборів глобальних даних та функціональних анотацій.

– Виконання просунутого аналізу та моделювання за допомогою плагінів Cytoscape.

– Візуалізація та аналіз даних про метаболічні шляхи людини, такі як WikiPathways, Reactome та KEGG.

Комплексний аналіз мереж:

– Розрахунок статистики мереж такими плагінами як NetworkAnalyzer та CentiScaPe.

– Знаходження найкоротшого шляху.

– Знаходження кластерів.

– Можливість використання інших інструментів для більш просунутого аналізу.

У соціології:

– Візуалізація та аналіз великих соціальних мереж міжособистісних відносин.

– Складання соціальних мереж з таблиць та форм.

– Збір соціальних взаємодій з Інтернету за допомогою різноманітних API веб-сервісів.

– Розрахувати статистику мережі за допомогою плагінів.

– Можливість використання інших інструментів (R, NetworkX) для просунутого аналізу мережі.

Базові можливості

Основна частина Cytoscape є мережевим графом з молекулярними видами як вузли (вершини) і міжмолекулярні взаємодії як зв'язки (ребери) між вузлами. Cytoscape Core надає базову функціональність для інтеграції довільних даних на графі, візуального представлення графа та інтегрованих даних, інструментів виділення та фільтрації та інтерфейсу для зовнішніх методів, реалізованих у вигляді модулів, що підключаються.

Візуалізація мереж

У Cytoscape можливі різноманітні способи візуального представлення мереж (графів): циклічний, у вигляді дерева, force-directed тощо. Також користувач може організувати по-своєму аналізовану мережу. Накладені на мережу рівні експресії та значення p-value можуть бути відображені у вигляді кольору вершин або ребер, товщини або кольору крайових ліній тощо. Користувач може скористатися готовими схемами візуалізації та налаштувати їх самостійно.

Інтеграція даних

Дані інтегруються з графічною моделлю за допомогою атрибутів (Attributes). Це пари (ім'я, значення), які зіставляють імена вузлів чи ребер певним значенням даних. Значення атрибута можуть приймати будь-який тип (наприклад, текстові рядки, дискретні чи безперервні числа, URL-адреси або списки) або завантажуються зі сховища даних, або генеруються динамічно в сеансі. Графічні браузери дозволяють користувачеві переглядати всі атрибути вибраних вузлів та ребер.

Передача анотацій

У той час як атрибут є предикатом вузла або ребра, Annotation представляє ієрархічну класифікацію (тобто формально орієнтований граф без циклів) описів груп вузлів або ребер. Анотації зазвичай відповідають існуючому сховищу знань, яке є великим, складним та відносно статичним. Cytoscape об'єднує анотації з іншими типами мережевих даних шляхом передачі бажаних рівнів анотації на атрибути вузлів або ребер. Використовуючи контролер анотацій, можна мати безліч рівнів анотації активними і відображаються одночасно, кожен як окремий атрибут на потрібних вузлах або ребрах.

Візуалізація графа

Одним із найбільш фундаментальних інструментів для інтерпретації даних молекулярної взаємодії є візуалізація вузлів та ребер у вигляді двовимірної

мережі. Cytoscape підтримує безліч алгоритмів автоматизованого компонування мережі, включаючи ієрархічну та кругову компонування.

Візуалізація атрибутів

Візуалізація таких атрибутів як експресія генів та p-value. Cytoscape підтримує широкий спектр візуальних властивостей, таких як колір, форма та розмір вузла, колір та товщина межі вузла, колір краю, товщина та стиль. Візуалізація атрибуту відбувається з використанням таблиці пошуку чи інтерполяції залежно від цього, є атрибут дискретним чи безперервним.

Пошук та фільтрація частин графа

Щоб зменшити складність мережі з великою молекулярною взаємодією, необхідно відображати підмножини вузлів та ребер вибірково. Вузли та ребра можуть бути обрані відповідно до широкого ряду критеріїв, включаючи вибір на ім'я, за списком імен або на основі атрибуту. Більш складні запити вибору мережі підтримуються за допомогою набору інструментів фільтрації, який включає фільтр мінімальних сусідніх вузлів, що вибирає вузли з мінімальною кількістю сусідів на заданій відстані мережі; фільтр локальної відстані, який вибирає вузли на заданій відстані від групи попередньо вибраних вузлів; комбінований фільтр, який вибирає вузли довільно та/або за допомогою комбінацій інших фільтрів та інші. Cytoscape дозволяє шукати вершин або ребер за їх назвами.

Виділення вершин або ребер

Користувач може виділити підсіти вершин та/або ребер, які мають будь-які властивості. Наприклад, користувач може вибрати всі вершини, що мають ступінь більше встановленого порога, мають певну функціональну анотацію, або всі вершини, що позначають гени, чий рівень експресії сильно змінився хоча б в одному експерименті у відповідність до p-value, завантаженими разом з даними про рівні експресії . Користувач може створити нову мережу, виділивши частину попередньої.

Пошук модулів та кластерів

При дослідженні мереж взаємодії генів Cytoscape уможливлює пошук окремих ділянок, які складаються з генів з високою експресійною активністю. Більш того, в будь-якому об'єкті, що вивчається, можна здійснювати пошук ділянок з високою зв'язністю елементів, або кластерів.

Підтримка багатьох форматів

Cytoscape підтримує багато стандартних форматів, що передають взаємодії молекул та їх анотацію: SIF (Simple Interaction Format), GML, XGMML, BioPAX, PSI-MI, GraphML, KGML (KEGG XML), SBML, OBO та Gene Association. Текстові файли з роздільниками та формат Microsoft Excel підтримуються програмою. Користувач може імпортувати файли, що містять дані про рівні експресії генів та GO анотацію, завантажувати та зберігати довільні атрибути на вершинах та ребрах мережі (графа). Наприклад, до вершин, що позначають білки, може бути приписана їх функція, а до ребрів, що позначають взаємодії між білками, достовірність цих взаємодій, цю інформацію можна отримати з бази даних STRING.

Функціональна сумісність

Через те, що Cytoscape підтримує широкий спектр форматів, його легко можна поєднувати з іншими програмами. Наприклад, якщо користувач працював із мережами у програмах igraph або Bioconductor, Cytoscape може завантажити файли видачі цих програм, провести аналіз та зберегти результати, наприклад, у форматі PSI-MI, який далі може бути використаний для обробки іншими біоінформатичними програмами чи скриптами.

Зв'язок із зовнішніми базами даних

Cytoscape здатний безпосередньо підключатися до сторонніх баз даних, завантажувати мережі та анотацію. Приклади баз даних, що використовуються: Pathway Commons, IntAct, BioMart, NCBI Entrez Gene.

Збереження сесії

Поточний стан роботи може бути збережений у вигляді файлу сесії, який включає всі налаштування, аналізовані мережі, їх візуалізацію, стилі, стан робочого вікна, плагінів та ін. Файл сесії має розширення Cytoscape Session.

Збереження зображень

Cytoscape дозволяє зберігати зображення у високій якості. Підтримує такі формати: PDF, PS, SVG, PNG, JPEG та BMP.

Перегляд

Перегляд мереж у Cytoscape полегшений можливістю збільшення та зменшення зображень, їх прокручування, ручного редагування. Щоб спростити роботу з величезними мережами (наприклад, що містять понад 100 000 вершин або ребер), є вікно виду з пташиного польоту.

Менеджер додатків та Cytoscape App Store

До цієї програми доступні програми для вивчення мереж та молекулярних профілів. Cytoscape створено на платформі Java, тому можна створювати додаткові програми для імпорту даних, їх аналізу та візуалізації. Безліч програм доступні на Cytoscape App Store. Більшість програм можна встановлювати за допомогою менеджера App Manager або безпосередньо з App Store.

Плагіни Cytoscape

Модулі, що підключаються, являють собою потужні засоби розширення можливостей для реалізації нових алгоритмів, додаткового мережевого аналізу та біологічної семантики. Модулі, що підключаються, отримують доступ до мережної моделі Core і можуть також керувати відображенням мережі. Хоча Cytoscape Core – це відкрите програмне забезпечення, плагіни – це окреме програмне забезпечення, яке може бути захищене будь-якою ліцензією залежно від авторів плагінів.

Legend Creator

Плагін для створення легенд у Cytoscape. Він додає панель керування, яка може сканувати мережу та таблицю стилів для створення легенди для кожної

характеристики. Його основною функцією є додавання колірного градієнта визначення кількісного значення атрибутів.

GNC

GNC – новий плагін Cytoscape для оцінки біологічної когерентності генних мереж шляхом порівняння зі стандартною. GNC плагін використовує алгоритм GNC для оцінки біологічної когерентності генних мереж. Плагін був інтегрований у Cytoscape для збільшення доступності алгоритму для користувачів. Ця інтеграція дозволила користувачеві аналізувати як глобальну біологічну узгодженість мережі, а й біологічну узгодженість лише на рівні взаємовідносин генів. Це дозволяє користувачеві використовувати Cytoscape для подальшого аналізу та візуалізації мереж.

ReNE

ReNE - плагін Cytoscape 3.x розроблений для автоматичного збагачення стандартної регуляторної мережі, створеної на основі генів з більш докладними транскрипційними, посттранскрипційними та трансляційними даними. В результаті виходить розширена мережа, яка точніше моделює фактичні біологічні регуляторні механізми. ReNE може автоматично імпортувати мережевий макет з Reactome або KEGG або працювати з користувачами, описаними з використанням стандартного формату даних OWL / XML, який приймає процедура імпорту Cytoscape. Крім того, ReNE дозволяє дослідникам об'єднувати кілька шляхів, що надходять із різних джерел. Розширена мережа, що виходить в результаті, як і раніше, є повністю функціональною мережею Cytoscape, де кожен регуляторний елемент (фактор транскрипції, miRNA, ген, білок) і регуляторний механізм (підвищуюча регуляція / знижувальна регуляція) чітко візуально ідентифікуються, що дозволяє краще візуально зрозуміти їх роль впливом геть поведінка мережі. Удосконалена мережа, створена ReNE, експортується до різних форматів для подальшого аналізу через сторонні програми. ReNE розширює мережу, інтегруючи дані лише з загальнодоступних джерел, без висновків чи прогнозів.

NOA

NOA - це плагін Cytoscape, який використовується для аналізу онтології мереж. Реалізований алгоритм NOA заснований на збагаченні мереж шляхом розширення анотацій Gene Ontology до посилань на мережі або грані графа. Цей плагін полегшує аналіз однієї чи кількох мереж Cytoscape відповідно до заданих користувачем параметрів. Плагін представляє результати у вигляді таблиць, а також формує теплові карти та складає огляд мереж із Cytoscape.

ClusterMaker

ClusterMaker - плагін Cytoscape реалізує кілька алгоритмів кластеризації та візуалізації, які можна використовувати незалежно або в комбінації для аналізу та візуалізації наборів біологічних даних, а також для підтвердження або створення гіпотезобіологічної функції. Плагін надає результати у вигляді мережі, дендрограми та теплової карти.

CytoCluster

CytoCluster – плагін Cytoscape для кластерного аналізу біологічних мереж. CytoCluster поєднує шість алгоритмів кластеризації. А саме: HC-PIN (алгоритм ієрархічної кластеризації мереж взаємодій білків), OH-PIN (ідентифікація перекриваються та ієрархічних модулів мереж взаємодій білків), IPCA (алгоритм ідентифікації комплексних білків), ClusterONE (кластеризація з розширенням сусідів, що перекриваються), комплексів на основі моделі невизначеного графа), IPC-MCE (ідентифікація білкових комплексів на основі комплексу максимального розширення) та BinGO (онтологія генів біологічних мереж). Користувач може вибрати будь-який із перерахованих алгоритмів кластеризації відповідно до своїх вимог. Основна функція цих шести алгоритмів кластеризації полягає у виявленні білкових комплексів чи функціональних модулів. Більш того, BinGO можна використовувати для визначення того, які категорії Gene Ontology (GO) статистично представлені кілька разів у наборі генів або підграфі біологічної мережі.

StringApp

STRING є одним із найпопулярніших джерел білкових мереж, проте її веб-інтерфейс в основному призначений для перевірки невеликих мереж та відповідних доказів. Програмне забезпечення Cytoscape набагато краще підходить для роботи з великими мережами та пропонує більшу гнучкість у плані мережевого аналізу, імпорту та візуалізації додаткових даних. У зв'язку з цим було створено плагін stringApp, що поєднує Cytoscape STRING. І тим самим спрощує імпорт мереж STRING у Cytoscape, зберігає зовнішній вигляд та багато функцій STRING, а також інтегрує дані зі зв'язаних баз даних.

CyClust3D

CyClust3D - плагін для кластеризації мотивів мереж інтегрованих у мережі. Традиційні алгоритми кластеризації графів що неспроможні виявити щільні топологічні структури чи функціональні модулі, у яких збираються мотиви мереж інтегровані молекулярні мережі. Плагін CyClust3D дозволяє виявляти такі модулі за допомогою кластеризації складових тривузлових мережевих мотивів з використанням алгоритму тривимірної спектральної кластеризації.

PiNGO

PiNGO – плагін Cytoscape для пошуку генів кандидатів у біологічні мережі. PiNGO є інструментом для скринінгу біологічних мереж на наявність генів-кандидатів, тобто генів, які, як передбачають, будуть залучені в біологічний процес, що цікавить. Користувач може звузити пошук генів з конкретними відомими функціями або виключити гени, що належать до певних функціональних класів. PiNGO забезпечує підтримку широкого спектру організмів та схем класифікації генної онтології, і його можна легко налаштувати для інших організмів та функціональних класифікацій.

Pajek

Pajek, PajekXXL і Pajek3XL — це програмні пакети для аналізу та візуалізації дуже великих мереж (мережі, що містять до десяти мільярдів вершин,

обмежень на кількість рядків немає, крім розміру пам'яті). Pajek доступний з 1996 року. У Pajek аналіз і візуалізація великих мереж виконуються з використанням шести типів даних (об'єктів): мережа (граф); розбиття (номінальні або порядкові властивості вершин); вектор (числові властивості вершин); кластер (підмножина вершин); перестановка (перевпорядкування вершин, порядкові властивості); та ієрархія (загальна структура дерева на вершинах). Основними цілями в дизайні Pajek є: підтримувати абстракцію шляхом (рекурсивного) розкладання великої мережі на кілька менших мереж, які можна обробляти далі за допомогою більш складних методів; надати користувачеві деякі потужні засоби візуалізації; реалізувати вибір ефективних (субквадратичних) алгоритмів для аналізу великих мереж. Згідно з основними цілями Pajek містить кілька основних операцій над своїми об'єктами. Pajek не є «програмою в один клік», деякі користувачі називають її «мережевим калькулятором». Це означає, що для отримання якогось результату необхідно послідовно виконати декілька основних операцій. Насправді можливість комбінувати різні основні операції надає Pajek особливої сили. Крім стандартних Pajek існують також спеціальні версії PajekXXL і Pajek3XL. PajekXXL і Pajek3XL — це спеціальні версії програми Pajek, споживання пам'яті яких значно менше. Верхня межа PajekXXL становить два мільярди, а Pajek3XL — десять мільярдів вершин. Для такої ж розрідженої мережі їм потрібно принаймні в 2–3 рази менше фізичної пам’яті, ніж Pajek. Тому операції, які інтенсивно потребують пам’яті, виконуються набагато швидше. Зазвичай вони використовуються для великих мереж, які не вміщуються в доступну пам’ять комп’ютера.

NodeXL

NodeXL — пакет програмного забезпечення для аналізу та візуалізації мережі для Microsoft Excel 2007/2010/2013/2016. Пакет схожий на інші інструменти візуалізації мережі, такі як Pajek, UCINet і Gephi. Він широко використовується в кільцевих, відображеннях вершин і ребер, а також настроюваних візуальних атрибутах і тегах. NodeXL дозволяє дослідникам

аналізувати такі показники роботи соціальних мереж, як центральність, ступінь і кластеризація, а також контролювати реляційні дані та описувати загальну структуру реляційної мережі. У застосуванні до аналізу даних Twitter він показав загальну мережу всіх користувачів, які беруть участь у публічному обговоренні, і її внутрішню структуру за допомогою аналізу даних. Це дозволяє аналізу соціальних мереж (SNA) акцентувати увагу на взаємозв'язках, а не на ізольованих особах чи організаціях, дозволяючи зацікавленим сторонам досліджувати двосторонній діалог між організаціями та громадськістю. SNA також надає гнучку систему вимірювання та вибір параметрів для підтвердження впливових вузлів у мережі, таких як центральність за ступенем і за ступенем. Програмне забезпечення містить візуалізацію мережі, функції аналізу соціальних мереж, доступ до імпортерів даних соціальних мереж, розширені мережеві показники та автоматизацію.

Особливості

NodeXL призначений для користувачів з невеликим або без досвіду програмування, щоб дозволити їм збирати, аналізувати та візуалізувати різноманітні мережі. NodeXL інтегрується в Microsoft Excel 2007, 2010, 2013, 2016, 2019 і 365 і відкривається як робоча книга з різноманітними робочими аркушами, що містять елементи структури графіка, такі як ребра та вузли. NodeXL також може імпортувати різноманітні формати графіків, такі як списки країв, матриці суміжності, GraphML, UCINet .dl і Pajek .net.

