Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія проходження лабіринту на базі генетичного алгоритму»

Виконав: студент 2-го курсу, групи 2КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Канаєв Є. Ю.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
_____ Арсенюк І. Р.
(прізвище та ініціали)
«15» 12 2022 р.

Опонент: к.т.н., доцент каф. АІІТ
_____ Маслій Р. В.
(прізвище та ініціали)
«15» 12 2022 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«16» 12 2022 р.

Вінниця ВНТУ – 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.

15.09 **2022 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Канаєв Євгеній Юрійович
(прізвище, ім'я, по батькові)

1. Тема магістерської кваліфікаційної роботи: «Інформаційна технологія проходження лабіринту на базі генетичного алгоритму»
керівник роботи_ к. т. н., доцент кафедри КН Арсенюк І. Р.
Затверджені наказом вищого навчального закладу від "14" 09 2022 року №203
2. Строк подання студентом роботи 18 листопада 2022 року
3. Вихідні дані до роботи:
вид лабіринту повинен бути в 2D вимірі, мінімальна розмірність лабіринту повинна бути 10x10, максимальна розмірність – 100x100, мінімальна розмірність популяції в алгоритмі – 10, операційна система – Windows 7/8/10/11, мова програмування – повинна мати ООП та забезпечувати можливість маніпулювання даними і управління базами даних.
4. Зміст текстової частини:
Вступ, аналіз предметної області проходження лабіринтів, розробка інформаційної технології проходження лабіринту, програмна реалізація інформаційної технології проходження лабіринту, економічна частина, висновки, перелік використаних джерел, додатки.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
Схема основних вузлів роботи генетичного алгоритму, представлення хромосоми, операція кросовера, операція мутації, схема роботи архітектурного шаблону MVC, математична модель функціонування системи, загальна UML-діаграма класів, приклад роботи програмного засобу проходження лабіринту на базі генетичного алгоритму.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціалита посада консультанта	Підпис, дата	
		завдання видав	викона прийн
1-3	Арсенюк І. Р., к.т.н., доц. каф. КН	14.09.22	14.09.22
4	Бурєннікова Н. В., д. е. н., професор каф. ЕПВМ	19.09.22	19.09.22

7. Дата видачі завдання 14.09 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Пр мі
1	Аналіз сучасного рівня інформаційних технологій проходження лабіринту. Постановка задач дослідження	14.09.22р – 01.10.22р.	
2	Побудова моделей проходження лабіринту на базі генетичного алгоритму та функціонування генетичного алгоритму	02.10.22р – 16.10.22р.	
3	Практичне застосування та оцінка ефективності розроблених моделей	17.10.22р. – 07.11.22р.	
4	Підготовка економічної частини	08.11.22р – 21.11.22р.	
5	Апробація та/або впровадження результатів дослідження	23.11.22р – 01.12.22р.	
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	02.12.22р. – 14.12.22р.	

Студент _____ (підпис) Канаєв Є. Ю.

Керівник роботи _____ (підпис) Арсенюк І. Р.

АНОТАЦІЯ

УДК 004.8

Канаєв Є. Ю. Інформаційна технологія проходження лабіринту на базі генетичного алгоритму. Магістерська кваліфікаційна робота зі спеціальності 122 – комп’ютерні науки, освітня програма – комп’ютерні науки. Вінниця: ВНТУ, 2022. 120 с.

На укр. мові. Бібліогр.: 32 назв; рис.: 15; табл. 7.

Дана магістерська кваліфікаційна робота присвячена розробці програмного забезпечення для проходження лабіринту. Були розглянуті та проаналізовані існуючі алгоритми проходження лабіринтів, як найбільш перспективний, було обрано генетичний алгоритм. Обґрунтовано підхід до розв’язання задачі подолання лабіринту. Розроблено алгоритм проходження лабіринту. Створено UML-діаграму класів.

Відповідно до розробленої моделі, схеми алгоритму та UML-діаграми класів програмно реалізовано додаток. Обґрунтовано вибір середовища (Visual Studio) та мови програмування (C#). Розроблено програмне забезпечення та здійснено його тестування, яке довело правильність роботи алгортиму.

Ключові слова: проходження, лабіринт, генетичний алгоритм, шлях, сутності, покоління, генетична пам’ять.

ABSTRACT

UDC 004.8

Kanaev Y. Y. Information technology of passing the maze based on the genetic algorithm. Master's qualification thesis on specialty 122 - computer science, educational program - computer science. Vinnytsia: VNTU, 2022. 120 p.

In Ukrainian speech Bibliography: 32 titles; Fig.: 15; table 7.

This master's thesis is devoted to the development of software for passing the maze. The existing algorithms for passing the labyrinths were considered and analyzed, and the genetic algorithm was chosen as the most promising. The approach to solving the problem of overcoming the labyrinth is substantiated. An algorithm for passing the labyrinth has been developed. A UML class diagram has been created.

In accordance with the developed model, algorithm scheme and UML class diagram, the application was implemented programmatically. The choice of environment (Visual Studio) and programming language (C#) is justified. The software was developed and tested, which proved the correctness of the algorithm.

Key words: passage, maze, genetic algorithm, path, entities, generation, genetic memory.

ЗМІСТ

ВСТУП

Актуальність. Для багатьох проблемних завдань реального світу рішення полягає в покроковому русі, де кожен вибір дозволяє просунутися далі. Завдання знаходження шляху в лабіринті належати до подібних завдань. Тому її рішення буде не менш значуще.