Імпорт даних

NodeXL Pro здатний імпортувати файли UCINet і GraphML, а також електронні таблиці Excel, які містять списки країв або матриці суміжності, у робочі книги NodeXL. Крім того, NodeXL Pro пропонує набір інструментів імпорту, які можуть швидко збирати дані соціальних медіа з різних джерел, включаючи електронну пошту, Twitter, YouTube і Flickr. Перш ніж збирати будь-які дані, NodeXL запитує дозвіл користувача та зосереджує свої зусилля зі збору даних на загальнодоступній інформації, такій як статуси в Твіттері та зв'язки

підписників для осіб, які вирішили зробити свої облікові записи загальнодоступними. Ці атрибути дають змогу користувачам NodeXL ефективно аналізувати відповідні дані соціальних медіа та оптимізувати збір і аналіз даних соціальних мереж за допомогою одного інструменту.

Представлення даних

Робочі книги NodeXL містять чотири аркуші: ребра, вершини, групи та загальні показники. Відповідні дані про сутності на графіку та зв'язки між ними розташовані на відповідному аркуші у форматі рядка. Наприклад, робочий аркуш ребер містить щонайменше два стовпці, а кожен рядок містить щонайменше два елементи, що відповідають двом вершинам, які утворюють ребро в графі. Метрики графіка та візуальні властивості ребер і вершин відображаються як додаткові стовпці у відповідних робочих аркушах. Це представлення дозволяє користувачеві використовувати електронну таблицю Excel для швидкого редагування існуючих властивостей вузла та створення нових, наприклад, шляхом застосування формул Excel до існуючих стовпців.

Аналіз графів

NodeXL Pro містить бібліотеку часто використовуваних метрик графа: центральність, коефіцієнт кластеризації та діаметр. NodeXL розрізняє спрямовані та ненаправлені мережі. NodeXL Pro реалізує різноманітні алгоритми виявлення спільноти, щоб дозволити користувачеві автоматично виявляти кластери у своїх соціальних мережах.

Візуалізація графіка

NodeXL створює інтерактивне полотно для візуалізації графіків. Проект дозволяє користувачам вибирати з кількох добре відомих алгоритмів компонування графіків, спрямованих силою, таких як Fruchterman-Reingold і Harel-Koren. NodeXL дозволяє користувачеві вибирати, перетягувати та опускати вузли на полотно та вручну редагувати їхні візуальні властивості (розмір, колір і непрозорість). Крім того, NodeXL дозволяє користувачам зіставляти візуальні

властивості вузлів і ребер з показниками, які він обчислює, і взагалі з будь-яким стовпцем на аркуші ребер і вершин.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Unity Technologies розробила кросплатформний ігровий движок, що отримав назву Unity. Цей ігровий двигун вперше був представлений на Всесвітній конференції розробників Apple у червні 2005 року як ігровий двигун для Mac OS X. З того часу Unity поступово розширювався, щоб підтримувати широкий спектр платформ, включаючи настільні комп'ютери, мобільні пристрої, консолі та платформи віртуальної реальності.

Особливо популярність Unity здобув у розробці мобільних ігор для iOS та Android. Він вважається простим у використанні для початківців, а також є улюбленим інструментом для розробки незалежних ігор. Unity надає безліч можливостей для створення як тривимірних, так і двовимірних ігор, а також використовується для інтерактивного моделювання та інших захоплюючих проектів. Завдяки своїй неперервній еволюції та інноваціям Unity залишається провідним інструментом у галузі розробки ігор.

Unity надає можливість користувачам створювати як 2D, так і 3D ігри, а його механізм пропонує основний API сценаріїв на базі мови програмування C#, використовуючи Mono як редактор Unity та вбудований у самі ігри. Крім того, Unity забезпечує зручну функцію перетягування для реалізації функціональності.

Раніше Unity підтримував мову програмування Boo, яка була використовувана, перш ніж C# стала основною мовою програмування для двигуна. З випуском Unity 5 мова Boo була вилучена, а також застаріла реалізація JavaScript на основі Boo, відома як UnityScript, була офіційно відмінена в серпні 2017 року після випуску Unity 2017.1. Замість цього Unity активно використовує мову програмування C# як свою основну мову для розробки.

У сфері 2D-ігор, Unity забезпечує можливість імпортувати спрайти та використовувати розширені засоби візуалізації 2D-світу. Щодо 3D-ігор, Unity дозволяє налаштовувати різні параметри текстурного стиснення, MIP-карт та роздільної здатності для кожної платформи, що підтримується цим ігровим двигуном. Крім того, Unity надає широкі можливості відображення рельєфу, відображення світла, паралаксу, ефектів ambient occlusion (SSAO), динамічних тіней за допомогою тіньової карти, рендерингу до текстури та повноекранної постобробки для створення вражаючих візуальних ефектів.

Unity пропонує два різні конвеєри візуалізації, які включають High Definition Render Pipeline (HDRP) і Universal Render Pipeline (URP, раніше відомий як LWRP), разом зі застарілим вбудованим конвеєром. Важливо зазначити, що ці три конвеєри візуалізації несумісні один з одним. Однак Unity надає інструмент для оновлення шейдерів, які використовують застарілий рендерер, до URP або HDRP, що дозволяє користувачам покращити графічну якість своїх проектів.

Unity є мультиплатформовим двигуном. Редактор Unity підтримується на операційних системах Windows, macOS і Linux, а сам двигун дозволяє розробляти ігри для понад 19 різних платформ, включаючи мобільні пристрої, настільні комп'ютери, консолі і віртуальну реальність. Unity 2020 LTS офіційно підтримує наступні платформи:

– Мобільні платформи: iOS, Android (включаючи Android TV), tvOS;

– Настільні платформи: Windows (включаючи Universal Windows Platform), macOS, Linux;

– Веб-платформа: WebGL;

– Консольні платформи PlayStation (PS4, PS5), Xbox (Xbox One, Xbox Series X/S), Nintendo Switch, Stadia;

– Платформи віртуальної/розширеної реальності: Oculus, PlayStation VR, ARCore від Google, ARKit від Apple, Windows Mixed Reality (HoloLens), Magic Leap і, за допомогою Unity XR SDK, Steam VR і Google Cardboard.

Unity раніше був використовуваний як SDK за замовчуванням для відеоігрової консолі Nintendo Wii U. Компанія Nintendo додавала безкоштовну копію Unity до кожної ліцензії розробника Wii U, і Unity Technologies назвала це "першим у галузі" об'єднанням стороннього SDK. Упродовж перших десяти років свого існування Unity мав платні версії, але в 2016 році компанія перейшла на модель передплати. Unity пропонує безкоштовні та платні варіанти ліцензування. Безкоштовна ліцензія призначена для особистого використання або невеликих компаній з річним доходом менше 100 000 доларів США (пізніше збільшено до 200 000 доларів США), тоді як платні підписки базуються на доходах, отриманих від ігор, створених з використанням Unity. Unity Pro, платний варіант, був доступний для розробників з річним доходом понад 200 000 доларів США, а також для розробників консольних ігор за ліцензією від виробників консолей. Unity Pro ключі були частиною іншого SDK від виробників консолей, за який розробники платили. Однак, з червня 2021 року Unity вимагає від розробників, які створюють ігри для закритих консольних систем, таких як PlayStation, Nintendo Switch і Xbox, мати ліцензію Unity Pro або ключ Preferred Platform License Key, незалежно від їх доходів. Sony і Nintendo надають це як частину свого SDK, але Microsoft ще не впровадила цю функцію в своє SDK. Вихідний код двигуна Unity ліцензується "на окремій основі за спеціальними домовленостями".

Протягом 2010-х років Unity Technologies розширила своє використання ігрового двигуна на інші галузі, використовуючи платформу 3D в реальному часі, зокрема кіно і автомобільну промисловість. Unity вперше експериментував у кіно з короткометражним фільмом "Адам", де розповідалася історія робота, що втікає з в'язниці. Пізніше Unity співпрацював з режисером Нілом Бломкампом, чия студія Oats використовувала інструменти двигуна, включаючи рендеринг в реальному часі та Cinemachine, для створення двох комп'ютерно згенерованих короткометражних фільмів: "Адам: Дзеркало" та "Адам: Пророк". На конференції Unite Europe 2017 в Амстердамі Unity акцентувала увагу на створенні фільмів за

допомогою нового інструменту Cinemachine в Unity 2017.1. В 2018 році Disney Television Animation випустила три короткометражки під назвою "Baymax Dreams", створені з використанням двигуна Unity. Двигун Unity також використовувався компанією Disney для створення фону для фільму "Король Лев" у 2019 році.

Автовиробники використовують технологію Unity для створення повномасштабних моделей нових транспортних засобів у віртуальній реальності, створення віртуальних складальних ліній та навчання працівників. Двигун Unity також використовується компанією DeepMind, що належить Alphabet Inc., для навчання штучного інтелекту. Інші сфери використання Unity Technologies включають архітектуру, проектування та будівництво.

C# - це мова програмування високого рівня загального призначення, яка підтримує кілька парадигм. Вона має статичну і жорстку типізацію, лексичну область видимості та підтримує імперативну, декларативну, функціональну, загальну, об'єктно-орієнтовану (на основі класів) та компонентно-орієнтоване програмування.

Андерс Хейлсберг з Microsoft розробив мову програмування C# у 2000 році. Вона була прийнята як міжнародний стандарт Ecma (ECMA-334) у 2002 році та ISO/IEC (ISO/IEC 23270) у 2003 році. Початково C# була представлена разом з .NET Framework і Visual Studio, які були пропрієтарними продуктами, оскільки в той час Microsoft не мав відкритих продуктів. Проте, через чотири роки, у 2004 році, розпочався проект з відкритим вихідним кодом під назвою Mono, який надавав кросплатформений компілятор і середовище виконання для C#. За десять років Microsoft випустила безкоштовні продукти з відкритим кодом, такі як Visual Studio Code (редактор коду), Roslyn (компілятор) і уніфіковану платформу .NET (фреймворк програмного забезпечення), які підтримують C# та є кросплатформенними. Mono також став частиною Microsoft, але не був об'єднаний з .NET.

Станом на листопад 2022 року найактуальнішою стабільною версією мови C# є C# 11.0, яка була випущена в рамках .NET 7.0 у 2022 році.

Під час розробки бібліотеки класів .NET Framework спочатку використовувалась система компілятора, яку називали "Simple Managed C" (SMC). У січні 1999 року Андерс Хейлсберг створив команду для розробки нової мови, що отримала назву Cool, що означала "C-подібна об'єктно-орієнтована мова". Компанія Microsoft розглядала можливість залишити цю назву як остаточну для мови, але вирішила змінити її з міркувань торгової марки. Коли проект .NET був офіційно оголошений на конференції для розробників у липні 2000 року, мову було перейменовано на C#, а бібліотеки класів та середовище виконання ASP.NET були переведені на C#.

Андерс Хейлсберг є головним розробником мови C# та провідним архітектором в компанії Microsoft. Він також брав участь у розробці Turbo Pascal, Embarcadero Delphi (раніше відома як CodeGear Delphi, Inprise Delphi та Borland Delphi) та Visual J++. У своїх інтерв'ю та технічних документах він висловлював думку, що недоліки багатьох основних мов програмування, таких як C++, Java, Delphi та Smalltalk, стимулювали створення основи Common Language Runtime (CLR), що з свого боку спонукало до розробки мови C#.

Джеймс Гослінг, який створив мову програмування Java у 1994 році, та Білл Джой, співзасновник Sun Microsystems та автор Java, називали C# «імітацією» Java. Потім Гослінг зазначив, що «C# є різновидом Java, але без надійності, продуктивності та безпеки». Клаус Крефт і Анжеліка Лангер, автори книги про потоки C++, в своєму блозі заявили, що «Java і C# є практично ідентичними мовами програмування. Це нудне повторення без інновацій», «Малоймовірно, що хтось стверджуватиме, що Java або C# є революційними мовами програмування, що змінили спосіб написання програм», і «C# позичив багато з Java, і навпаки. Тепер, коли C# підтримує упаковку та розпакування, у нас буде дуже схожа функція в Java». У липні 2000 року Хейлсберг заявив, що C# «не є клоном Java» і що вона «набагато ближча до C++» за своїм дизайном.

Починаючи з релізу C# 2.0 у листопаді 2005 року, мови C# та Java почали рухатися в різних напрямках, перетворившись на дві зовсім відмінні мови. Одне з найбільш помітних відхилень сталося при додаванні генериків до обох мов, але з використанням різних реалізацій. C# використовує реифікацію для створення "першокласних" загальних об'єктів, які можна використовувати так само, як будь-який інший клас, з генерацією коду, що відбувається під час завантаження класу.

Крім того, C# додав кілька важливих функцій для програмування у функціональному стилі. В одному з найяскравіших виявів цього напрямку стали розширення LINQ, які були випущені з версії C# 3.0, а також введення лямбда-виразів, методів розширення та анонімних типів. Ці функції дозволяють розробникам C# використовувати функціональне програмування, наприклад, замикання, коли це необхідно. Розширення LINQ та функціональний імпорт допомагають зменшити кількість шаблонного коду, який зазвичай використовується для таких рутинних завдань, як запити до баз даних, синтаксичний аналіз файлів XML або пошук у структурах даних. Це дозволяє акцентувати увагу на реальній логіці програми і покращує її читабельність та обслуговування.

.NET Framework є патентованою програмною основою, розробленою компанією Microsoft, яка головним чином використовується в операційній системі Microsoft Windows. Спочатку воно було основною реалізацією загальної мовної інфраструктури (CLI), поки його не замінив багатоплатформний проект .NET. Фреймворк включає в себе обширну бібліотеку класів, відому як Framework Class Library (FCL), і забезпечує можливість використання мов програмування взаємодіючих між собою (тобто, код, написаний однією мовою, можна використовувати іншими мовами програмування). Програми, розроблені для .NET Framework, виконуються в середовищі виконання програм, відомому як Common Language Runtime (CLR). CLR є віртуальною машиною програми, яка надає різні служби, такі як безпека, управління пам'яттю і обробка виключень.

Таким чином, код, написаний з використанням .NET Framework, відомий як "керований код". FCL і CLR утворюють разом .NET Framework.

FCL надає функціональність для інтерфейсу користувача, доступу до даних, роботи з базами даних, криптографії, розробки веб-додатків, числових алгоритмів і мережевої комунікації. Розробники створюють програмне забезпечення, комбінуючи вихідний код з .NET Framework та іншими бібліотеками. Фреймворк призначений для використання в більшості нових програм, розроблених для платформи Windows. Крім того, Microsoft надає інтегроване середовище розробки під назвою Visual Studio для програмного забезпечення .NET.

.NET Framework початково був пропрієтарним програмним забезпеченням, але компанія вже майже з самого початку працювала над стандартизацією стеку програмного забезпечення, навіть до його першого випуску. Незважаючи на ці зусилля щодо стандартизації, розробники, особливо з вільного та відкритого програмного забезпечення, висловили своє занепокоєння щодо умов та перспектив вільної реалізації з відкритим кодом, особливо щодо патентів на програмне забезпечення. Відтоді компанія Microsoft змінила підхід до розробки .NET з метою кращого відповідання сучасній моделі проектування програмного забезпечення, розробленого спільнотою. Наприклад, вони випустили оновлення своєї патентної політики, обіцяючи вирішити ці проблеми.

У квітні 2019 року Microsoft випустила останню версію .NET Framework 4.8 як свою власну пропозицію. Після цього були випущені лише щомісячні виправлення помилок безпеки та надійності для цієї версії. Наразі не планується внесення подальших змін у цю версію.

2.3 Розгорнута постановка завдання

Відповідно до технічного завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, планується розробити програмне

забезпечення для системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

У процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти слід виконати такі завдання:

а) провести аналіз існуючих аналогічних систем для виявлення їх позитивних та негативних характеристик. Результати аналізу будуть враховані у подальших розробках;

б) вибрати та обґрунтувати методику побудови системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення для системи, яке відповідатиме поставленим технічним вимогам. Побудувати блок-схеми алгоритмів та підпрограми програми;

г) створити користувацький інтерфейс для формування та виводу на екран повідомлень про некоректні дії користувача та нестандартні ситуації в роботі системи;

д) підготувати висновки щодо виконаних робіт та отриманих результатів.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Функціонування системи імітаційного моделювання розповсюдження інформації можна поділити на декілька етапів:

– виведення інтерфейсу користувача на екран;

– створення соціального графу;

– візуалізація соціального графу;

– симуляція розповсюдження інформації у даному графі.

Етап виведення інтерфейсу користувача є важливим кроком у розробці системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах, оскільки він визначає спосіб візуалізації програми та створення головного вікна, яке слугує основним інтерфейсом взаємодії з користувачем.

На цьому етапі розробник створює головне вікно програми, яке містить елементи інтерфейсу, такі як кнопки, панелі, меню та інші елементи управління. Головне вікно дозволяє користувачеві взаємодіяти з системою, налаштовувати параметри симуляції, переглядати результати та керувати процесом моделювання.

При проектуванні головного вікна важливо враховувати принципи зручності, ефективності та естетичного оформлення. Головне вікно повинно бути інтуїтивно зрозумілим для користувача, забезпечувати зручний доступ до основних функцій системи та мати привабливий зовнішній вигляд.

Після створення головного вікна, розробник може продовжувати роботу над іншими компонентами інтерфейсу, додавати додаткові функції та покращувати зовнішній вигляд програми. Етап виведення інтерфейсу користувача є початковою точкою для подальшої роботи з інтерфейсом, його

налаштування та удосконалення з метою забезпечення зручності та задоволення користувачів під час взаємодії з системою імітаційного моделювання.

Головне вікно складається із таких блоків як:

– панель генерації графу;

– панель симуляції;

– робоча область;

– блок вибору режимів відобрадення віршин.