Лабіринт - це досить важка головоломка, в якій зазвичай є тільки одне правильне рішення. Але вряди-годи, якщо лабіринт дуже величезний, то виходів може бути деяке число, але це не означає, що вийти з такого лабіринту просто, навпаки, чим більше лабіринт, тим складніше виявити вихід, всупереч кількості виходів. Якщо людину поставити в центр великого лабіринту, то, швидше за все, вихід із нього він не виявить. З давніх-давен лабіринти несли почуття таємниці та загадки. Один із перших лабіринтів, відомих суспільству, описує Геродот – це був єгипетський лабіринт, у якому було п'ять тисяч кімнат. Пізніше лабіринти втратили своє релігійно-містичне значення і стали об'єктами веселощів, перетворившись на сади та парки у вигляді зелених огорож важкої конфігурації.

Лабіринти оточують та творять наш світ. Усі комунікаційні системи, методи передачі та зберігання інформації, Інтернет-мережі, транспортні мережі, нарешті, робота мозку та наші думки – це теж оригінальний лабіринт.

В цей час лабіринти широко застосовуються в науці та техніці. Психологи з їхньою підтримкою осягають поведінку людей і тварин у повторюваних або екстремальних ситуаціях. За тезою лабіринту виробляють глушники у двигунах внутрішнього згоряння, заповнюють частини деталей під високим тиском. Кібернетикам лабіринти допомагають конструювати ЕОМ, зокрема роботів, здатних до самонавчання. Такі експерименти першим провів американський математик Клод Шеннон: кібернетичні миші вченого за певними алгоритмами могли вибиратися з найзаплутаніших лабіринтів.

Розгадування лабіринтів завжди було цікавим заняттям, але ще цікавішим є розробка алгоритмів, здатних пройти лабіринт.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська кваліфікаційна робота виконана відповідно до напряму наукових досліджень кафедри комп'ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання досліджень. Метою наданої магістерської кваліфікаційної роботи є розвиток функціональних перспектив програмного забезпечення для проходження лабіринту.

Для досягнення поставленої мети слід вирішити наступні завдання:

- розглянути та проаналізувати існуючі лабіринти та алгоритми для їх проходження;
- запропонувати математичну модель для інформаційної технології проходження лабіринту;
- запропонувати етапи інформаційної технології та на їх основі розробити структуру та алгоритм роботи програмного засобу;
- виконати програмну реалізацію запропонованої інформаційної технології проходження лабіринту на базі генетичного алгоритму;
- провести тестування програмного продукту та виконати аналіз отриманих результатів.

Об'єктом дослідження – процес проходження лабіринту.

Предмет дослідження – інформаційна технологія проходження лабіринту на базі генетичного алгоритму.

Методи дослідження. У роботі використані такі методи наукових досліджень: методи системного аналізу, теорія генетичного алгоритму для реалізації інформаційної технології, методи схрещення для розробки процесу передачі інформації в генетичному алгоритмі, методи оцінки достовірності їх

роботи та обрахунків результатів експериментів із програмним забезпеченням, об’єктно-орієнтованого програмування.

Наукова новизна одержаних результатів. Набула подальшого розвитку інформаційна технологія проходження лабіринту на базі генетичного алгоритму, яка відрізняється збільшенням функціональних можливостей налаштування параметрів роботи генетичного алгоритма, що дозволяє швидше знаходити шлях у лабіринті.

Практичне значення одержаних результатів полягає в тому, що на основі проведених досліджень розроблено програмне забезпечення проходження лабіринту. Запропонована інформаційна технологія сприяє підвищенню швидкодії роботи програмних засобів проходження лабіринту, зокрема:

– розроблено алгоритм роботи програмного забезпечення проходження лабіринту на базі генетичного алгоритму;

– розроблено програмний продукт проходження лабіринту на базі генетичного алгоритму.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, коректністю використання математичного апарату методів дослідження, експериментальними дослідженнями тестування програмної реалізації інформаційної технології проходження лабіринту. Адекватність розроблених математичних моделей підтверджується результатами експериментальних досліджень.

Особистий внесок здобувача. Усі результати, що наведені у магістерській кваліфікаційній роботі, отримані самостійно. У працях, які написано у співавторстві, здобувачу належать: гнучке налаштування параметрів роботи генетичного алгоритма, що дозволяє пришвидшити процес пошуку шляху у лабіринті [1].

Апробація результатів роботи. Результати досліджень було апробовано на XVI міжнародній науково-практичній конференції “Modern Science: Innovations and Prospects”, Стокгольм, Швеція, 2022.

Публікації. За результатами магістерської кваліфікаційної роботи опубліковано матеріали XVI міжнародної науково-практичної конференції “Modern Science: Innovations and Prospects”, Стокгольм, Швеція, 2022 [1]. Отримано свідоцтво про реєстрацію авторського права на твір № 115466 (дата реєстрації 26.10.2022) – комп’ютерна програма «Програмний модуль подолання лабіринту на базі генетичного алгоритму» [2]. За результатами бакалаврської дипломної роботи опубліковано тези доповіді на науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії. – Вінниця: ВНТУ, 2021 [3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПРОХОДЖЕННЯ ЛАБІРИНТІВ

1.1 Постановка задачі проходження лабіринтів

Лабіринти – одна з важких, ще не дозволених загадок історії. У час ці незвичайні твори у вигляді печер, палаців чи споруд без покрівлі тощо. виникали всюди, де проживала людина.

У період середніх століть у Європі схеми лабіринтів вимазували мозаїкою на підлогах храмів. Наприклад, такий лабіринт вимостили у XII ст. на підлозі Шартрського храму у Франції. Пізніше поширені паркові лабіринти. Особливо знаменитий з них – парковий лабіринт з кущів, влаштований в 1699 р. біля Лондона у володінні Вільгельма Оранського в саду Гемптон-Корта.

Навколо цитаделі у давнину будували системи валів у формі лабіринту, план якого знав лише власник цитаделі. Такі споруди служили для оборони та могли стати хованням. Лабіринти застосовували для покарання. Засуджених до загибелі відводили до лабіринту. Там, не знаючи його будівлі, приречений після марних поневірянь помирав від спраги та голоду [4].

Найбільшого поширення набули лабіринти-головоломки. Вже діти стародавніх греків і римлян заповнювали дозвілля такими розвагами, про що свідчить креслення, знайдене на стіні одного з будників міста Помпеї. У креслення лабіринту написано: Лабіринт. Тут мешкає Мінотавр». З того часу і до наших днів ідея лабіринту стає у захватній математиці все більш ґрунтовною, збагачується новими мотивами завдань.

Урізноманітнюються самі форми лабіринтів, з'являються числові, об'ємні та інші лабіринти, де у непередбачених формах прогресує ідея, джерела якої губляться у найдавніших часах. Історії лабіринтів присвячено багато наукових досліджень і знаменитих видань.

У сучасній мові «лабіринт» також позначає не лише споруду, з якої важко виявити вихід, а й заплутані стосунки, ситуації, думки тощо. Вивчення даної теми є неперевершено актуальним, адже розробка, огляд та використання

результативних та багатофункціональних способів вирішення завдань (алгоритмів) є ключовим завданням комп'ютерних наук, а тема еволюційних алгоритмів ще недосвідчено досліджена. Спочатку означимо деякі уявлення [5].

Входом у лабіринт називається те місце, звідки ви починаєте шлях; Зазвичай вхід розміщується на периферії лабіринту.

Метою назвемо точку, до якої потрібно прийти. Ціль може знаходитися в будь-якому місці лабіринту, у тому числі на виході. Вузлом будемо вважати вхід, ціль, а ще будь-яку точку, де коридор розгалужується чи закінчується глухим кутом. Відрізок шляху між сусідніми вузлами назвемо гілкою. Шлях – це послідовність гілок. Стінка – це одна із 2-х сторін шляху. Що таке стіна для підземного лабіринту, мабуть, і не потребує пояснень.

У садовому лабіринті, наприклад, стіною може бути живий пліт або низький насип, що обмежує шлях з обох боків. Існує ряд різних алгоритмів для вирішення лабіринтів, тобто способів механічного пошуку виходу. Тремо розроблені для проходження лабіринту туристом без завчасного розуміння лабіринту, своєю чергою алгоритми: заповнення тупиків, алгоритм найкоротшого шляху лабіринт одночасно.

Лабіринти, які містять петель, ведені як «однозв'язні» чи «вчинені» лабіринти, вони рівнозначні дереву теоретично графів. Таким чином, багато алгоритмів рішення лабіринту тісно пов'язані з теорією графів. Підсвідомо, складаючи будь-який такий лабіринт, можна було б уявити його як дерева.

Отже, у даній роботі потрібно реалізувати програмний додаток для проходження лабіринту на основі генетичного методу. Потрібно розробити програмний додаток, в якому виробництва зможуть знайти шлях у 2D-лабіринті з мінімальною розмірністю 10x10.

Зокрема з допомогою генетичного алгоритму, у якому мінімальна розмірність населенню має бути п'ятдесят істот. Фінальним результатом роботи програмного забезпечення є найкоротший шлях від входу в лабіринт до виходу, а також пошук усіх допустимих виходів [6].

1.2 Огляд та класифікація видів лабіринту

Лабіринт – кожна структура (зазвичай у двовимірному чи тривимірному просторі), що складається із заплутаних шляхів до виходу (і/або шляхів, які ведуть глухий кут). Під лабіринтом у древніх греків і римлян малося на увазі більш-менш широкий простір, що складається з незліченних залів, камер, дворів і переходів, розташованих за важким і заплутаним планом, з метою заплутати та не дати виходу недосвідченому в плані лабіринту людині. Лабіринти загалом можуть бути систематизовані за семи різними параметрами. Це мірність, топологія, мозаїка, маршрутизація, текстура та фокус. Розглянемо деякі види лабіринтів різних класів. Мірність: клас відповідає за кількість вимірів у просторі, які лабіринт покриває. Типи:

− 2D (двовимірний) - безліч лабіринтів, на папері або в реальному житті, є двовимірними, тобто можна зобразити їх схему на аркуші паперу і переміщатися по ній, не перекриваючи ніяких інших проходів в лабіринті.

− 3D (3-вимірний лабіринт) - це лабіринт, в якому кілька ярусів, де проходи можуть йти вгору і вниз, на додаток до чотирьох сторін світу. 3D лабіринти зазвичай зображатися на екрані як масив 2d ярусів, зі сходами вгору і вниз.

− Higher-dimensions (багатомірні): допустимі 4d та лабіринти з величезним числом вимірювань. Вони представляються у вигляді 3d лабіринтів зі спеціальними "порталами", через які можна переміститися в 4-й вимір.

− Weave (переплетений): переплетеними називаються в основному 2d (або вірніше 2.5d) лабіринтів, де проходи можуть перекривати один одного. Реальні лабіринти, в яких є мости, що з'єднують одну частину лабіринту з іншою, частково переплетені.