Панель генерації графу містить компоненти:

– Кількість учасників соц. мережі;

– Кількість зв’язків нового учасника;

– Метод відображеняя графу;

– Кнопка «Сгернерувати граф»;

– Флаг «Всі вершини графу статичні?».

Панель симуляції складається із панелі підготовки до симуляції та власне панелі керування симуляцією.

Панель підготовки до симуляції міснить компоненті:

– Часовий інтервал одного кроку симуляції;

– Кнопка «Почати симуляцію».

Панель керування висуляцією містить компоненти:

– Показник кроку симуляції;

– Показник кількості інформованих осіб;

– Показник кількості осіб, які поширюють інформацію;

– Показник середньої оцінки інформації у соціальній мережі;

– Кнопка «Плей»;

– Кнопка «Пауза»;

– Кнопка «Каступний крок».

Робоча область – це область головного вікна програми, у якій відбувається відображення соціального графу, а також візуалізація процесів симуляції розповсюдження інформаційного вірусу.

Блок вибору режиму відображення вершин представляє собою список із наступними елементами-режимами:

– Інформовані;

– Розповсюджують;

– Комунікабельність;

– Консерватизм;

– Оцінка інформації;

Етап створення соціального графу ц е важлива складова системи. На цьому етапі програма генерує граф соціальної мережі на основі параметрів, які ввів користувач. Генерування графу розпочинається, коли користувач заповнив усі поля та натичнув кнопку «Сгенерувани граф».

Генерація графу відбувається за алгоритмом Барабаши-Альберт.

Генерація соціального графу за моделлю Барабаши-Альберт є важливим інструментом у вивченні та аналізі соціальних мереж. Ця модель використовується для створення графу, що відображає взаємозв'язки між учасниками в соціальній мережі, де кожен учасник представляє суб'єкт, такий як людина або організація, а зв'язки відображають залежності або взаємодії між ними.

Модель Барабаши-Альберт базується на принципі " переважне приєднання", де нові учасники мережі надають перевагу з'єднанням з вже існуючими учасниками, які вже мають значну кількість зв'язків. Це означає, що чим більше зв'язків має учасник, тим більше шансів з'явитися новому учаснику мережі налагодити зв'язок з ним. Цей принцип відображає існуючу соціальну реальність, де популярні люди або організації мають більше можливостей встановлювати нові зв'язки.

Процес генерації соціального графу за моделлю Барабаши-Альберт полягає у послідовному додаванні нових учасників до графу із встановленням зв'язків до існуючих учасників. Це відбувається шляхом вибору випадкового учасника з урахуванням його ступеня (кількості зв'язків), де учасник з більшим

ступенем має більше шансів бути обраним. Після вибору учасника новий учасник графу додається із створенням нового зв'язку до обраного учасника.

Таким чином, процес генерації продовжується, додаючи нових акторів та граф та встановлюючи зв'язки до існуючих акторів з врахуванням їхнього ступеня. Цей процес створює граф з характерною структурою, де деякі актори мають значну кількість зв'язків, що відображає впливність та популярність деяких учасників у соціальній мережі.

Генерація соціального графу за моделлю Барабаши-Альберт має декілька особливостей, які роблять її корисною у дослідженнях соціальних мереж. По-перше, ця модель дозволяє відтворити явища "багатства привернення" та "багатства наявності", що спостерігаються в реальних соціальних мережах. Вона також забезпечує наявність "хабів" або вузлів з високим ступенем, які відіграють ключову роль у передачі інформації та впливі в мережі.

Крім того, модель Барабаши-Альберт дозволяє варіювати параметри генерації, такі як кількість нових зв'язків, доданих для кожного нового актора, що дозволяє досліджувати різні сценарії розвитку соціальних мереж. За допомогою цієї моделі можна досліджувати процес поширення інформації, впливу та динаміки в соціальних мережах, а також розуміти ключові вузли та структуру мережі.

У результаті, генерація соціального графу за моделлю Барабаши-Альберт створює мережу, що відтворює деякі характеристики реальних соціальних мереж. Цей інструмент дозволяє досліджувати та аналізувати властивості соціальних мереж, розуміти їх структуру та вивчати різні аспекти їхньої динаміки. Застосування моделі Барабаши-Альберт у генерації соціального графу відкриває широкі можливості для дослідження різних соціальних явищ, таких як поширення інформації, вплив та взаємодія учасників, формування груп та спільнот, поява впливових лідерів та багато іншого.

Ця модель також знаходить своє застосування в інших галузях, таких як дослідження Інтернету, транспортних мереж, біологічних систем та інших

складних систем. Вона дозволяє створювати графи з характерними властивостями без необхідності великої кількості вхідних даних із реальних систем.

Загалом, генерація соціального графу за моделлю Барабаши-Альберт є потужним інструментом для вивчення та аналізу соціальних мереж. Вона дозволяє реплікувати деякі властивості реальних соціальних мереж, досліджувати їхню структуру та взаємодію учасників. Використовуючи цю модель, дослідники можуть отримати нові уявлення про соціальні явища та розвинути стратегії управління та впливу в соціальних мережах.

Для виведення соціального графу на екран, користувач може обрати два методи:

– Виведення силовим методом;

– Виведення випадковим методом;

Виведення соціального графу силовим алгоритмом є одним з методів візуалізації та аналізу соціальних мереж. Цей алгоритм базується на фізичних принципах взаємодії між частинками і допомагає відобразити структуру та взаємозв'язки в графі на двовимірній площині.

Силовий алгоритм працює наступним чином. Кожен вузол (учасник соціальної мережі) представляється як точка на площині. Між вузлами розміщуються пружні засилля, що моделюють їхні взаємодії. Силові алгоритми використовують два основних види сил:

1. Відштовхуючі сили: Кожен вузол відштовхується від інших вузлів, які знаходяться ближче за певну відстань. Це допомагає уникнути перекриття між вузлами та розташувати їх рівномірно на площині.

2. Притягуючі сили: Вузли, що мають зв'язки між собою, притягуються один до одного. Це допомагає згруповувати вузли, які мають близькі зв'язки, та показувати їхні зв'язки в графічному вигляді.

Після встановлення початкових положень вузлів і початкових значень сил, алгоритм постійно перераховує сили та переміщує вузли, щоб досягти

стабільного розташування. Цей процес триває до досягнення рівноваги між силами.

Результатом застосування силового алгоритму є графічне представлення соціального графу, де вузли розташовані на площині таким чином, що відображають їхні взаємозв'язки та групування. За допомогою силового алгоритму можна виявити структурні особливості соціального графу, такі як центральні вузли, густину зв'язків, громади або підгрупи.

Після виведення соціального графу силовим алгоритмом можна здійснювати різноманітний аналіз та візуалізацію мережі. Наприклад, можна виділити важливі учасники, які мають багато зв'язків, або виявити вузли, що знаходяться в окремих групах або громадах. Також можна дослідити вплив різних факторів на структуру мережі шляхом виконання модифікацій та спостережень за змінами в графі.

Виведення соціального графу випадковим методом набагато ефективніше коли треба швидко створити соціальний граф на декілька тисяч учасників. Алгоритм цього методу полягає у випадковому розташуванні вершин графу у певній областасті. Перевага чього методу у швидкості, а недолік у складності детального аналізу такого графу.

Вершини графу можуть бути забарвлені у ріний колір в залежності від обраного режиму відображення вершин. Програма передбачає наступні кольорові схеми відображення вершин в залежності від обраного режиму:

– Режим «Інформовані» - вершина може буті забарвлена у 2 кольори: білий, якшо учасник не проінформований; жовтий, якщо учасник проінформований.

– Режим «Розповсюджують» - вершина може буті забарвлена у 2 кольори: білий, якщо учасник не розповсюджує інформацію на даному кроці; червоний, якщо учасник розповсюджує інформацію сусідам.

– Режим «Консерватизм» - вершина може бути забарвлена від білого до синього кольору, де білий означає, що учасник не спирається на власну думку

при оцінці інформації, а синій означає, що учасник дуже сильно враховує аласну думку при оцінці інформації.

– Режим «Комунікабельність» - вершина може бути забарвлена від білого до помаранчевого кольору, де білий означає, що учасник не дуже комунікабельний, а помаранчевий означає високий рівень комунікабельності учасника.

– Режим «Оцінка інформації» - віршина може бути забарвлена від червоного до білого і від білого до червоного кольорів, де червоний – різко неганивне ставлення, білий – нейтральне ставлення та зелений – позитивне ставлення.

Етап симуляції розповсюдження інформаційного вірусу у соціальній мережі починається після заповнення усіх полів панелі підготовки до симуляції та натиснення кнопки «Почати симуляцію».

На нульовому кроці користувач повинен обрати агентів, які будуть розповсюджувани інфрмацію у соціальній мережі.

На першому кроці відбуваються наступні дії:

– Программа розраховує показник поширення інформації на основі індивідуальних якостей агента розповсюдження інформації: оцінка інформації, комунікабельність, інформаваність.

– Якщо показник поширення інформації дорівнює істині – учасник передає інформацію усім учасникам, з якими спілкується. Окрім інформіції, учасник ділиться і власною думкою з приводу даної інформації.

– Далі програма розраховує поточну оцінку інформації для учасників які щойно її отрималі. Цей розрахунок відбувається на основі показника консерватизму, початкової думки учасника та соціально-інтеграційної сили – середнє значення усіх позитивних і негативних думок, які учасник отримав у даний момент часу.

– Крок симуляції завершено. Програма переходить до першого кроку.

3.2 Розробка структурної схеми

Структурна схема системи складається з наступних компонентів:

– менеджер застосунку;

– контролер інтерфейсу;

– контролер симуляції;

– програма генерації соціального графу;

– контролер відображення графу;

– програма розрахунку стану вершин;

– програма розрахунку стану ребер;

Менеджер застосунку

Компонент "менеджер застосунку" є одним з головних елементів будь-якої програмної системи, він є своєрідною шиною для передачі даних між всіма іншими компонентами системи. Основна функція цього компонента полягає у виконанні алгоритму основної програми та передачі даних між компонентами. Завдяки цьому компоненту, різні частини програмної системи можуть взаємодіяти між собою, обмінюватися даними та виконувати необхідні дії.

Окрім цього, компонент "менеджер застосунку" може мати інші функції, наприклад, контроль над даними, що передаються між компонентами, моніторинг працездатності та відлагодження системи. Він також може забезпечувати безпеку даних, перевіряти правильність введених даних та вирішувати проблеми, пов'язані зі збереженням інформації.

У свою чергу, розробка ефективного та надійного компонента "менеджер застосунку" є важливим етапом при створенні будь-якої програмної системи. Його якість та функціональні можливості можуть впливати на роботу всієї системи та визначати її продуктивність. Тому, під час розробки програмної системи, компонент "менеджер застосунку" потрібно уважно проробити та перевірити на наявність помилок та недоліків.

У підсумку, можна сказати, що компонент "менеджер застосунку" є невід'ємною частиною будь-якої програмної системи, яка забезпечує взаємодію між різними її елементами та передачу даних між ними. Його вірна реалізація може допомогти досягти високої продуктивності

Контролер інтерфейсу

Компонент "контролер інтерфейсу" є невід'ємною частиною більшості програмних систем. Основною функцією цього компоненту є забезпечення взаємодії між користувачем та програмою, включаючи прийом та обробку введення користувача та відображення результатів роботи програми на екрані.

Крім того, контролер інтерфейсу може мати інші функції, такі як перевірка правильності введення даних, валідація введених даних, забезпечення безпеки користувача та реалізація інших функцій, пов'язаних з інтерфейсом користувача. Компонент також відповідає за передачу даних між іншими компонентами програмної системи.

Один з ключових аспектів роботи контролера інтерфейсу полягає у забезпеченні зручного та ефективного інтерфейсу для користувачів програми. Успіх програми в значній мірі залежить від того, наскільки легко та зручно користувач може взаємодіяти з програмою та виконувати потрібні завдання. Тому розробники звертають особливу увагу на проектування та розробку інтерфейсу, щоб зробити його зрозумілим та легким у використанні для користувачів.

Контролер симуляції

Компонент «контролер симуляції» відповідає за виконання алгоритму симуляції розповсюдження інформації у соціальному графі.

Генератор соціального графу

Модель Барабаши-Альберт є однією з найпоширеніших моделей генерації випадкових графів, що використовуються в соціальних мережах. Ця модель базується на принципі "багатства прив'язок", який стверджує, що користувачі, які мають більше зв'язків, мають більше шансів на те, щоб утворити нові зв'язки.

У генераторі соціального графу на основі моделі Барабаши-Альберт, користувачі та їх зв'язки генеруються випадковим чином, з використанням заздалегідь визначених параметрів, які задає користувач. Ці параметри можуть включати кількість вузлів (користувачів) в графі, кількість зв'язків, що має кожен вузол, та кількість нових зв'язків, що додаються під час генерації графу.

Після введення параметрів, генератор соціального графу застосовує модель Барабаши-Альберт для створення графу. Кожен новий вузол додається з певною кількістю зв'язків до наявних вузлів в графі, залежно від їх ступеню (кількості наявних зв'язків). Цей процес повторюється до досягнення заданої кількості вузлів та зв'язків в графі.

Отже, генератор соціального графу на основі моделі Барабаши-Альберт може бути корисним інструментом для дослідження та аналізу соціальних мереж, а також для створення випадкових графів для тестування різних алгоритмів та програм.

Контролер відображення графу

Компонент «контролер відображення графу» отримує дані іершин да ребер із менеджера застосунку та відображає граф за допомогою Силового алгоритму. У цому компоненті для кожної вершини розраховується сила відштовхування та сила стягування. Також цей компонент безпосередньо відповідає за візуалізацію абстракції графу.

Розрахунок стану вершин

Цей компонент отримує дані із «менеджера застосунку» про вибраний користувачем режим відображення вершин та на підставі цього розраховує їх розмір та колір.

Розрахунок стану ребер

Цей компонент відповідає за своєчасне оновлення стану відображення ребер у візуалізації графу.

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

Функціональна схема системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах зображена на рисунку 3.2 та складається з наступних блоків:

– Блок управління додатком;

– Блок управління інтерфейсом;

– Блок управління симуляцією;

– Блок відображення графу;

Блок управління додатком керує усіма процесами, які відбуваються у додатку, та передачею даних між іншими блоками.

Блок управління інтерфейсом відповідає за коректне відображення інтерфейсу користувача та склаждається з блоків:

– Блок валідації даних;

– Блок передаці даних для генерації;

– Блок передачі даних для симуляції;

Блок управління симуляцією відповідає за коректний хід симуляції поширення інформаційного вірусу у соціальній мережі та складається з наступних блокив:

– Блок управління показниками симуляції;

– Блок розрахунку показника поширеняя інформації;

– Блок розрахунку поточної оцінки інформації;

– Блок поширення інформації;

Блок відображення графу виводить соціальний граф на екран враховючи параметри відображення такі, як метод відображення, режим відображення віршин, стабілізація вершин. Блок складається з наступних блоків:

– Блок виведення вершин;

– Блок виведення ребер;

– Блок керування режимами відображення графу;

– Блок стабілізації вершин.

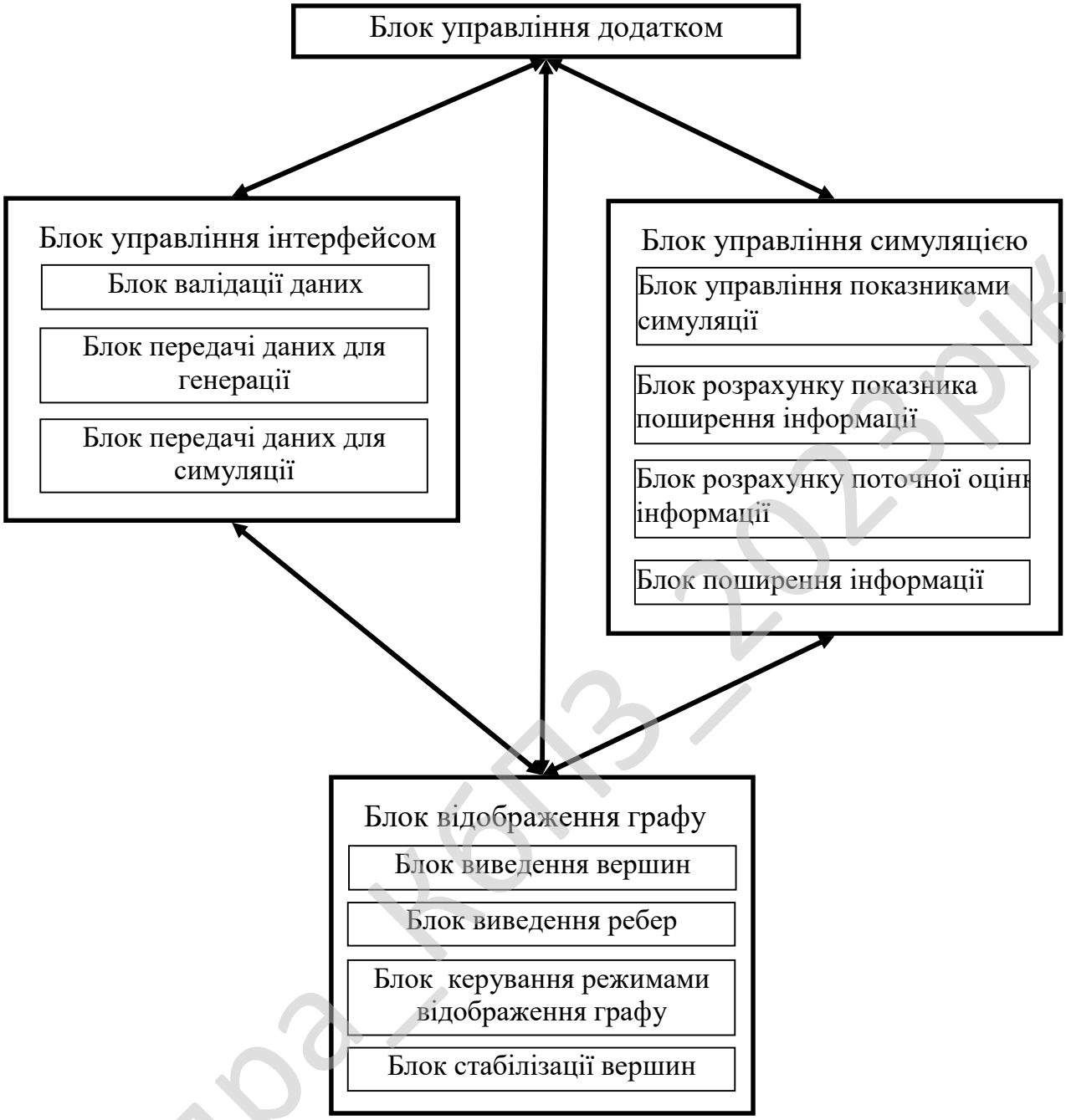

Рисунок 3.2 – Функціональна схема системи

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврської дипломної роботи, наведена на рисунку 3.3. З нього видно, що процеси взаємодіють наступним чином.