Топологія: клас топології визначає геометрію простору, у якому загалом існує лабіринт:

− Planair – термін належить до лабіринту з неправильною топологією. Це означає, що краї лабіринту об'єднані незвичайними способами. Наприклад,

лабіринт лежить на поверхні куба. Мозаїка: клас складання мозаїки відповідає за геометрію окремих клітин, що становлять лабіринт: Ортогональний – стандартна прямокутна сітка, де клітини мають проходи, що перетинаються під прямим кутом. У контексті складання мозаїки це можна викликати Гамма-лабіринтом [7].

− Delta. Лабіринт Delta складається з взаємопов'язаних трикутників, де у будь-якої клітини може бути до трьох проходів, об'єднаних із нею.

− Sigma. Лабіринт Sigma складається із взаємопов'язаних шестикутників, де у будь-якої клітини може бути до шести проходів, об'єднаних із нею.

− Theta. Лабіринти Theta складається з концентричних кіл, де початок чи кінець перебуває у центрі, а інший на зовнішньому краю. Клітини зазвичай мають по чотири допустимі прохідні сполуки, але можуть мати та більше, якщо у зовнішніх прохідних кільцях буде більше клітин.

− Upsilon. Лабіринти Upsilon складається з взаємопов'язаних восьмикутників і квадратів, де у кожної клітини може бути до восьми або чотири допустимі проходи, об'єднаних із нею.

− Zeta. Лабіринт Zeta знаходиться на прямокутній сітці, але на додаток до прямих кутів можуть бути проходи між клітинами по діагоналі на сорок п'ять градусів.

− Crack. Це безформний лабіринт без безперервної мозаїки, але має стіни чи проходи у випадкових кутах.

− Fractal. Фрактальний лабіринт - лабіринт, що складається з менших лабіринтів. Вкладена клітина фрактального лабіринту – це лабіринт коїться з іншими лабіринтами, складовими мозаїчну схему між будь-якої клітини, у якій процес може повторюватися неодноразово. Безмежний рекурсивний фрактальний лабіринт – це справжній фрактал, у якому лабіринт містить копії себе й у результаті безмежно великим лабіринтом. Маршрутизація: клас маршрутизації належати до самої генерації лабіринту.

Лабіринти відрізняються за типами проходів:

− Коса – це лабіринт без глухих кутів. У такому лабіринті проходи намотуються навколо і заплітаються один в одного (від сюди термін "коса") і змушують вас витрачати час, ходячи по колу замість того, щоб врізатися в глухий кут. Відмінно розроблених лабіринтів може бути набагато важчим, ніж ідеальний лабіринт того ж розміру.

− Унікурсальний – це лабіринт без перехресть. Унікурсальний лабіринт має лише один довгий змієподібний прохід. Пройти такий лабіринт не дуже складно, якщо ненароком не заблукати на половині шляху і не повертатися назад.

− Розріджений - в даному лабіринті немає проходів через будь-яку клітину, тобто деякі проходи не зроблені. Тут можуть бути недосяжні області. Цей лабіринт можна назвати зворотним лабіринту-косою [8].

1.3 Огляд та аналіз основних алгоритмів проходження лабіринтів

Існує багато способів проходження лабіринтів, і кожен з них має індивідуальні характеристики. Нижче проаналізуємо основні подальші алгоритми.

− Дотримання по стінах. Це звичайний метод вирішення лабіринтів. Пріоритетом для нього є лабіринт, що проходить об'єкт («ви»), він безперервно дуже стрімкий і не використовує додаткову пам'ять. Починаємо йти проходами та при досягненні розвилки безперервно повертаємо праворуч (або завжди ліворуч). Щоб застосувати таке рішення в реальному світі, потрібно покласти руку на праву (або ліву) стінку і постійно утримувати її на стінці в процесі проходження лабіринту. За бажання можна позначати вже відвідані комірки та комірки, відвідані дворазово. Наприкінці можна повернутися назад за рішенням, слідуючи лише по осередках, відвіданим один раз. Даний метод не обов'язково виявить найкоротше рішення, і він абсолютно не працює, якщо мета знаходиться в центрі лабіринту і його оточує замкнутий ланцюг, тому що ви будете ходити навколо центру і з часом прийдете до початку. Дотримання по

стіни в 3D-лабіринті можна реалізувати детермінованим методом, спроектувавши 3D-проходи на 2D-площину, тобто прикинувшись, що проходи, що ведуть вгору, насправді ведуть на північний захід, а провідні вниз ведуть на південний схід, а після цього застосувати.

− Алгорифм Пледжа. Це модифікована версія проходження по стіні, здатна перестрибувати між островами для вирішення тих лабіринтів, які не здатні проходження стінами. Це гарантований спосіб досягнення виходу на зовнішньому краї будь-якого 2D-лабіринту з будь-якої точки всередині, проте він не здатний виконати обернену задачу, тобто виявити рішення всередині лабіринту. Він чудово підходить для реалізації з підтримкою робота, що збігає з лабіринту, тому що він зуміє вибратися з будь-якого лабіринту, не позначаючи та не запам'ятовуючи жодним чином шлях. Починаємо з вибору напрямку і за можливості постійно рухаємося в цьому напрямку. Упершись у стіну, починаємо слідувати по ній, поки не зуміємо знову піти у вибраному напрямку. Слід зазначити, що проходження по стіні необхідно починати з далекої стіни, в яку ми вперлися. Якщо в цьому місці прохід виготовляє поворот, це може призвести до розвороту посередині проходу і повернення тим самим шляхом, яким ми прийшли. При проходженні по стіні вважаємо число зроблених поворотів, наприклад поворот наліво - це -1, а поворот направо - це 1. Припиняємо проходження по стінах і починаємо рухатися у вибраному напрямку тільки тоді, коли загальна сума зроблених поворотів дорівнює, тобто якщо ви повернулися на триста шістдесят градусів і більше, то продовжуємо слідувати по стіні доки не розплутати. Підрахунок гарантує, що рано чи пізно ми досягнемо далекої частини «острова», в якому перебуваємо наразі, і перестрибнемо на подальший острів у вибраному напрямку, після цього продовжимо стрибати між островами, поки не впораємося в стіну кордону, після цього проходження по стінах приведе нас до виходу. Слід розглядати, що спосіб Пледжа може змусити нас відвідати прохід чи початкову точку кілька разів, але наступні рази ви завжди матимете іншу суму поворотів. Без розмітки

шляху винятковий спосіб дізнатися, що лабіринт нерозв'язних – безперервне збільшення суми поворотів, щоправда, в лабіринтах зі спіральним проходженням сума поворотів теж може домагатися великих значень.

− Алгоритм ланцюгів. Алгоритм ланцюгів (Chain algorithm) вирішує лабіринт, сприймаючи його як безліч лабіринтів меншого розміру (аналогічно ланкам ланцюга) і вирішуючи їх підряд. Ми повинні вказати необхідні місця початку і кінця, і алгоритм постійно виявить шлях від початку остаточно, якщо він існує. При цьому рішення схильне бути розумно коротким, навіть не найкоротшим. Він особливо схожий на алгоритм Пледжа, тому що по суті це алгоритм проходження по стінах з методом перестрибування між островами. Починаємо з малювання прямої лінії (або правда лінії без самобачення) від початку до кінця, дозволяючи їй при необхідності перетинати стіни. Після цього просто прямуємо по лінії від початку до кінця. Якщо ми натикаємося на стіну, то не можемо пройти через неї, а отже, маємо обминати. Відправляємо 2-х наступних по стінах «роботів» по кожному з напрямків по стінки, на яку натрапили. Якщо робот знову перетнеться з що спрямовує по лінії в тій точці, де вона ближче до виходу, тоді зупиняємося і прямуємо за цією стіною, поки не дістанемося до неї самі. Продовжуємо слідувати по лінії та повторювати процес, доки не досягнемо кінця. Якщо обидві роботи повернуться до їх початкових локацій та напрямків, то наступні точки по прямій недоступні та лабіринт не можна.

− Алгоритм Трем. Даний спосіб дозволу лабіринтів розроблений для застосування людиною усередині лабіринту. Він знаходить рішення для всіх лабіринтів: при русі проходом ми малюємо лінію за собою, позначаємо наш шлях. При попаданні в глухий кут повертаємося назад і повертаємося тим самим шляхом, яким прийшли. Дійшовши до розвилки, на якій ще не були, випадково обираємо новий прохід. Якщо ми проходимо по новому проходу і доходимо до розвідки, що відвідується раніше, то вважаємо її глухим кутом і повертаємося тим же шляхом, яким прийшли. (Цей остаточний етап є

найголовнішим, він не дозволяє рухатися колами та пропускати проходи в плетеному лабіринті.) Якщо рухаючись проходом, який ми відвідували раніше (тобто раніше позначили), ми натрапили на розвилку, то вибираємо всякий новий прохід, якщо це цілком імовірно, інакше вибираємо старий прохід (тобто той, який ми раніше позначили). Всі проходи будуть або порожніми, тобто ми їх ще не відвідували, поміченими один раз, тобто ми там були рівно один раз, або помічені дворазово, тобто ми рухалися ними та повинні були повертатися у зворотному напрямку. Коли ми нарешті досягнемо рішення, то помічені якось шляхи становитимуть прямий шлях до самого початку. Якщо лабіринту немає рішення, то ми виявимося спочатку, а всі проходи будуть позначені дворазово.

− Заповнювач глухих кутів. Це звичайний метод вирішення лабіринтів. Пріоритетом для нього є лабіринт, він дуже стрімкий і не використовує додаткову пам'ять. Ми просто скануємо лабіринт і заповнюємо будь-який глухий кут, заливаючи прохід у зворотному порядку від глухого кута, поки не досягнемо розвилки. У тому числі це стосується і заливання проходів, що стали частинами глухих кутів після видалення інших глухих кутів. Наприкінці у нас залишиться одне рішення, або кілька рішень, якщо у лабіринту їх більше одного. Алгоритм завжди знаходить для бездоганних лабіринтів одне неповторне рішення, але не особливо процвітає в лабіринтах з потужним плетінням, і насправді практично непотрібний у всіх лабіринтах без глухих кутів.

− Наповнювач Cul-de-sac. Цей метод знаходить і заповнює тупикові розв'язки або петлі, тобто конструкції в лабіринті, що складаються з тупикового шляху з винятковою петлею в кінці. Пріоритетом тут є лабіринт, алгоритм завжди швидкий і не використовує додаткову пам'ять. Скануємо лабіринт і для будь-якої петльової розвилки (петлева розвилка - це така розвилка, в якій два провідні з неї проходи з'єднуються один одному, шляхом не утворюючи нових розвилок) додаємо стіну, щоб перетворити всю петлю на довгий глухий кут. Після цього ми запускаємо dead end filler. У лабіринтах можуть бути петлі,

видатні з інших конструкцій, які стають петлями після видалення перших петель, тому цілий процес необхідно повторювати, поки при скануванні нічого не протікатиме. Даний алгоритм не надто придатний у важких плетених лабіринтах, але буде відсікати більше шляхів, ніж простий dead end filler.