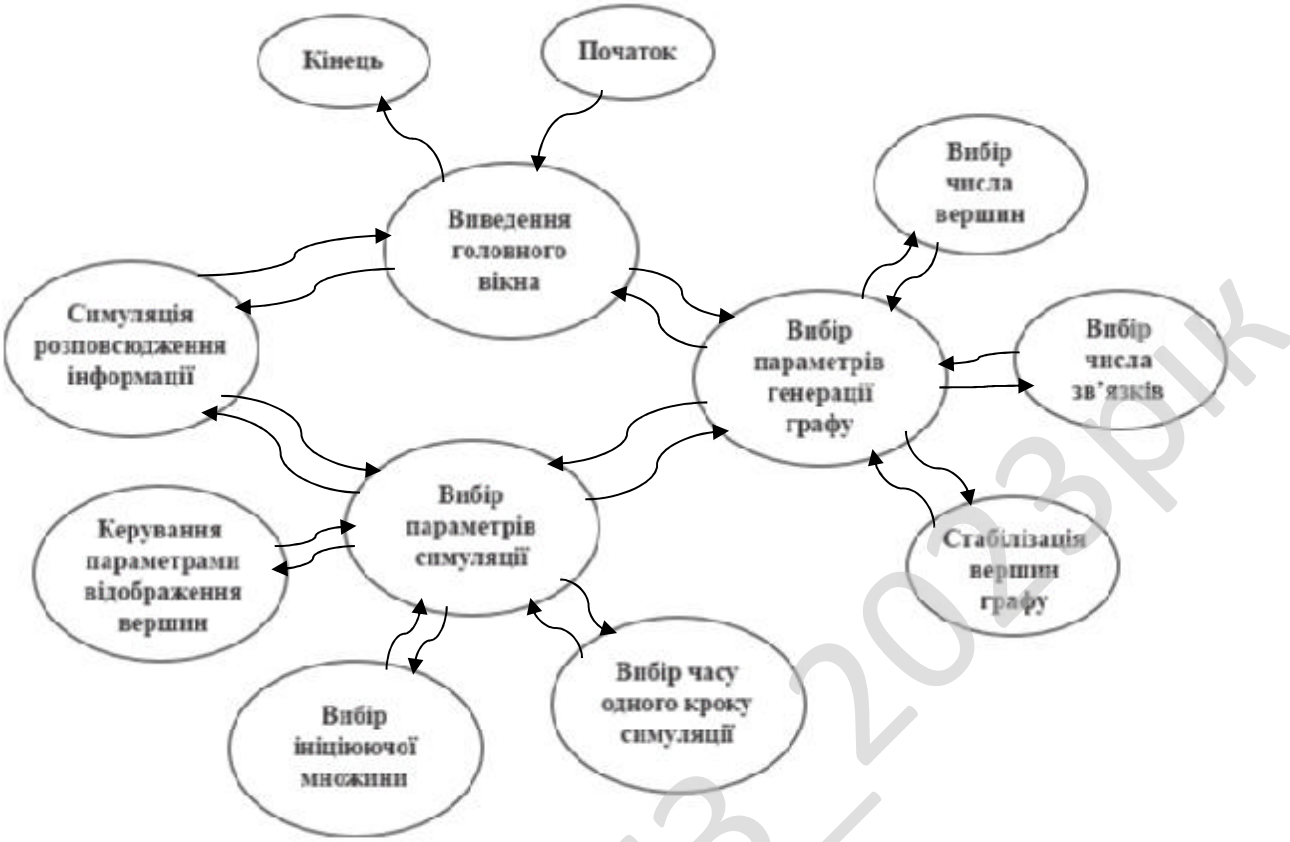

Рисунок 3.3 – Діаграма процесів системи

Процес «Початок» взаємодіє із процесом «Виведення головного вікна».

Процес «Виведення головного вікна» виконує важливу роль у візуалізації інтерфейсу користувача та симуляції розповсюдження інформації. Його основна функція полягає у створенні та відображенні головного вікна, яке забезпечує візуальну платформу для взаємодії користувача з системою імітаційного моделювання.

Цей процес включає в себе ряд дій, що спрямовані на створення інтерфейсу, який дозволяє користувачу легко взаємодіяти з системою імітаційного моделювання. Він включає у себе розміщення елементів керування, таких як кнопки, меню, панелі інструментів тощо, у головному вікні, а також налаштування їх візуальних властивостей.

Крім того, процес «Виведення головного вікна» також відповідає за візуалізацію симуляції розповсюдження інформації. Він відображає результати

симуляційних процесів у вигляді графічних об'єктів, діаграм, графіків або інших візуальних елементів. Це дозволяє користувачеві спостерігати за динамікою поширення інформації, аналізувати її характеристики та зробити висновки щодо ефективності розробленої системи.

Цей процес взаємодіє із процесами «Кінець», «Вибір параметрів генерації графу» та «Симуляція розповсюдження інформації».

Процес «Кінець» відповідає за коректне завершення роботи додатку.

Процес «Вибір параметрів генерації графу» відповідає за налаштування генератора соціального графу. Процес взаємодіє із прикладними процесами «Вибір числа вершин», «Вибір числа зв’язків» та «Стабілізація вершин графу». Також цей процес взаємодіє із процесом «Вибір параметрів симуляції».

Процес «Вибір числа вершин» відбовідає за налаштування відповідног параметра у генераторі соціального графу.

Процес «Вибір числа зв’язків» відповідає за налаштування числа зв’язків у нової вершини графу.

Процес «Стабілізація вершин графу» призначений для полегшення візуалізації графу. Він заморожує усі вершини та вимикає усі розрахунки пов’язані із їх положенням у просторі.

Процес «Вибір параметрів симуляції» відповідає за налаштування контролера симуляції. Процес взаємодіє із прикладними процесами «Керування параметрами відображення вершин», «Вибір ініціюючої множини» та «Вибір часу одного крогу симуляції». Також цей процес взаємодіє із процесом «Симуляція розповсюдження інформації».

Процес «Керування параметрами відображення вершин» відповідає за вибір режиму відображення вершин та змінює колір вершин в залежності від обраного режиму відображення.

Процес «Вибір ініціюючої множини» відповідає за вібір агентів розповсюдження інформації.

Процес «Вибір часу одного крогу симуляції» задає час одного кроку

симуляції коли вона запущена.

Процес «Симуляція розповсюждення інвормації» відповідає за покрокову реалізацію алгоритму розповсюдження інформаційних вірусів у соціальних мережах.

Таким чином утворюється цілісна система імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розроблені під час виконання роботи блок-схеми програмного забезпечення представлено на рисунках 4.1 та 4.2.

З блок-схеми основної програми видно, що спочатку відбувається виведення головного вікна програми.

Потім користувачу запропоновано вибрати параметри генерації соціального графу за моделлю Барабаші-Альберт.

Користувач вводить кількість учасників соціального графу. Цей параметр відповідає за кількість учасників моделі соціальної мережі, яка буде предметом імітаційного моделювання.

Користувач вводить кількість зв’язків нової вершини. Цей параметр потрібен для генерації соціального графу за моделлю Барабаші-Альберт.

Користувач обирає метод відображення графу. Граф може бути виведений на екраз силовим методом, або випадковим методом.

Як показано на блок-схемі – далі відбувається генерація моделі соціальної мережі за заданими параметрами.

Після того, як граф згенеровано – його слід вивести на екран. Цим займається підпрограма відображення графу.

Коли граф сгенеровано та відображено у робочій області додатку – можна мочинати симуляцію розповсюдження інформаційного вірусу.

Для цього потрібно обрати параметри симуляції.

Користувач вводить час одного кроку симуляції.

За коректне функціонування симуляції відповідає підпрограма Симуляція.

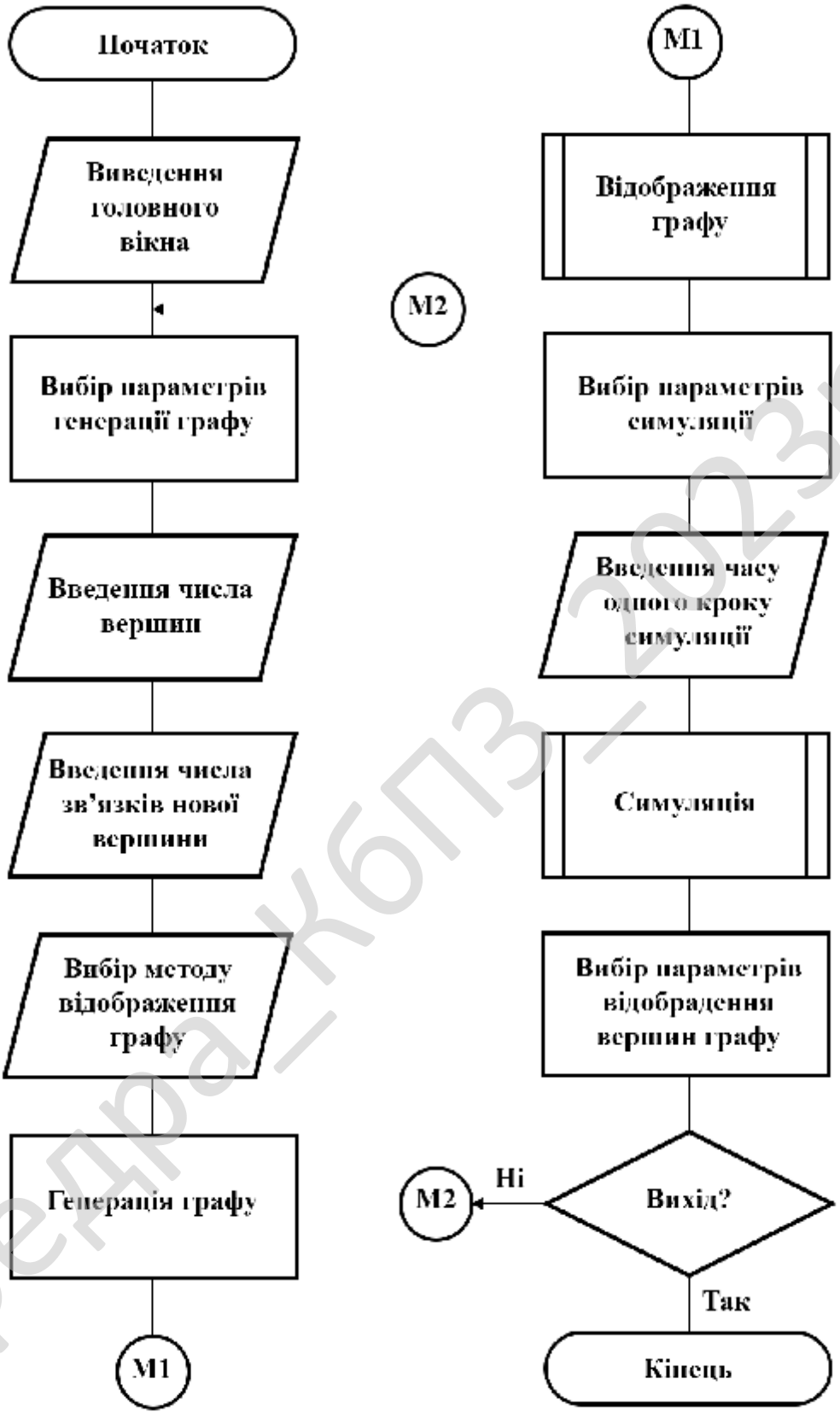

Рисунок 4.1 – Блок-схема основної програми

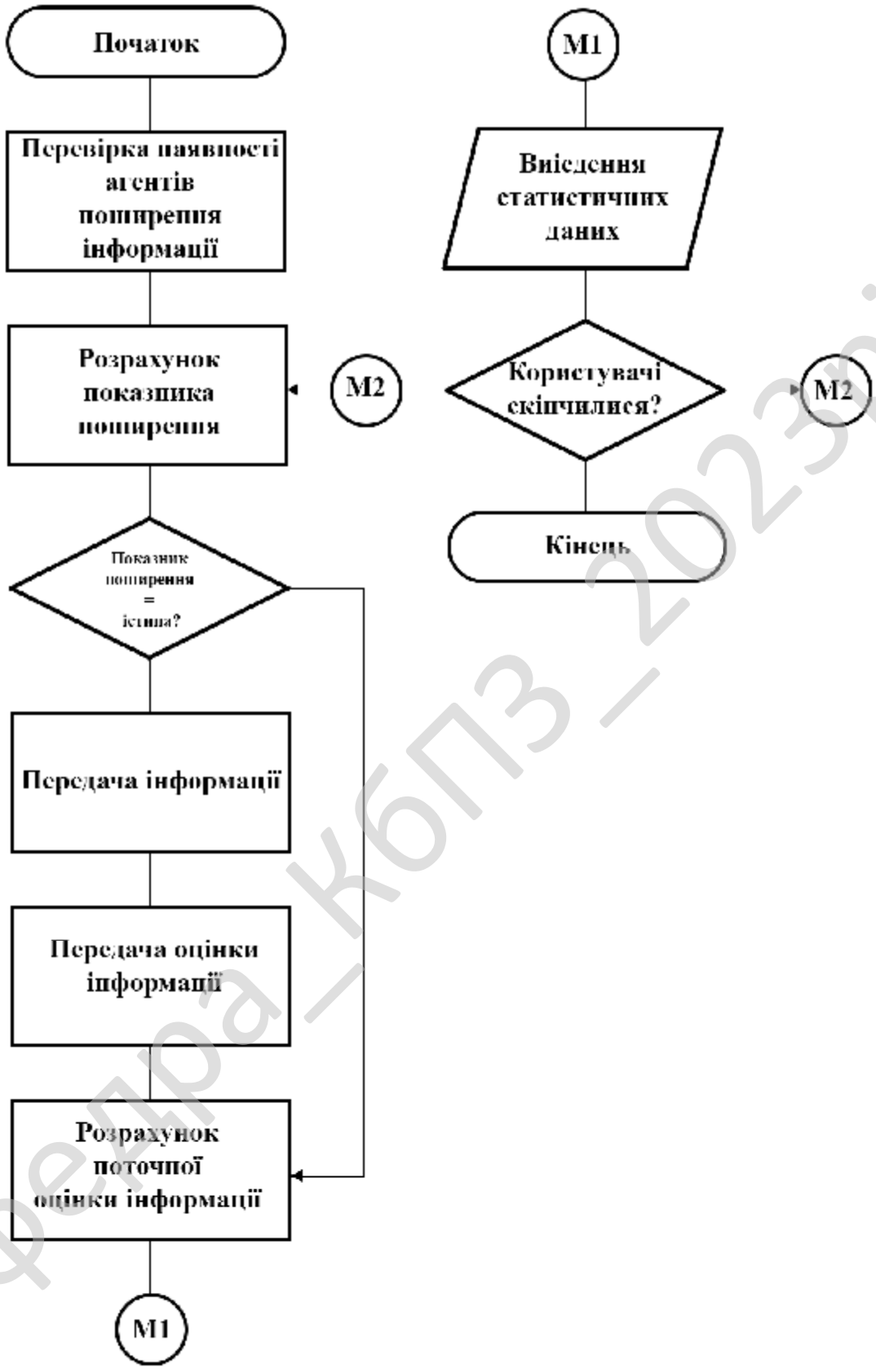

Рисунок 4.2 – Блок-схема підпрограми симуляції

Далі у користувача є вибір: або вийти із додатку, або почати процес генерації та симуляції з початку.

На рисунку 4.2 зображена блок-схема підпрограми симуляції.

З неї видно, що перш за все перевіряється наявність інформованих осіб. Якщо користувач не обрав агентів розповсюдження інформації – програма видасть помилку.

Далі для учасників соціального графу відбувається розрахунок показника поширення.

Якщо показник поширення дорівнює істині – учасник передає інформацію усім, з ким спілкується.

Також, із інформацією, учасник ділиться своєю думкою з приводу інформації.

Далі для кожного учасника відбувається розрахунок поточної оцінки інформації.

Як тільки усі розрахунки завершені – підпрограма симуляції виводить на екран статистичні дані.

Якщо більше нема користувачів для обробки – підпрограма завершує своє виконання.

Для стіорення додатку використовувався ігроіий двигун Unity3D 2019.4.40f1 та мова програмування C#.

Перш за все потрыбно розробити структури даних Network та Node.

Структура даних Network має наступний вигляд:

```
public class Network
    {

        #region Constructors

        public Network(int NodesCount)
        {
            _Nodes = new List<Node>();
            _Links = new int[NodesCount,NodesCount];

            for(int i = 0; i<NodesCount; i++){
                for(int j = 0; j<NodesCount; j++){
                    _Links[i,j] = 0;
                }
            }
        }
```

```
        #endregion

        #region Properties

        /// <summary>
        /// List of all nodes in the network.
        /// </summary>
        [SerializeField]
        [Tooltip("List of all nodes in the network.")]
        private List<Node> _Nodes;

        /// <summary>
        /// List of all nodes in the network.
        /// </summary>
        public List<Node> Nodes { get { return _Nodes; } }



        /// <summary>
        /// List of all links connecting the nodes.
        /// </summary>
        [SerializeField]
        [Tooltip("List of all links connecting the nodes.")]
        private int[,] _Links;

        /// <summary>
        /// List of all links connecting the nodes.
        /// </summary>
        public int[,] Links { get { return _Links; } }

        #endregion

        #region Methods

        public void CreateLink(int FirstNodeID, int SecondNodeID)
        {
            _Links[FirstNodeID, SecondNodeID] = 1;
        }

        public void ShowLinksMatrix()
        {
            Debug.Log(_Nodes.Count);
            for(int i = 0; i<_Nodes.Count; i++){
                string massage = "";
                for(int j = 0; j<_Nodes.Count; j++){
                    massage+=_Links[i,j]+" ";
                }
                Debug.Log(massage);
            }
        }

        public List<Node> FindNeighbors(int NodeID)
        {
            List<Node> Neighbors = new List<Node>();
            for(int i = 0; i< _Nodes.Count; i++){
                if(_Links[NodeID, i] == 1){
                    Neighbors.Add(_Nodes[i]);
                }
            }

            return Neighbors;
        }

        #endregion

    }
```

Структура даних Node має наступний вигляд:

```
public class Node
    {

        #region Constructors

        public Node(int NodeID)
        {
            System.Random rand = new System.Random(Guid.NewGuid().GetHashCode());
            _Id = NodeID;
            _Informed = false;
            _Opinion = NextFloat(-1, 1, rand);
            _Conservatism = NextFloat(0,1, rand);
            _Comunicability = NextFloat(0,1, rand);
            _Opinions_Got = new List<float>();
            _RepostMarker = false;
        }

        #endregion

        #region Properties

        /// <summary>
        /// Ідентивікатор.
        /// </summary>
        [SerializeField]
        private int _Id;

        public int Id { get { return _Id; } }

        /// <summary>
        /// Показник інформованості.
        /// </summary>
        [SerializeField]
        private bool _Informed;

        public bool Informed { get { return _Informed; } set { _Informed = value; }
}

        /// <summary>
        /// Оцінка інформації.
        /// </summary>
        [SerializeField]
        private float _Opinion;

        public float Opinion { get { return _Opinion; } }

        /// <summary>
        /// Коефіцієнт консерватизму.
        /// </summary>
        [SerializeField]
        private float _Conservatism;

        public float Conservatism { get { return _Conservatism; } }

        /// <summary>
        /// Коефіцієнт комунікабельності.
        /// </summary>
        [SerializeField]
        private float _Comunicability;
```

```
        public float Comunicability { get { return _Comunicability; } }

        /// <summary>
        /// Прийняті думки за останній крок.
        /// </summary>
        [SerializeField]
        private List<float> _Opinions_Got;

        public List<float> Opinions_Got { get { return _Opinions_Got; } }

        /// <summary>
        /// Індинатор репосту.
        /// </summary>
        [SerializeField]
        private bool _RepostMarker;

        public bool RepostMarker { get { return _RepostMarker; } }

        #endregion

        #region Methods

        public void CalculateRepostMarker(){
            if(((_Opinion < 0.64f) && (_Opinion > -0.64f) && (_Comunicability <
0.3f)) || (Informed == false)){
                _RepostMarker = false;
            } else {
                _RepostMarker = true;
            }
        }

        public void CalculateCurrentOpinion(){
            if(_Opinions_Got.Count != 0){
                float CSP = 0;
                float opinions_sum = 0;
                foreach(float op in _Opinions_Got){
                    opinions_sum += op;
                }
                CSP = opinions_sum/_Opinions_Got.Count;
                float current_op = _Opinion + (1 - _Conservatism) * CSP;
                if(current_op<-1){
                    current_op = -1;
                }
                if(current_op>1){
                    current_op = 1;
                }
                _Opinion = current_op;
                _Opinions_Got = new List<float>();
            }
        }

        private float NextFloat(float min, float max, System.Random random){
            double val = (random.NextDouble() * (max - min) + min);
            return (float)val;
        }

        #endregion

    }
```

Далі слід сгенерувати соціальний граф за моделлю Барабаші-Альберт. За це відповідає функція GenerateBarabasiAlbertNetwork. Вона виглядає так:

```
/// <summary>
    /// Генерація графу за алгоритмом Барабаші-Альберт.
    /// </summary>
    public void GenerateBarabasiAlbertNetwork()
    {
        Simulation.StopSimulation();

        // Start a new network
        ForceDirectedGraph.DataStructure.Network network = new
ForceDirectedGraph.DataStructure.Network(NumberOfMembers);

        // Add a triangle network
        GenerateInitialNetwork(network, NewMemberConnectionsCount);
        Debug.Log("Initial Network Generated");


        for(int nodeToBeAdded = network.Nodes.Count; nodeToBeAdded <
NumberOfMembers; nodeToBeAdded++){
            network.Nodes.Add(new
ForceDirectedGraph.DataStructure.Node(nodeToBeAdded));
            Debug.Log("Node " + nodeToBeAdded + " added");

            List<int> prob = new List<int>();
            for(int j = 0; j < nodeToBeAdded; j++){
                int degree = network.FindNeighbors(j).Count;
                for(int k = 0; k < degree; k++)
                    prob.Add(j);
            }

            var random = new System.Random(Guid.NewGuid().GetHashCode());
            List<int> nextNodeToConnectIDs = new List<int>();
            for(int j = 0; j < NewMemberConnectionsCount; j++){
                for(;;){
                    int id = random.Next(prob.Count);
                    if(!nextNodeToConnectIDs.Contains(prob[id])){
                        Debug.Log("Node " + id);
                        nextNodeToConnectIDs.Add(prob[id]);
                        break;
                    }
                }

            }

            for(int j = 0; j < nextNodeToConnectIDs.Count; j++){
                network.CreateLink(nodeToBeAdded, nextNodeToConnectIDs[j]);
                network.CreateLink(nextNodeToConnectIDs[j], nodeToBeAdded);
            }

            Debug.Log("Node " + nodeToBeAdded + " added");
        }
        //network.ShowLinksMatrix();

        Graph.Initialize(network, GraphMode);

    }
```

Ця функція використовує допоміжну функцію GenerateInitialNetwork, яка генерує ініціюючу множину для моделі Барабаші-Альберт:

```
/// <summary>
    /// Генерація початкової мережі.
    /// </summary>
```

```
    private void GenerateInitialNetwork(ForceDirectedGraph.DataStructure.Network
network, int initialNumber)
    {
        var random = new System.Random(Guid.NewGuid().GetHashCode());
        for(int i = 0; i < initialNumber; i++){
            network.Nodes.Add(new ForceDirectedGraph.DataStructure.Node(i));
        }

        foreach(var node in network.Nodes){
            int neighborsCount = random.Next(1,initialNumber);

            for(int i = 0; i < neighborsCount; i++){
                int nextNodeID;
                for(;;){
                    nextNodeID = random.Next(network.Nodes.Count);
                    if(nextNodeID != node.Id){
                        break;
                    }
                }

                network.CreateLink(node.Id, nextNodeID);
                network.CreateLink(nextNodeID, node.Id);
            }
        }

    }
```

Коли граф соціальної мережі згенеровано – слід вивести його на екран. За це цідповідає функція DisplayForceDirected компоненту GraphManager.cs. Вона має такий вигляд:

```
private void DisplayForceDirected()
        {
            // Clear everything
            Clear();

            // Display nodes
            DisplayNodes();

            // Display links
            DisplayLinks();

            // Shuffle the nodes
            ShuffleNodes();
        }
```

Граф фифедено на екран. Тепер, якщо користувач вказав силовий метод відображення, програма повинна пропаховувати та застосовувани сили притягування та відштовхування для кожної віршини. За це відповідають наступні функції:

```
/// <summary>
        /// Computes and applies forces to nodes.
        /// </summary>
        private void ApplyForces()
        {
            if (GraphNodes == null)
                return;
```

```
            // Stores all the forces to be applied to each node
            Dictionary<GraphNode, List<Vector2>> nodeForces = new
Dictionary<GraphNode, List<Vector2>>();
            foreach (var node1 in GraphNodes)
                nodeForces.Add(node1, new List<Vector2>());

            // Compute repulsion forces
            foreach (var node1 in GraphNodes)
                foreach (var node2 in GraphNodes)
                    if (node1 != node2)
                        nodeForces[node1].Add(ComputeRepulsiveForce(node1, node2));

            // Compute attraction forces
            foreach (var link in GraphLinks)
            {
                var force = ComputeAttractionForce(link);
                nodeForces[link.FirstNode].Add(-force);
                nodeForces[link.SecondNode].Add(force);
            }

            // Apply forces
            foreach (var node in nodeForces.Keys)
                node.ApplyForces(nodeForces[node]);
        }

        /// <summary>
        /// Computes the distance between two nodes.
        /// </summary>
        private float ComputeDistance(GraphNode node1, GraphNode node2)
        {
            return (float)
                Math.Sqrt
                (
                    Math.Pow(node1.transform.position.x -
node2.transform.position.x, 2)
                    +
                    Math.Pow(node1.transform.position.y -
node2.transform.position.y, 2)
                );
        }

        /// <summary>
        /// Computes the repulsive force against a node.
        /// </summary>
        private Vector2 ComputeRepulsiveForce(GraphNode node, GraphNode
repulsiveNode)
        {
            // Compute distance
            float distance = ComputeDistance(node, repulsiveNode);
            if (distance > REPULSION_DISTANCE)
                return Vector3.zero;

            // Compute force direction
            Vector2 forceDirection = (node.transform.position -
repulsiveNode.transform.position).normalized;

            // Compute distance force
            float distanceForce = (REPULSION_DISTANCE - distance) /
REPULSION_DISTANCE;

            // Compute repulsive force
```

```
            return forceDirection * distanceForce * REPULSION_FORCE *
Time.deltaTime;
        }

        /// <summary>
        /// Computes the attraction force between two nodes.
        /// </summary>
        private Vector2 ComputeAttractionForce(GraphLink link)
        {
            // Compute force direction
            Vector2 forceDirection = (link.FirstNode.transform.position -
link.SecondNode.transform.position).normalized;

            // Compute repulsive force
            return forceDirection * link.Width * ATTRACTION_FORCE * Time.deltaTime;
        }
```

Потрібно дати користувачу можливість взаємодіяти із користувачами, аби він міг розташувати їх так, як йому буде зручно. За це відповідає компонент Dragable.cs.

За розрахунок кольору вершини в залежності від обраног режиму відображення вершин відповідає функція CalculateColor компоненту GraphNode.cs:

```
private void CalculateColor(int mode){
            switch (mode)
            {
                case 0:
                    if(Node.Informed)
                        NodeColor = Color.yellow;
                    if(!Node.Informed)
                        NodeColor = Color.white;
                    break;

                case 1:
                    if(Node.RepostMarker)
                        NodeColor = Color.red;
                    if(!Node.RepostMarker)
                        NodeColor = Color.white;
                    break;
                case 2:
                    NodeColor = conserv.Evaluate(Node.Conservatism);
                    break;
                case 3:
                    NodeColor = comunicab.Evaluate(Node.Comunicability);
                    break;
                case 4:
                    float val = (Node.Opinion+1)/2;
                    NodeColor = opinion.Evaluate(val);
                    break;
            }
        }
```

Після генерації та відображення графу можна починати симуляцію. Коли користувач почав симуляцію – програма виконує функцію SimulationStep коли таймер одного кроку симуляції добігає кінця:

```
public void SimulationStep(){
            int informedCount = 0;
            if(_SimulationStep == 0){
                foreach(DataStructure.Node node in _Network.Nodes){
                    if(node.Informed == true)
                        informedCount++;
                }
                if(informedCount < 1){
                    WarningMassage.SetActive(true);
                    Massage.text = "Оберіть агентів розповсюдження інформації!!!";
                    return;
                }
                else
                    _SimulationStep++;
            }

            SimStepText.text = _SimulationStep.ToString();

            informedCount = 0;
            int repostingCount = 0;
            float avgOpinion = 0.0f;

            foreach(DataStructure.Node node in _Network.Nodes){
                node.CalculateRepostMarker();
                if(node.RepostMarker == true)
                    repostingCount++;
            }
            foreach(DataStructure.Node node in _Network.Nodes){
                if(node.RepostMarker == true){
                    List<DataStructure.Node> neighbors =
_Network.FindNeighbors(node.Id);
                    foreach(DataStructure.Node neighbor in neighbors){
                        neighbor.Informed = true;
                        neighbor.Opinions_Got.Add(node.Opinion);
                    }
                }
            }
            foreach(DataStructure.Node node in _Network.Nodes){
                node.CalculateCurrentOpinion();
                avgOpinion += node.Opinion;
                if(node.Informed == true)
                    informedCount++;
            }

            avgOpinion = avgOpinion/_Network.Nodes.Count;

            InformedCount.text = informedCount.ToString();
            RepostingCount.text = repostingCount.ToString();
            AvgOpinion.text = avgOpinion.ToString("0.00");

            _SimulationStep++;
        }
```

Спочатку ця функція перевіряє, чи обрав користувач агентів розповсюдження інформації. Потім розраховується показник поширення

інформації. Наступним кроком програма розповсюджує інформацію. Далі відбувається розрахунок поточної оцінки інформації користувачами.

Також програма розраховує та оновлює статистичні дані.

4.2 Захист розробленого програмного забезпечення

Ліцензія GNU General Public License (GPL) дозволяє додатку бути розповсюдженим, змінюваним та поширюваним у рамках вільного використання програмного забезпечення. Вона надає можливість вільно користуватися вихідним кодом та вносити зміни до нього, забезпечуючи збереження авторських прав та походження додатку.

Розповсюдження додатку за ліцензією GNU General Public License (GPL) є важним аспектом вільного програмного забезпечення і відкритого коду. Ліцензія GPL встановлює визначені правила та умови, які регулюють використання, зміну та поширення програмного забезпечення.

Основними принципами ліцензії GPL є:

1. Свобода використання: Ліцензія GPL гарантує користувачам право вільно використовувати додаток для будь-яких цілей без обмежень.

2. Доступ до вихідного коду: Одним з важливих аспектів ліцензії GPL є вимога надання вихідного коду програми разом з дистрибутивом. Це дозволяє користувачам вивчати, модифікувати та адаптувати програмне забезпечення під їхні потреби.

3. Поширення з умовами ліцензії: Якщо ви розповсюджуєте програмне забезпечення, яке ліцензовано за GPL, ви повинні забезпечити, що всі отримувачі отримують ту саму ліцензію та права, які встановлені в GPL. Це означає, що всі модифікації та похідні роботи також повинні бути доступними на умовах GPL.

4. Взаємодія з іншими вільними програмами: Ліцензія GPL дозволяє використовувати програмне забезпечення, ліцензоване за GPL, разом з іншими вільними програмами. Це сприяє розвитку екосистеми вільного програмного

забезпечення та сприяє взаємодії між різними проектами.

Розповсюдження додатку за ліцензією GPL важливе для забезпечення вільності, доступності та збереження відкритого коду. Вона забезпечує користувачам права та можливості досліджувати, вдосконалювати та адаптувати додаток під свої потреби. Крім того, ліцензія GPL сприяє спільноті розробників, які можуть вносити свої внески до проекту, допомагати у розвитку програми та спільно вирішувати проблеми.

За розповсюдження додатку за ліцензією GPL також можуть існувати деякі вимоги та обмеження. Наприклад, якщо ви змінюєте програмне забезпечення, ліцензоване за GPL, то ви зобов'язані розповсюджувати зміни та покращення разом з вихідним кодом. Також можуть бути вимоги щодо надання повідомлення про зміни, атрибуції авторства та збереження ліцензії при поширенні.

Розповсюдження додатку за ліцензією GPL має значний вплив на розвиток вільного програмного забезпечення та сприяє принципам відкритості та спільної роботи. Вона забезпечує права користувачів, сприяє створенню великої спільноти розробників та сприяє інноваціям у сфері програмного забезпечення.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Впровадження системи імітаційного моделювання поширення інформації в промислову експлуатацію може бути розділена на кілька етапів:

Збір та аналіз вхідних даних: необхідно зібрати всю необхідну інформацію, яка буде використовуватися в системі моделювання. Це може включати дані про користувачів, їхню поведінку та взаємодії, а також дані про саму інформацію, яка поширюється.

Розробка математичної моделі: на основі зібраних даних розробляється математична модель, яка відображає процес поширення інформації. Ця модель може включати різні параметри, такі як швидкість поширення інформації та її вплив на користувачів.

Розробка програмної системи: на основі математичної моделі створюється програмна система для моделювання процесу поширення інформації. Ця система може включати різні інструменти для відображення даних та аналізу результатів моделювання.

Тестування та оптимізація: після створення програмної системи проводиться тестування моделі, щоб переконатися, що вона працює правильно. Після цього можуть бути здійснені додаткові оптимізації, щоб покращити результати моделювання та забезпечити більш точне відображення процесу поширення інформації.