− Наповнювач глухої алеї. Цей метод знаходить всі допустимі рішення в залежності від того, наскільки вони довгі або короткі. Він робить це, закладаючи всі тупикові розв'язки. Тупикова розв'язка - це така конструкція, в якій рухається в одному напрямку, для досягнення цілей приходить повернення назад тим же шляхом в інше виробництво. Всі тупики є тупиковими рішеннями, як і всі петлі, описані в алгоритмі тупикового наповнювача, а також сегменти проходів будь-якого розміру. З'єднання з залишковою частиною лабіринту одним-єдиним проходом. Пріоритет віддає лабіринт, не використовує додаткову пам'ять, але, на жаль, досить неквапливий. У будь-яку розробку ми відправляємо наступної наснаги роботи за всім проходом, що веде з не, і дивимося, повернувшись відправленим шляхом роботи по тому ж маршруту (якщо немає іншого приводу і не вийшов з лабіринту). Якщо це відбувається, то такий прохід і все після того, як він не може бути якимось шляхом рішення, тому ми перекриваємо цей прохід і заливаємо все за ним. Даний алгоритм заповнює все те, що і наповнювач глухого кута і деяка кількість ще, проте описаний нижче алгоритм вирішувача зіткнень, заповнює все це, що і даний алгоритм і деяка кількість ще.

− Ущільнювач сліпої алеї. Цей алгоритм схожий на blind alley filler тим, що він також знаходить всі допустимі рішення, видаляючи тупикові розв'язки з лабіринту. Втім він заповнює лише проходи у всяке тупикове рішення і не стосується комплекту проходів у їхньому кінці. У результаті він зробить у всіх глухих розв'язках і петлях складніше примітивних глухих кутів недоступні проходи. Цей алгоритм віддає пріоритет лабіринту, працює значно швидше, ніж blind alley filler, щоправда, вимагає додаткової пам'яті. Ми визначаємо будь-яку об'єднану секцію стін у неповторне безліч. Щоб зробити це, для будь-яких

сегментів стін, що ще не знаходяться в безлічі, ми виконуємо заливку поверх стін в цій точці, і визначаємо всі стіни, що досягаються, в нове безліч. Після того, як всі стіни будуть у великих числах, ми перевіряємо будь-яку секцію проходів. Якщо стіни з обох боків знаходяться в одному множині, то ми перекриваємо цей прохід. Такий прохід має бути тупиковою розв'язкою, тому що стіни з обох боків об'єднані один одному, утворюючи обгороджений майданчик. Слід зазначити, що подібну техніку можна використовувати допомоги у вирішенні гіперлабіринтів завдяки перекриттю простору між гілками, об'єднаними між собою.

− Пошук найкоротшого шляху. Як можна зрозуміти з назви, даний алгоритм знаходить найкоротше рішення, за наявності декількох рішень вибираючи одне з них. Він безліч разів віддає пріоритет що знаходиться в лабіринті, стрімкий для всіх типів лабіринтів і вимагає досить багато додаткової пам'яті, пропорційної розміру лабіринту. Як і collision solver, він, по суті, заливає лабіринт "водою" так, що все на одній відстані спочатку заливається одночасно, хоча кожна "крапля" або піксель запам'ятовує, яким пікселем вони були заповнені. Як тільки у рішення потрапляє «крапля», ми повертаємося назад від неї на початок, і це буде найкоротшим шляхом. Цей алгоритм якісно працює для будь-яких вхідних даних, оскільки на відміну більшості інших не вимагає наявності в лабіринті проходів у піксель шириною, якими можна пройти. Слід зазначити, що це, по суті, алгоритм пошуку шляху A* без евристики, тобто всьому руху надається ідентична вага.

− Random mouse. Для розмаїття наведу неефективний спосіб вирішення лабіринту, який, власне, полягає у ненавмисному переміщенні, тобто. рух в одному напрямку і проходження цим проходом з усіма поворотами, поки ми не досягнемо подальшої розвилки. Ми не робимо жодних поворотів на сто вісімдесят градусів, якщо без них можна обійтись. Це симулює поведінку людини, яка випадково блукає лабіринтом і не пам'ятає, де вона вже була. Алгоритм неквапливий і гарантує укладання чи рішення лабіринту, а по

досягненні кінця буде так само важко повернутися до початку, але він примітивний і не вимагає додаткової пам'яті для реалізації. Розглянувши відомі алгоритми, такі як пошук найкоротших шляхів, дотримання по стінах, заповнювач глухих кутів, алгоритм Тремо, алгоритм ланцюгів, алгоритм Пледжа заключимо, що дані алгоритми мають як переваги, так і певні недоліки. Насамперед не всі алгоритми дають гарантію на перебування виходу. Також деякі алгоритми створені навмисно під певний вид лабіринту. Також постає питання про швидкість знаходження шляху [9].

1.4 Огляд та аналіз програм аналогів для проходження лабіринту

Проходження лабіринту є досить нетривіальним завданням. На сьогоднішній день для автоматизації таких дій розроблено не багато програмних аналогів. Наприклад, це A-Mazer, MazeT, Maze Navigation, Codebox Maze.

Maze Navigation та Codebox Maze мають змогу генерувати довільний 2D лабіринт, що є плюсом. Проте, ці лабіринти є однотипними.