Впровадження в експлуатацію: після успішного тестування та оптимізації систему моделювання можна впровадити в промислову експлуатацію. Вона може використовуватися для аналізу та прогнозування процесу поширення інформації.

Програмно-апаратні вимоги:

- Загальний обсяг ОЗП: 8 Гб.
- Процесор: AMD Ryzen 5 2600.

– Вільний простір на жорсткому диску: 1 Гб.

– Операційна система Microsoft Windows 10/11.

Головне вікно програми зображене на рисунку 5.1.

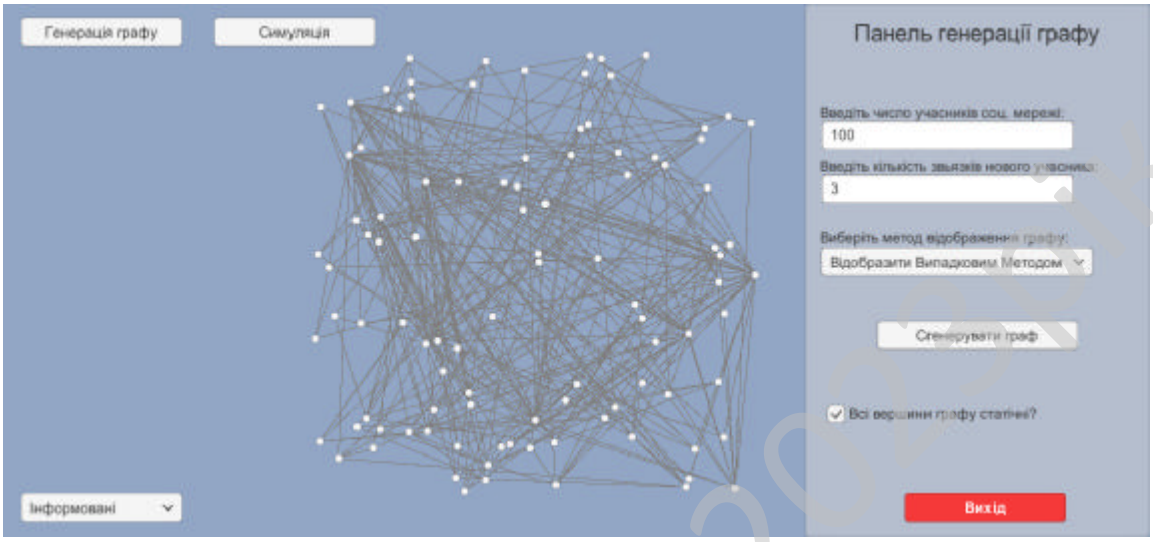

Рисунок 5.1 – Головне вікно програми

Процес симуляції зображено на рисунку 5.2.

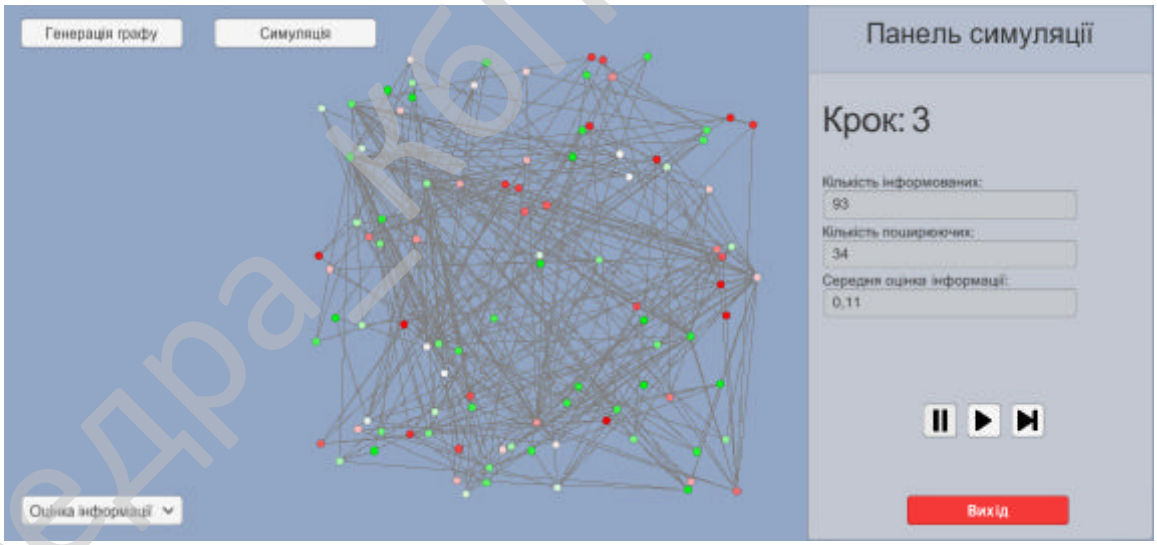

Рисунок 5.2 – Процес симуляції

6 ОСНОВНІ ВИСНОВКИ

У ході виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, було розроблене програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

Нажаль представлено мало вітчизняних розробок у цій сфері.

Під час виконання роботи були вирішені наступні задачі:

- Був проведений огляд існуючих систем імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.
- Було сформаовне ТЗ.
- Обгрунтовано обрані засоби реалізації завдання згідно із ТЗ.
- Розроблено алгоритми функціонування програмного забезпечення.
- На основі отриманих результатів досліджень створена програмна реалізація системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

У цілому, розроблене програмне забезпечення підтверджує коректність прийнятих проектних рішень і повністю відповідає вимогам, поставленим у технічному завданні. Результат роботи включає потенціал для подальшого вдосконалення та може бути успішно застосованим у різних сферах діяльності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. M. Cha, H. Haddadi, F. Benevenuto, and K. P. Gummadi. Measuring User Influence in Twitter: The Million Follower Fallacy. In ICWSM ’10 , 2010.

2. M. Goetz, J. Leskovec, M. Mcglohon, and C. Faloutsos. Modeling blog dynamics. In ICWSM, 2009.

3. J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In KDD ’09, 2009.

4. D. Liben-Nowell and J. Kleinberg. Tracing information flow on a global scale using Internet chain-letter data. PNAS, 105(12):4633–4638, 2008.

5. Daley DJ, Kendall DG, Stochastic rumors, J. Inst. Math. Appl. 142(1965), pp. 42-55.

6. Kermack, W. O.; McKendrick, A. G. (1927). "A Contribution to the Mathematical Theory of Epidemics". Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences. 115 (772): 700. Bibcode:1927RSPSA.115..700K. doi:10.1098/rspa.1927.0118. JSTOR 94815.

7. Granovetter M. Threshold Models of Collective Behavior // American Journal of Sociology. 1978. V. 83, № 6. P. 1420-1443.

8. Kempe D., Kleinberg J., Tardos E. Maximizing the Spread of Influence through a Social Network / Proceedings of the 9-th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2003. P. 137 – 146.

9. Myerson R.B. Game Theory: Analysis of Conflict. — London: Harvard Univ. Press, 1991.

10. Goldenberg J., Libai B., Muller E, Talk of the Network: A Complex Systems Look at the Underlying Process of Word-of-Mouth // Marketing Letters. 2001 № 2. P. 11-34.

11. De Groot M.H. Reaching a Consensus // Journal of American Statistical Assotiation. 1974. № 69. P. 118-121.

12. Friedkin N.E. Structural Cohesion and Equivalence Explanations of Social Homogeneity // Sociological Methods and Research. 1984. № 12. P. 235-261.

13. H. W. Hethcote. The mathematics of infectious diseases. SIAM Review, 42(4):599–653, 2000.

14. John Von Neumann, John; Burks, Arthur W. (1966), Theory of Self-Reproducing Automata. University of Illinois Press, Urbana and London 1966.

15. R. Isea and R. Mayo-García. Mathematical analysis of the spreading of a rumor among different subgroups of spreaders. Pure and Applied Mathematics Letters (2015), Vol. 2015, pp 50-54.

16. Chi-Jen Lin. Assignment Problem for Team Performance Promotion under Fuzzy Environment // Hindawi Publishing Corporation. Mathematical Problems in Engineering. 2013. 10 p. URL: http://dx.doi.org/10.1155/2013/791415 (дата обращения: 25.03.17).

17. Kaufmann A., Gupta M. Introduction to Fuzzy Arithmetic: Theory and Applications // Van Nostrand Reinhold. 1991. – 161 p.

18. Surapati Pramanik, Pranab Biswas. Multi-objective Assignment Problem with Generalized Trapezoidal Fuzzy Numbers // International Journal of Applied Information Systems (IJAIS). Foundation of Computer Science FCS, New York, USA. – 2012. – Vol. 2, № 6. URL: www.ijais.org (дата обращения: 25.03.17).

19. Grandjean, Martin (2014). La connaissance est un réseau. Les Cahiers du Numérique 10 (3): 37–54. doi:10.3166/lcn.10.3.37-54

20. Zadorozhnyi V., Yudin E. Growing Network: Nonlinear Extension of the Barabasi-Albert Model. Communications in Computer and Information Science. P. 432-439 (2014)

21. Bakhshaliyev, R., Shikhametova, D., & Ahmad, F. (2019). Simulation of the spread of computer viruses on social networks. Proceedings of the 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2277-2280.

22. Bi, Y., Chen, C., Zhang, C., & Yang, Y. (2018). A novel epidemic

model of information dissemination in social networks. IEEE Access, 6, 11329-11338.

23. Duan, L., Xue, X., & Li, Y. (2018). Research on information diffusion model of social media based on virus spreading mechanism. Journal of Physics: Conference Series, 1087(5), 052019.

24. Li, Y., Zhang, M., & Zhang, S. (2017). A survey on modeling and simulation of computer virus spreading. IEEE Access, 5, 2854-2864.

25. Liu, J., & Zhang, L. (2020). Research on the propagation of malicious information on social networks based on a virus spreading model. Journal of Physics: Conference Series, 1619(1), 012102.

26. Wang, J., Wu, F., & Li, J. (2018). Modeling and simulation of computer virus spreading in social networks. IEEE Access, 6, 34706-34717.

27. Wu, Q., Li, S., Li, Y., & Wang, H. (2020). Epidemic model of information propagation on social media. Journal of Physics: Conference Series, 1620(1), 012076.

28. Zhang, M., Li, Y., & Li, J. (2017). Modeling and simulation of computer virus spreading under the influence of immunization and quarantine. IEEE Access, 5, 19746-19757.

29. Bastian, Mathieu; Heymann, Sebastien; Jacomy, Mathieu (2009). Gephi : An Open Source Software for Exploring and Manipulating Networks. AAAI Publications, Third International AAAI Conference on Weblogs and Social Media.

30. Desmedt, Patrice (2011). Sébastien Heymann - Le cartographe des données. L'Usine Nouvelle.

31. The Gephi Consortium - Members. The Gephi Consortium.

32. Grandjean, Martin (2015). GEPHI - Introduction to Network Analysis and Visualisation.

33. Correa, Debora C. (2011). Using Digraphs and a Second-Order Markovian Model for Rhythm Classification.

34. Leetaru, Kalev H. (2011). Culturomics 2.0:Forecasting Large-scale human behavior using global news media tone in time and space. First Monday.

35. Change History, Social Media Research Foundation

36. Smith, Marc A.; Shneiderman, Ben; Milic-Frayling, Natasa; Rorigues, Eduarda; Barash, Vladimir; Dunne, Cody; Capone, Tony; Perer, Adam; Gleave, Eric (2009), "Analyzing (social media) networks with NodeXL", Proceedings of the Fourth International Conference on Communities and Technologies, ACM: 255–264

37. Hansen, Derek L.; Shneiderman, Ben; Smith, Marc (2010), Analyzing social media networks with NodeXL: Insights from a Connected World, Morgan Kaufmann

38. Yao, Q. et al (2021) Construction Safety Knowledge Sharing on Twitter: A Social Network Analysis, Safety Science, 143, 105411, https://www.researchgate.net/publication/353546913_Safety_knowledge_sharing_on_Twitter_A_social_network_analysis

39. For Programmers: About NodeXL Graph Data Providers, Social Media Research Foundation, retrieved May 13, 2013

40. Bonsignore, E.M.; Dunne, Cody; Rotman, D.; Smith, M.; Capone, T.; Hansen, D.L.; Shneiderman, B. (2009), "First Steps to Netviz Nirvana: Evaluating Social Network Analysis with NodeXL" (PDF), International Conference on Computational Science and Engineering: 332–339

41. Allnutt, Luke (April 11, 2012), "Pictures at a Revolution", Foreign Policy, retrieved May 13, 2013

42. Himelboim, Itai; McCreery, Stephen; Smith, Marc (2013-01-01). "Birds of a Feather Tweet Together: Integrating Network and Content Analyses to Examine Cross-Ideology Exposure on Twitter". Journal of Computer-Mediated Communication. 18 (2): 40–60.

43. Riccitiello, John (23 жовтня 2014). Інтерв'ю з Dean Takahashi. «John Riccitiello sets out to identify the engine of growth for Unity Technologies (interview)». VentureBeat.

44. Unity - Manual: Working In Unity. docs.unity3d.com.

45. Unity - Manual: The Main Windows. docs.unity3d.com.

46. Unity - Manual: Physics. docs.unity3d.com.

47. Unity - Manual: Cameras. docs.unity3d.com.

48. Kelliher, Fiona (24 серпня 2018). Video game company grabs two buildings on Mission Street for big expansion. San Francisco Business Times.

49. Robertson, Adi (3 березня 2015). Unity officially releases its new game engine: Unity 5. The Verge.

50. Dillet, Romain (5 вересня 2018). Unity CEO says half of all games are built on Unity. TechCrunch.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРБ-125.23.0018.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Середа О.О.			Програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах	Літ. Б	Аркуш 1 / Аркушів 6
Перевірив		Мелешко Є.В.					
Н. Контр.		Гермак В.С.				ЦНТУ КБ-19	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 12-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи моделювання з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– систему імітаційного моделювання поширення інформаційних вірусів у соціальних мережах;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на платформах з ОС Windows 10/11, а також у браузерах, які підтримують WebGL 1.0/2.0 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Рекомендовані системні вимоги:

Процесор AMD Ryzen 5 2600;

ОЗП 8 Гб;

Жорсткий диск 1 Гб.

5.8.2 Мова програмування

Середовище Unity3D 2019.4.40f1 із використанням мови програмування C#.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 68 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 6.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Мелешко Є.В.

Програмне забезпечення системи імітаційного моделювання поширення інформаційних вірусів у соціальних мережах

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 29

Літера: РП

Кропивницький – 2023 року

MainSceneManager.cs - головний файл проекту

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

public class MainSceneManager : MonoBehaviour
{

    #region Fields/Properties

    /// <summary>
    /// Посилання на об'єкт який відповідає за відображення графу.
    /// </summary>
    [SerializeField]
    [Tooltip("The graph displaying the network.")]
    private GraphManager Graph;

    /// <summary>
    /// Посилання на контролер симуляції.
    /// </summary>
    [SerializeField]
    private SimulationManager Simulation;

    /// <summary>
    /// Кількість учасників мережі.
    /// </summary>
    [SerializeField]
    private int NumberOfMembers;
    public int _NumberOfMembers { set { NumberOfMembers = value; }}

    /// <summary>
    /// Параметр для генерації графу за алгоритмом Барабаші-Альберт.
    /// </summary>
    [SerializeField]
    private int NewMemberConnectionsCount;
    public int _NewMemberConnectionsCount { set { NewMemberConnectionsCount =
value; }}

    /// <summary>
    /// Час одного кроку симуляції.
    /// </summary>
    [SerializeField]
    private float OneStepTime;
    public float _OneStepTime { set { OneStepTime = value; }}

    /// <summary>
    /// Режим відображення графу.
    /// </summary>
    [SerializeField]
    private int GraphMode;
    public int _GraphMode { set { GraphMode = value; }}

    /// <summary>
    /// Режим відображення вершин графу.
    /// </summary>
    [SerializeField]
    private Dropdown NodeDisplayMode;

    #endregion

    #region Methods
```

```
    /// <summary>
    /// Генерація графу за алгоритмом Барабаші-Альберт.
    /// </summary>
    public void GenerateBarabasiAlbertNetwork()
    {
        Simulation.StopSimulation();

        // Start a new network
        ForceDirectedGraph.DataStructure.Network network = new
ForceDirectedGraph.DataStructure.Network(NumberOfMembers);

        // Add a triangle network
        GenerateInitialNetwork(network, NewMemberConnectionsCount);
        Debug.Log("Initial Network Generated");


        for(int nodeToBeAdded = network.Nodes.Count; nodeToBeAdded <
NumberOfMembers; nodeToBeAdded++){
            network.Nodes.Add(new
ForceDirectedGraph.DataStructure.Node(nodeToBeAdded));
            Debug.Log("Node " + nodeToBeAdded + " added");

            List<int> prob = new List<int>();
            for(int j = 0; j < nodeToBeAdded; j++){
                int degree = network.FindNeighbors(j).Count;
                for(int k = 0; k < degree; k++)
                    prob.Add(j);
            }

            var random = new System.Random(Guid.NewGuid().GetHashCode());
            List<int> nextNodeToConnectIDs = new List<int>();
            for(int j = 0; j < NewMemberConnectionsCount; j++){
                for(;;){
                    int id = random.Next(prob.Count);
                    if(!nextNodeToConnectIDs.Contains(prob[id])){
                        Debug.Log("Node " + id);
                        nextNodeToConnectIDs.Add(prob[id]);
                        break;
                    }
                }

            }

            for(int j = 0; j < nextNodeToConnectIDs.Count; j++){
                network.CreateLink(nodeToBeAdded, nextNodeToConnectIDs[j]);
                network.CreateLink(nextNodeToConnectIDs[j], nodeToBeAdded);
            }

            Debug.Log("Node " + nodeToBeAdded + " added");
        }
        //network.ShowLinksMatrix();

        Graph.Initialize(network, GraphMode);

    }

    /// <summary>
    /// Генерація початкової мережі.
    /// </summary>
    private void GenerateInitialNetwork(ForceDirectedGraph.DataStructure.Network
network, int initialNumber)
    {
        var random = new System.Random(Guid.NewGuid().GetHashCode());
        for(int i = 0; i < initialNumber; i++){
            network.Nodes.Add(new ForceDirectedGraph.DataStructure.Node(i));
        }

        foreach(var node in network.Nodes){
            int neighborsCount = random.Next(1,initialNumber);
```

```
            for(int i = 0; i < neighborsCount; i++){
                int nextNodeID;
                for(;;){
                    nextNodeID = random.Next(network.Nodes.Count);
                    if(nextNodeID != node.Id){
                        break;
                    }
                }

                network.CreateLink(node.Id, nextNodeID);
                network.CreateLink(nextNodeID, node.Id);
            }
        }

    }

    /// <summary>
    /// Початок симуляції.
    /// </summary>
    public void StartSimulation(){
        Simulation.Initialize(Graph.Network, OneStepTime);
    }

    /// <summary>
    /// Зміна режиму відображення вершин.
    /// </summary>
    public void ChangeNodeDisplayMode(){
        foreach(var node in Graph.Nodes){
            node.Mode = NodeDisplayMode.value;
        }
    }

    /// <summary>
    /// Закриття додатку.
    /// </summary>
    public void CloseApp(){
        Application.Quit();
    }

    #endregion

}
```

GraphManager.cs – контролер відображення графу

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

namespace ForceDirectedGraph
{
    public class GraphManager : MonoBehaviour
    {

        #region Constants


        /// <summary>
        /// Сила відштовхування.
        /// </summary>
        private const float REPULSION_FORCE = 40000f;

        /// <summary>
        /// Дистанція відштовхування.
        /// </summary>
        private float REPULSION_DISTANCE;

        private float RANDOM_AREA_SIZE;

        /// <summary>
        /// Сила притягування.
        /// </summary>
        private const float ATTRACTION_FORCE = 40000f;

        #endregion

        #region Initialization

        /// <summary>
        /// Ініціалізація графу.
        /// </summary>
        public void Initialize(DataStructure.Network network, int displayMode)
        {
            _Network = network;
            _DisplayMode = displayMode;

            if(_DisplayMode == 0){
                REPULSION_DISTANCE = network.Nodes.Count/4;
                DisplayForceDirected();
            }
            if(_DisplayMode == 1){
                RANDOM_AREA_SIZE = network.Nodes.Count/10;
                DisplayRandom();
            }
        }

        #endregion

        #region Fields/Properties

        private bool IsStatic;

        private int _DisplayMode;

        [Header("Nodes")]

        /// <summary>
        /// Об'єкт у якому зберігаються усі вершини.
        /// </summary>
        [SerializeField]
```

```
[Tooltip("References the parent holding all nodes.")]
private GameObject NodesParent;

/// <summary>
/// Об'єкт вершина.
/// </summary>
[SerializeField]
[Tooltip("Template used for initiating nodes.")]
private GameObject NoteTemplate;

/// <summary>
/// Список всіх вершин.
/// </summary>
private List<GraphNode> GraphNodes;
public List<GraphNode> Nodes { get { return GraphNodes; } }



[Header("Links")]

/// <summary>
/// Об'єкт у якому зберігаються усі ребра.
/// </summary>
[SerializeField]
[Tooltip("References the parent holding all links.")]
private GameObject LinksParent;

/// <summary>
/// Об'єкт ребро.
/// </summary>
[SerializeField]
[Tooltip("Template used for initiating links.")]
private GameObject LinkTemplate;

/// <summary>
/// Список всіх ребер.
/// </summary>
private List<GraphLink> GraphLinks;



[Header("Data")]

/// <summary>
/// The netwok being displayed.
/// </summary>
[SerializeField]
[Tooltip("The netwok being displayed.")]
private DataStructure.Network _Network;

/// <summary>
/// The netwok being displayed.
/// </summary>
public DataStructure.Network Network { get { return _Network; } }

#endregion

#region Display Methods

public void SetStatic(bool mode){
    if(mode == true){
        IsStatic = true;
        foreach(GraphNode node in GraphNodes){
            node.IsStatic = true;
            node.Rigidboby.bodyType = RigidbodyType2D.Kinematic;
        }
        foreach(GraphLink link in GraphLinks){
            link.IsStatic = mode;
        }
```

```
        }
        if(mode == false){
            IsStatic = false;
            foreach(var node in GraphNodes){
                node.IsStatic = false;
                node.Rigidboby.bodyType = RigidbodyType2D.Dynamic;
            }
            foreach(GraphLink link in GraphLinks){
                link.IsStatic = false;
            }
        }
    }

    /// <summary>
    /// Displays the network.
    /// </summary>
    private void DisplayForceDirected()
    {
        // Clear everything
        Clear();

        // Display nodes
        DisplayNodes();

        // Display links
        DisplayLinks();

        // Shuffle the nodes
        ShuffleNodes();
    }

    /// <summary>
    /// Displays the network.
    /// </summary>
    private void DisplayRandom()
    {
        // Clear everything
        Clear();

        // Display nodes
        DisplayNodesRandomly();

        // Display links
        DisplayLinks();

        // Shuffle the nodes
        ShuffleNodes();

    }

    /// <summary>
    /// Deletes all nodes and links in the graph.
    /// </summary>
    private void Clear()
    {
        IsStatic = false;

        // Clear nodes
        GraphNodes = new List<GraphNode>();
        foreach (Transform entity in NodesParent.transform)
            GameObject.Destroy(entity.gameObject);

        // Clear paths
        GraphLinks = new List<GraphLink>();
        foreach (Transform path in LinksParent.transform)
            GameObject.Destroy(path.gameObject);
    }

    /// <summary>
```

```
        /// Displays nodes on the graph.
        /// </summary>
        private void DisplayNodes()
        {
            // For each position, create an entity
            foreach (var node in Network.Nodes)
            {
                // Create a new entity instance
                GameObject graphNode = Instantiate(NoteTemplate,
NodesParent.transform);
                graphNode.transform.position = Vector3.zero;
                graphNode.transform.localRotation =
Quaternion.Euler(Vector3.zero);

                // Extract the script
                GraphNode script = graphNode.GetComponent<GraphNode>();

                // Initialize data
                script.Initialize(node,
(Network.FindNeighbors(node.Id).Count)/2);

                // Add to list
                GraphNodes.Add(script);
            }
        }

        /// <summary>
        /// Displays nodes on the graph.
        /// </summary>
        private void DisplayNodesRandomly()
        {
            var random = new System.Random();
            // For each position, create an entity
            foreach (var node in Network.Nodes)
            {
                // Create a new entity instance
                GameObject graphNode = Instantiate(NoteTemplate,
NodesParent.transform);

                float x = (float)(random.NextDouble() * RANDOM_AREA_SIZE);
                float y = (float)(random.NextDouble() * RANDOM_AREA_SIZE);
                Vector3 position = new Vector3(x,y, 0);
                graphNode.transform.position = position;
                graphNode.transform.localRotation =
Quaternion.Euler(Vector3.zero);

                // Extract the script
                GraphNode script = graphNode.GetComponent<GraphNode>();

                if(RANDOM_AREA_SIZE<=20){
                    script.Initialize(node, 1);
                }
                if(RANDOM_AREA_SIZE>20){
                    script.Initialize(node,RANDOM_AREA_SIZE/20);
                }
                // Initialize data

                // Add to list
                GraphNodes.Add(script);
            }
        }

        /// <summary>
        /// Displays links on the graph.
        /// </summary>
        private void DisplayLinks()
        {
            for(int i = 0; i<Network.Nodes.Count; i++){
                for(int j = i; j<Network.Nodes.Count; j++){
```

```
                    if(Network.Links[i,j]==1){
                        GraphNode firstNode = GraphNodes[i];
                        GraphNode secondNode = GraphNodes[j];

                        // Create a new entity instance
                        GameObject graphLink = Instantiate(LinkTemplate,
LinksParent.transform);
                        graphLink.transform.position = Vector3.zero;
                        graphLink.transform.localRotation =
Quaternion.Euler(Vector3.zero);

                        // Extract the script
                        GraphLink script = graphLink.GetComponent<GraphLink>();

                        // Initialize data
                        script.Initialize(firstNode, secondNode);

                        // Add to list
                        GraphLinks.Add(script);
                    }
                }
            }
        }

        /// <summary>
        /// Shuffles the nodes randomly.
        /// </summary>
        private void ShuffleNodes()
        {
            System.Random random = new System.Random();
            foreach (var node in GraphNodes)
                node.ApplyForces(new List<Vector2>() { new Vector2(random.Next(-
10, 10) / 10f, random.Next(-10, 10) / 10f) }, true);
        }

        #endregion

        #region Force Methods

        /// <summary>
        /// Continuously apply forces to nodes.
        /// </summary>
        private void Update()
        {
            if(!IsStatic && _DisplayMode == 0)
                ApplyForces();
        }

        /// <summary>
        /// Computes and applies forces to nodes.
        /// </summary>
        private void ApplyForces()
        {
            if (GraphNodes == null)
                return;

            // Stores all the forces to be applied to each node
            Dictionary<GraphNode, List<Vector2>> nodeForces = new
Dictionary<GraphNode, List<Vector2>>();
            foreach (var node1 in GraphNodes)
                nodeForces.Add(node1, new List<Vector2>());

            // Compute repulsion forces
            foreach (var node1 in GraphNodes)
                foreach (var node2 in GraphNodes)
                    if (node1 != node2)
                        nodeForces[node1].Add(ComputeRepulsiveForce(node1,
node2));
```

```
            // Compute attraction forces
            foreach (var link in GraphLinks)
            {
                var force = ComputeAttractionForce(link);
                nodeForces[link.FirstNode].Add(-force);
                nodeForces[link.SecondNode].Add(force);
            }

            // Apply forces
            foreach (var node in nodeForces.Keys)
                node.ApplyForces(nodeForces[node]);
        }

        /// <summary>
        /// Computes the distance between two nodes.
        /// </summary>
        private float ComputeDistance(GraphNode node1, GraphNode node2)
        {
            return (float)
                Math.Sqrt
                (
                    Math.Pow(node1.transform.position.x -
node2.transform.position.x, 2)
                    +
                    Math.Pow(node1.transform.position.y -
node2.transform.position.y, 2)
                );
        }

        /// <summary>
        /// Computes the repulsive force against a node.
        /// </summary>
        private Vector2 ComputeRepulsiveForce(GraphNode node, GraphNode
repulsiveNode)
        {
            // Compute distance
            float distance = ComputeDistance(node, repulsiveNode);
            if (distance > REPULSION_DISTANCE)
                return Vector3.zero;

            // Compute force direction
            Vector2 forceDirection = (node.transform.position -
repulsiveNode.transform.position).normalized;

            // Compute distance force
            float distanceForce = (REPULSION_DISTANCE - distance) /
REPULSION_DISTANCE;

            // Compute repulsive force
            return forceDirection * distanceForce * REPULSION_FORCE *
Time.deltaTime;
        }

        /// <summary>
        /// Computes the attraction force between two nodes.
        /// </summary>
        private Vector2 ComputeAttractionForce(GraphLink link)
        {
            // Compute force direction
            Vector2 forceDirection = (link.FirstNode.transform.position -
link.SecondNode.transform.position).normalized;

            // Compute repulsive force
            return forceDirection * link.Width * ATTRACTION_FORCE *
Time.deltaTime;
        }

        #endregion
```

```
        }
    }
```

GraphNode.cs – контролер відображення вершин

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

namespace ForceDirectedGraph
{
    public class GraphNode : MonoBehaviour
    {

        #region Constants

        /// <summary>
        /// The maximum value the node's velocity can be at any time.
        /// </summary>
        private const float MAX_VELOCITY_MAGNITUDE = 400f;

        #endregion

        #region Initialization

        /// <summary>
        /// Executes once on start.
        /// </summary>
        private void Awake()
        {
            _Rigidbody = GetComponent<Rigidbody2D>();
            Draggable = GetComponent<Draggable>();
            Sprite = GetComponent<SpriteRenderer>();

            // Freeze rotation
            _Rigidbody.angularVelocity = 0;
            _Rigidbody.freezeRotation = true;

            _IsStatic = false;
            _Mode = 0;
        }

        /// <summary>
        /// Initializes the graph entity.
        /// </summary>
        /// <param name="node">The node being presented.</param>
        public void Initialize(Node node, float scale)
        {
            _Node = node;
            Vector3 nodeScale = new Vector3(scale, scale, scale);
            this.transform.localScale = nodeScale;

            // Set the color
            NodeColor = Color.white;

        }

        #endregion

        #region Fields/Properties

        private bool _IsStatic;
        public bool IsStatic {set { _IsStatic = value; }}

        /// <summary>
        /// The node being presented.
        /// </summary>
        [SerializeField]
        [Tooltip("The node being presented.")]
        private Node _Node;
```

```
        /// <summary>
        /// The node being presented.
        /// </summary>
        public Node Node { get { return _Node; } }



        /// <summary>
        /// References the rigid body that handles the movements of the node.
        /// </summary>
        private Rigidbody2D _Rigidbody;
        public Rigidbody2D Rigidboby { get { return _Rigidbody; } }

        /// <summary>
        /// References the draggable script that will notify us if the node is
being dragged.
        /// </summary>
        private Draggable Draggable;

        private SpriteRenderer Sprite;

        /// <summary>
        /// List of all forces to apply.
        /// </summary>
        private List<Vector2> Forces;

        private Color NodeColor;

        [SerializeField]
        private Gradient conserv;

        [SerializeField]
        private Gradient comunicab;

        [SerializeField]
        private Gradient opinion;

        private int _Mode;
        public int Mode { set { _Mode = value; } }

        #endregion

        #region Movement

        /// <summary>
        /// Apply forces to the node.
        /// </summary>
        /// <param name="applyImmediately">States whether we should apply the
forces immediately or wait till the next frame.</param>
        public void ApplyForces(List<Vector2> forces, bool applyImmediately =
false)
        {
            if (applyImmediately)
                foreach (var force in forces)
                    _Rigidbody.AddForce(force);
            else
                Forces = forces;
        }

        /// <summary>
        /// Updates the forces applied to the node.
        /// </summary>
        private void Update()
        {
            if(_IsStatic == false){

                // Check if the object is being dragged
                if (Draggable.IsBeingDragged)
                {
```

```
                    // Do nothing
                }

                // The object is not being dragged
                else
                {
                    _Rigidbody.velocity = Vector3.zero;

                    Vector2 velocity = Vector2.zero;
                    if (Forces != null)
                        foreach (var force in Forces)
                            velocity += force;

                    velocity = velocity.normalized *
Mathf.Clamp(velocity.magnitude, 0f, MAX_VELOCITY_MAGNITUDE);

                    _Rigidbody.AddForce(velocity);
                }
            }

            UpdateColor();
        }

        private void UpdateColor(){
            CalculateColor(_Mode);
            Sprite.color = NodeColor;
        }

        private void CalculateColor(int mode){
            switch (mode)
            {
                case 0:
                    if(Node.Informed)
                        NodeColor = Color.yellow;
                    if(!Node.Informed)
                        NodeColor = Color.white;
                    break;

                case 1:
                    if(Node.RepostMarker)
                        NodeColor = Color.red;
                    if(!Node.RepostMarker)
                        NodeColor = Color.white;
                    break;
                case 2:
                    NodeColor = conserv.Evaluate(Node.Conservatism);
                    break;
                case 3:
                    NodeColor = comunicab.Evaluate(Node.Comunicability);
                    break;
                case 4:
                    float val = (Node.Opinion+1)/2;
                    NodeColor = opinion.Evaluate(val);
                    break;
            }
        }