A-Mazer та MazeT для обробки та знаходження шляху в лабіринті відсилають його на сервер та обробляють його там, що дає змогу зменшити навантаження на девайс на якому працює програмний засіб. Це є як плюсом так і мінусом, тому що для роботи вони потребують постійного підключення до мережі Інтернет.

Всі наведені аналоги використовують генетичний алгоритм для знаходження шляху, але жоден з них не дає змогу налаштовувати вхідні параметри для алгоритму.

Це є великим мінусом, тому що в більших або більш заплутаних, не стандартних, лабіринтах вони будуть знаходити шлях довше [10].

1.5 Висновок до розділу 1

Проведено аналіз предметної області та досліджені основні алгоритми проходження лабіринту. Наведено поняття лабіринту, розглянуто існуючі класи лабіринтів та їх особливості.

Проаналізовані основні алгоритми проходження лабіринту. Були знайдені їх переваги та недоліки, а також визначено, що аналізовані алгоритми використовуються головним чином знаходження шляху в певних видах лабіринту. Крім того, більшість алгоритмів не пристосовані для гарантованого пошуку виходу.

Виконано огляд та аналіз існуючих програмних засобів, для проходження лабіринту за допомогою генетичного алгоритму. В наведених існуючих програмних засобах відсутня можливість налаштування вхідних параметрів для роботи з генетичним алгоритмом, що значно зменшує швидкість розв’язання задачі проходження лабіринту.

Наведено і обґрунтовано вимоги до розроблюваного програмного продукту.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПРОХОДЖЕННЯ ЛАБІРИНТУ

2.1 Обгрунтування вибору алгоритму для проходження лабіринту

Генетичний алгоритм (ГА) виготовляє послідовні покоління індивідів, обчислюючи їхню "пристосованість" на кожному кроці та вибираючи кращі з них, доки не буде досягнуто термінальної умови. Нижче показаний процес звичайного генетичного методу.

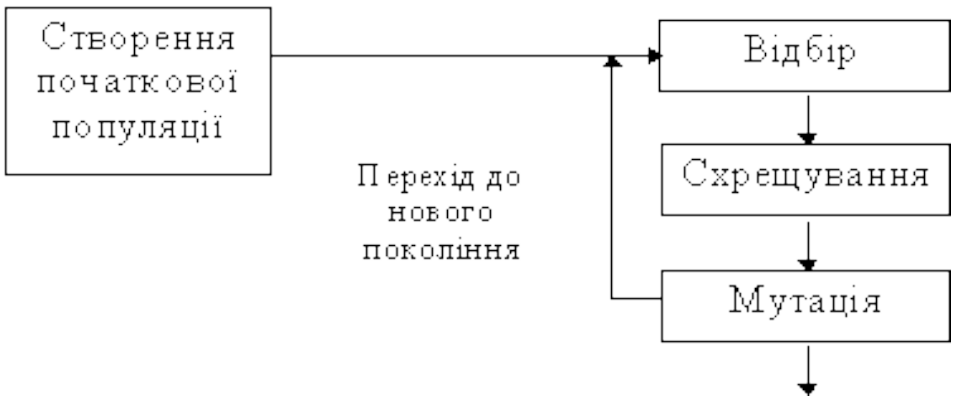


Рисунок 2.1 – Схема основних вузлів роботи генетичного алгоритму

1. Створити вихідну випадкову популяцію.
2. Розрахувати фітнес-функцію (ФФ) індивідів популяції.
3. Повторити наступні кроки, поки не буде досягнуто термінальне умова:
    3.1. Вибрати найбільш пристосованих індивідів з поточної популяції і зробити потомство.
    3.2. Оцінити ФФ кожного нащадка.
    3.3. Замінити найменш пристосованих індивідів з поточної популяції згенерованої нащадками.

У хромосомі застосовуються цілі числа від нуля до трьох включно для подання переміщення з однієї клітини в іншу. Малюнок один показує уявлення

хромосоми та рівнозначний шлях. Для набуття оптимального шляху довжина хромосоми встановлюється рівною подвоєному розміру лабіринту. Надалі планується потроху збільшувати довжину хромосоми з кожним поколінням [11].

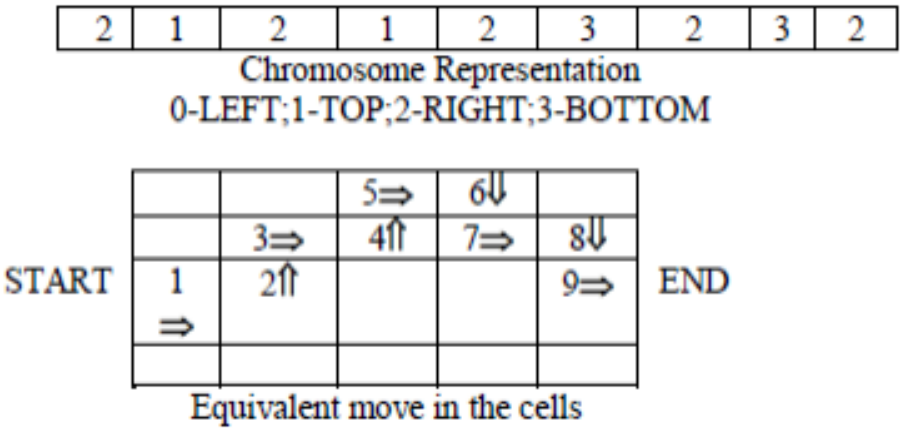


Рисунок 2.2 – Представлення хромосоми

Фітнес-функція даного алгоритму використовує шлях, заданий в хромосомі для досягнення Фінішною клітини.

$$\varphi = \left(\frac{\gamma-\alpha}{\mu-\alpha}\right) \times 100,$$

де: φ – значення пристосованості; γ – поточна клітина; α – стартова клітина; μ – фінішна клітина.