        #endregion

    }
}
```

GraphLink.cs – контролер відображення ребер

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

namespace ForceDirectedGraph
{
    public class GraphLink : MonoBehaviour
    {

        #region Initialization

        /// <summary>
        /// Executes once on start.
        /// </summary>
        private void Awake()
        {
            LineRenderer = GetComponent<LineRenderer>();

            _IsStatic = false;
        }

        /// <summary>
        /// Initializes the graph entity.
        /// </summary>
        /// <param name="link">The link being presented.</param>
        /// <param name="firstNode">The first graph node this entity is attached
to.</param>
        /// <param name="secondNode">The second graph node this entity is
attached to.</param>
        public void Initialize(GraphNode firstNode, GraphNode secondNode)
        {
            _FirstNode = firstNode;
            _SecondNode = secondNode;
            _FirstNodeOffset = (_FirstNode.transform.localScale.x/10);
            _SecondNodeOffset = (_SecondNode.transform.localScale.x/10);

            // Set color
            LineRenderer.startColor = Color.grey;
            LineRenderer.endColor = Color.grey;

            // Set width
            _Width = 0.05f;
            float width = _Width == 0 ? 0 : _Width * 0.08f + 0.02f; // [0.02 ->
0.1]
            LineRenderer.startWidth = width;
            LineRenderer.endWidth = width;
        }

        #endregion

        #region Fields/Properties

        private bool _IsStatic;
        public bool IsStatic {set { _IsStatic = value; }}

        /// <summary>
        /// The first graph node this entity is attached to.
        /// </summary>
        [SerializeField]
        private GraphNode _FirstNode;

        /// <summary>
        /// The first graph node this entity is attached to.
        /// </summary>
        public GraphNode FirstNode { get { return _FirstNode; } }
```

```
        /// <summary>
        /// The second graph node this entity is attached to.
        /// </summary>
        [SerializeField]
        private GraphNode _SecondNode;

        /// <summary>
        /// The second graph node this entity is attached to.
        /// </summary>
        public GraphNode SecondNode { get { return _SecondNode; } }

        private float _FirstNodeOffset;

        private float _SecondNodeOffset;


        /// <summary>
        /// Normalized width of the link [0-1].
        /// </summary>
        [SerializeField]
        [Tooltip("Normalized width of the link [0-1].")]
        private float _Width;

        /// <summary>
        /// Normalized width of the link [0-1].
        /// </summary>
        public float Width { get { return _Width; } }



        /// <summary>
        /// References the line renderer that displays the link.
        /// </summary>
        private LineRenderer LineRenderer;

        #endregion

        #region Methods

        /// <summary>
        /// Update the line to keep the two nodes connected.
        /// </summary>
        private void Update()
        {
            if(_IsStatic == false){

                LineRenderer.useWorldSpace = true;

                Vector3 firstPosition = _FirstNode.transform.position +
(_SecondNode.transform.position - _FirstNode.transform.position).normalized *
_FirstNodeOffset;
                Vector3 secondPosition = _SecondNode.transform.position +
(_FirstNode.transform.position - _SecondNode.transform.position).normalized *
_SecondNodeOffset;

                LineRenderer.SetPosition(0, firstPosition);
                LineRenderer.SetPosition(1, secondPosition);

            }
        }

        #endregion

    }
}
```

Dragable.cs – компонент для переміщення вершин

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

namespace ForceDirectedGraph
{
    public class Draggable : MonoBehaviour
    {

        #region Constants

        /// <summary>
        /// Force applied when dragging objects.
        /// </summary>
        private const float DRAGGING_FORCE = 20000f;

        #endregion

        #region Initialization

        /// <summary>
        /// Executes once on start.
        /// </summary>
        private void Awake()
        {
            Rigidbody = GetComponent<Rigidbody2D>();
        }

        #endregion

        #region Fields/Properties

        /// <summary>
        /// References the rigid body that handles the movements of the node.
        /// </summary>
        private Rigidbody2D Rigidbody;

        /// <summary>
        /// States whether the object is currently being dragged or not.
        /// </summary>
        public bool IsBeingDragged { private set; get; }

        #endregion

        #region Methods

        /// <summary>
        /// Starts the dragging process.
        /// </summary>
        private void OnMouseDown()
        {
            IsBeingDragged = true;
        }

        /// <summary>
        /// Stops the dragging process.
        /// </summary>
        private void OnMouseUp()
        {
            IsBeingDragged = false;
            Rigidbody.velocity = Vector3.zero;
            Rigidbody.angularVelocity = 0;
        }

        /// <summary>
        /// Drags the object around.
        /// </summary>
```

```
        private void Update()
        {
            if (!IsBeingDragged)
                return;

            // Get mouse position
            Vector3 mousePosition =
Camera.main.ScreenToWorldPoint(Input.mousePosition);
            mousePosition = new Vector3(mousePosition.x, mousePosition.y, 0);

            // Get the force vector
            Vector3 force = mousePosition - transform.position;
            force *= DRAGGING_FORCE * Time.deltaTime;

            // Apply force
            Rigidbody.velocity = Vector3.zero;
            Rigidbody.AddForce(force);
        }

        #endregion

    }
}
```

Network.cs – клас Мережа та його методи

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

namespace ForceDirectedGraph.DataStructure
{
    public class Network
    {

        #region Constructors

        public Network(int NodesCount)
        {
            _Nodes = new List<Node>();
            _Links = new int[NodesCount,NodesCount];

            for(int i = 0; i<NodesCount; i++){
                for(int j = 0; j<NodesCount; j++){
                    _Links[i,j] = 0;
                }
            }
        }

        #endregion

        #region Properties

        /// <summary>
        /// List of all nodes in the network.
        /// </summary>
        [SerializeField]
        [Tooltip("List of all nodes in the network.")]
        private List<Node> _Nodes;

        /// <summary>
        /// List of all nodes in the network.
        /// </summary>
        public List<Node> Nodes { get { return _Nodes; } }



        /// <summary>
        /// List of all links connecting the nodes.
        /// </summary>
        [SerializeField]
        [Tooltip("List of all links connecting the nodes.")]
        private int[,] _Links;

        /// <summary>
        /// List of all links connecting the nodes.
        /// </summary>
        public int[,] Links { get { return _Links; } }

        #endregion

        #region Methods

        public void CreateLink(int FirstNodeID, int SecondNodeID)
        {
            _Links[FirstNodeID, SecondNodeID] = 1;
        }

        public void ShowLinksMatrix()
        {
            Debug.Log(_Nodes.Count);
            for(int i = 0; i<_Nodes.Count; i++){
```

```
                string massage = "";
                for(int j = 0; j<_Nodes.Count; j++){
                    massage+=_Links[i,j]+" ";
                }
                Debug.Log(massage);
            }
        }

        public List<Node> FindNeighbors(int NodeID)
        {
            List<Node> Neighbors = new List<Node>();
            for(int i = 0; i< _Nodes.Count; i++){
                if(_Links[NodeID, i] == 1){
                    Neighbors.Add(_Nodes[i]);
                }
            }

            return Neighbors;
        }

        #endregion

    }
}
```

Node.cs – клас Вершина та його методи

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

namespace ForceDirectedGraph.DataStructure
{
    [Serializable]
    public class Node
    {

        #region Constructors

        public Node(int NodeID)
        {
            System.Random rand = new
System.Random(Guid.NewGuid().GetHashCode());
            _Id = NodeID;
            _Informed = false;
            _Opinion = NextFloat(-1, 1, rand);
            _Conservatism = NextFloat(0,1, rand);
            _Comunicability = NextFloat(0,1, rand);
            _Opinions_Got = new List<float>();
            _RepostMarker = false;
        }

        #endregion

        #region Properties

        /// <summary>
        /// Ідентивікатор.
        /// </summary>
        [SerializeField]
        private int _Id;

        public int Id { get { return _Id; } }

        /// <summary>
        /// Показник інформованості.
        /// </summary>
        [SerializeField]
        private bool _Informed;

        public bool Informed { get { return _Informed; } set { _Informed =
value; } }

        /// <summary>
        /// Оцінка інформації.
        /// </summary>
        [SerializeField]
        private float _Opinion;

        public float Opinion { get { return _Opinion; } }

        /// <summary>
        /// Коефіцієнт консерватизму.
        /// </summary>
        [SerializeField]
        private float _Conservatism;

        public float Conservatism { get { return _Conservatism; } }

        /// <summary>
        /// Коефіцієнт комунікабельності.
        /// </summary>
        [SerializeField]
```

```
        private float _Comunicability;

        public float Comunicability { get { return _Comunicability; } }

        /// <summary>
        /// Прийняті думки за останній крок.
        /// </summary>
        [SerializeField]
        private List<float> _Opinions_Got;

        public List<float> Opinions_Got { get { return _Opinions_Got; } }

        /// <summary>
        /// Індинатор репосту.
        /// </summary>
        [SerializeField]
        private bool _RepostMarker;

        public bool RepostMarker { get { return _RepostMarker; } }

        #endregion

        #region Methods

        public void CalculateRepostMarker(){
            if(((_Opinion < 0.64f) && (_Opinion > -0.64f) && (_Comunicability <
0.3f)) || (Informed == false)){
                _RepostMarker = false;
            } else {
                _RepostMarker = true;
            }
        }

        public void CalculateCurrentOpinion(){
            if(_Opinions_Got.Count != 0){
                float CSP = 0;
                float opinions_sum = 0;
                foreach(float op in _Opinions_Got){
                    opinions_sum += op;
                }
                CSP = opinions_sum/_Opinions_Got.Count;
                float current_op = _Opinion + (1 - _Conservatism) * CSP;
                if(current_op<-1){
                    current_op = -1;
                }
                if(current_op>1){
                    current_op = 1;
                }
                _Opinion = current_op;
                _Opinions_Got = new List<float>();
            }
        }

        private float NextFloat(float min, float max, System.Random random){
            double val = (random.NextDouble() * (max - min) + min);
            return (float)val;
        }

        #endregion

    }
}
```

SimulationManager.cs – контролер симуляції

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

namespace ForceDirectedGraph
{
    public class SimulationManager : MonoBehaviour
    {
        public void Initialize(DataStructure.Network network, float
oneStepTime){
            _Network = network;
            _OneStepTime = oneStepTime;

            _SimulationPoused = true;
            _SimulationStarted = true;

            SimPrepare.SetActive(false);
            SimProcces.SetActive(true);
            WarningMassage.SetActive(true);
            Massage.text = "Оберіть агентів розповсюдження інформації!!!";

            _SimulationStep = 0;
        }

        public void StopSimulation(){
            _Network = null;
            _OneStepTime = 0;

            _SimulationPoused = true;
            _SimulationStarted = false;

            InformedCount.text = "0";
            RepostingCount.text = "0";
            AvgOpinion.text = "0";
            SimStepText.text = "0";

            SimPrepare.SetActive(true);
            SimProcces.SetActive(false);

            _SimulationStep = 0;
        }


        private DataStructure.Network _Network;
        private float _OneStepTime;
        private bool _SimulationPoused;
        private bool _SimulationStarted;
        public bool SimulationStarted {get { return _SimulationStarted; } }
        private int _SimulationStep;
        public int SimStep {get { return _SimulationStep; } }

        [SerializeField]
        private GameObject SimProcces;
        [SerializeField]
        private GameObject SimPrepare;
        [SerializeField]
        private GameObject WarningMassage;

        [SerializeField]
        private Text Massage;

        [SerializeField]
        private Text SimStepText;
        [SerializeField]
        private InputField InformedCount;
        [SerializeField]
```

```
        private InputField RepostingCount;
        [SerializeField]
        private InputField AvgOpinion;

        public void SimulationStep(){
            int informedCount = 0;
            if(_SimulationStep == 0){
                foreach(DataStructure.Node node in _Network.Nodes){
                    if(node.Informed == true)
                        informedCount++;
                }
                if(informedCount < 1){
                    WarningMassage.SetActive(true);
                    Massage.text = "Оберіть агентів розповсюдження
інформації!!!";
                    return;
                }
                else
                    _SimulationStep++;
            }

            SimStepText.text = _SimulationStep.ToString();

            informedCount = 0;
            int repostingCount = 0;
            float avgOpinion = 0.0f;

            foreach(DataStructure.Node node in _Network.Nodes){
                node.CalculateRepostMarker();
                if(node.RepostMarker == true)
                    repostingCount++;
            }
            foreach(DataStructure.Node node in _Network.Nodes){
                if(node.RepostMarker == true){
                    List<DataStructure.Node> neighbors =
_Network.FindNeighbors(node.Id);
                    foreach(DataStructure.Node neighbor in neighbors){
                        neighbor.Informed = true;
                        neighbor.Opinions_Got.Add(node.Opinion);
                    }
                }
            }
            foreach(DataStructure.Node node in _Network.Nodes){
                node.CalculateCurrentOpinion();
                avgOpinion += node.Opinion;
                if(node.Informed == true)
                    informedCount++;
            }

            avgOpinion = avgOpinion/_Network.Nodes.Count;

            InformedCount.text = informedCount.ToString();
            RepostingCount.text = repostingCount.ToString();
            AvgOpinion.text = avgOpinion.ToString("0.00");

            _SimulationStep++;
        }

        public void PouseSimulanion(){
            _SimulationPoused = true;
            Debug.Log("Simulation Poused");
        }

        public void PlaySimulanion(){
            if(_SimulationStep == 0){
                int informedCount = 0;
                foreach(DataStructure.Node node in _Network.Nodes){
                    //node.CalculateRepostMarker();
                    if(node.Informed == true)
```

```
                    informedCount++;
                }
                if(informedCount < 1){
                    WarningMassage.SetActive(true);
                    Massage.text = "Оберіть агентів розповсюдження
інформації!!!";
                    return;
                }
                else
                    _SimulationStep++;
            }
            _SimulationPoused = false;
            Debug.Log("Simulation Plaing");
        }

        private float nextStep = 0.0f;
        void Update()
        {
            if(_SimulationStarted == true && _SimulationStep == 0){
                int informedCount = 0;
                foreach(DataStructure.Node node in _Network.Nodes){
                    //node.CalculateRepostMarker();
                    if(node.Informed == true)
                        informedCount++;
                }
                InformedCount.text = informedCount.ToString();
            }
            if(_SimulationPoused == false && _SimulationStep >= 1 && Time.time >
nextStep){
                nextStep = Time.time + _OneStepTime;
                SimulationStep();
            }

        }
    }
}
```

UIManager.cs – контролер інтерфейсу

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

public class UIManager : MonoBehaviour
{
    [SerializeField]
    private MainSceneManager SceneManager;

    [SerializeField]
    private GraphManager graph;

    [SerializeField]
    private Dropdown GraphDisplayMode;

    [SerializeField]
    private InputField NumberOfMembers;

    [SerializeField]
    private InputField NumberOfNewMemberConnections;

    [SerializeField]
    private InputField OneStepTime;

    [SerializeField]
    private Toggle IsStatik;

    [SerializeField]
    private GameObject WarningMassage;

    [SerializeField]
    private Text Massage;

    public void GenerateGraph(){
        if(NumberOfMembers.text == ""){
            WarningMassage.SetActive(true);
            Massage.text = "Заповніть усі поля!!!";
            return;
        }
        if(NumberOfNewMemberConnections.text == ""){
            WarningMassage.SetActive(true);
            Massage.text = "Заповніть усі поля!!!";
            return;
        }

        int numberOfMembers = int.Parse(NumberOfMembers.text);
        int numberOfNewMemberConnections =
int.Parse(NumberOfNewMemberConnections.text);

        if(numberOfMembers < 2 || numberOfMembers > 5000){
            WarningMassage.SetActive(true);
            Massage.text = "Поля заповнені не вірно!!!";
            return;
        }
        if(numberOfNewMemberConnections < 2 || numberOfNewMemberConnections > 6
|| numberOfNewMemberConnections > numberOfMembers){
            WarningMassage.SetActive(true);
            Massage.text = "Поля заповнені не вірно!!!";
            return;
        }

        SceneManager._NumberOfMembers = numberOfMembers;
        SceneManager._NewMemberConnectionsCount = numberOfNewMemberConnections;
        SceneManager._GraphMode = GraphDisplayMode.value;
```

```
            SceneManager.GenerateBarabasiAlbertNetwork();

            IsStatik.isOn = false;
        }

        public void StartSimulation(){
            if(OneStepTime.text == ""){
                WarningMassage.SetActive(true);
                Massage.text = "Заповніть усі поля!!!";
                return;
            }

            float oneStepTime = float.Parse(OneStepTime.text);

            if(oneStepTime < 0.5f){
                WarningMassage.SetActive(true);
                Massage.text = "Поля заповнені не вірно!!!";
                return;
            }

            SceneManager._OneStepTime = oneStepTime;

            SceneManager.StartSimulation();
        }


        public void SetStatic(){
            graph.SetStatic(IsStatik.isOn);
        }

    }
```

SelectManager.cs – контролер вибору

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


public class SelectManager : MonoBehaviour
{
    private SimulationManager simMng;
    private GraphNode graphNode;


    // Start is called before the first frame update
    void Start()
    {
        simMng =
GameObject.Find("SimulationManager").GetComponent<SimulationManager>();
        graphNode = GetComponent<GraphNode>();
    }


    void OnMouseUp() {
        Debug.Log("HIT!!!");
        if(simMng.SimulationStarted == true && simMng.SimStep == 0){
            if(graphNode.Node.Informed == false)
                graphNode.Node.Informed = true;
            else
                graphNode.Node.Informed = false;
        }
    }


}
```

CameraController.cs – контролер камери

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

public class CameraController : MonoBehaviour
{
    private Transform _Transform;
    private Camera _Camera;
    private float zoom = 5f;

    [SerializeField]
    private float moveAmount;

    [SerializeField]
    private float zoomAmount;

    // Start is called before the first frame update
    void Start()
    {
        _Transform = GetComponent<Transform>();
        _Camera = GetComponent<Camera>();
    }

    // Update is called once per frame
    void Update()
    {
        if(Input.GetKey(KeyCode.W)){
            _Transform.position = _Transform.position + new Vector3(0,
moveAmount * Time.deltaTime, 0);
        }
        if(Input.GetKey(KeyCode.S)){
            _Transform.position = _Transform.position + new Vector3(0, -
1*(moveAmount * Time.deltaTime), 0);
        }
        if(Input.GetKey(KeyCode.A)){
            _Transform.position = _Transform.position + new Vector3(-
1*(moveAmount * Time.deltaTime), 0, 0);
        }
        if(Input.GetKey(KeyCode.D)){
            _Transform.position = _Transform.position + new Vector3(moveAmount *
Time.deltaTime, 0, 0);
        }

        if(Input.GetKey(KeyCode.KeypadPlus)){
            zoom -= zoomAmount * Time.deltaTime;
        }
        if(Input.GetKey(KeyCode.KeypadMinus)){
            zoom += zoomAmount * Time.deltaTime;
        }

        zoom = Mathf.Clamp(zoom, 2f, 100f);

        _Camera.orthographicSize = zoom;
    }
}
```