Операція кросовера показана на рисунку 2.

З поточного популяції вибирається двоє батьків та генерується двоє потомків, шляхом виконання операцій додавання і віднімання над відповідними генами батьків. Перший нащадок є результатом операції додавання, а другий – результатом операції віднімання [12].

До всіх генів нащадків застосовується операція "n модуль 4", щоб значення генів залишалися в межах [0; 3].

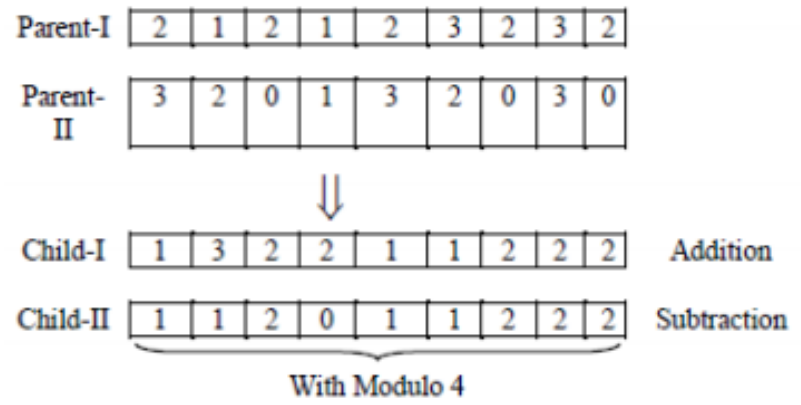


Рисунок 2.3 – Операція кросовера

Приклад випадкової мутації показаний на рисунку 3. Випадково обраний ген хромосоми замінюється на випадкове значення в межах від 0 до 3.

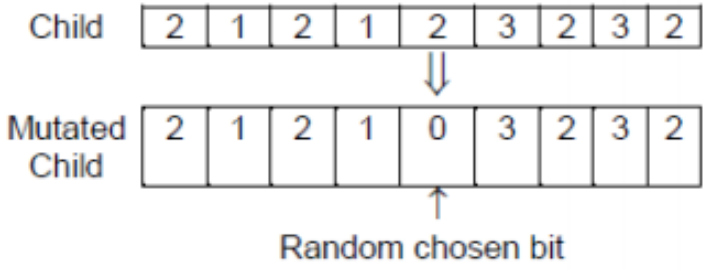


Рисунок 2.4 – Операція мутації

Процес виробництва структури лабіринту складається з трьох кроків. 1-й крок полягає у реалізації випадкової прямокутної сітки розміром nxn.

Далі будь-якій клітині, що досягається зі стартової клітини, і будь-якій клітині, що досягається з фінішної клітини, присвоюється мітка. На третьому етапі проводиться перевірка, чи збігаються мітки клітин. Якщо мітки клітин не збігаються, з'єднання проводиться шляхом видалення стіни.

Використаний метод дозволив отримати найкращі результати з погляду необхідного числа поколінь для отримання раціонального рішення [13].

2.2 Архітектура, математична модель та функціонування інформаційної технології проходження лабіринту

Для розробки цього програмного забезпечення було обрано архітектурний зразок MVC. В рамках архітектурного зразка модель-вид-контролер (MVC) програма ділиться на 3 окремі, але взаємопов'язані частини з поділом функцій між компонентами. Модель відповідає за зберігання даних та їх схему. Вид (View) відповідальний за подання цих даних користувачеві, тобто інтерфейс програми.

Контролер (Controller) управляє компонентами, отримує сигнали з допомогою реакції на функціонування користувача (видовидозміна становища курсора миші, натискання кнопки, введення даних у текстове поле) і передає дані модель. Модель є центральним компонентом зразка MVC і зображати поведінку програми, вільно від інтерфейсу користувача. Модель стосується прямого управління даними, логікою та правилами додатка.

Вид може бути будь-яке уявлення інформації, одержуваної на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька видів (уявлень) однієї й тієї інформації, наприклад гістограма керувати організації та таблиці для бухгалтерії. Контролер отримує вхідні дані і трансформує в команди для моделі чи виду. Модель інкапсулює ядро даних і їх головний функціонал і залежить від процесу введення чи результату даних. Вид може мати кілька взаємопов'язаних областей, наприклад, різні таблиці та поля форм, у яких зображатися дані. У функції контролера входить відстеження певних обставин, що у результаті дій користувача. Контролер дозволяє структурувати код шляхом угруповання пов'язаних дій у відокремлений клас [14].

Наприклад, у нормальному MVC-проекті може бути контролер користувача, що містить групу способів, пов'язаних з керуванням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю і зміна пароля. Зареєстровані події транслюються у різні запити, надіслані

компонентам моделі чи об'єктам, відповідальним за зображення даних. Відділення моделі від виду дозволяє вільно використовувати різні компоненти для зображення інформації. Таким чином, якщо користувач через контролер внесе видозміни в модель даних, то інформація, представлена одним або декількома візуальними компонентами, буде механічно скоригована відповідно до змін, що відбулися.

Розроблено загальну структурну схему роботи системи проходження лабіринту за допомогою генетичного алгоритму, зображену на малюнку два [15].

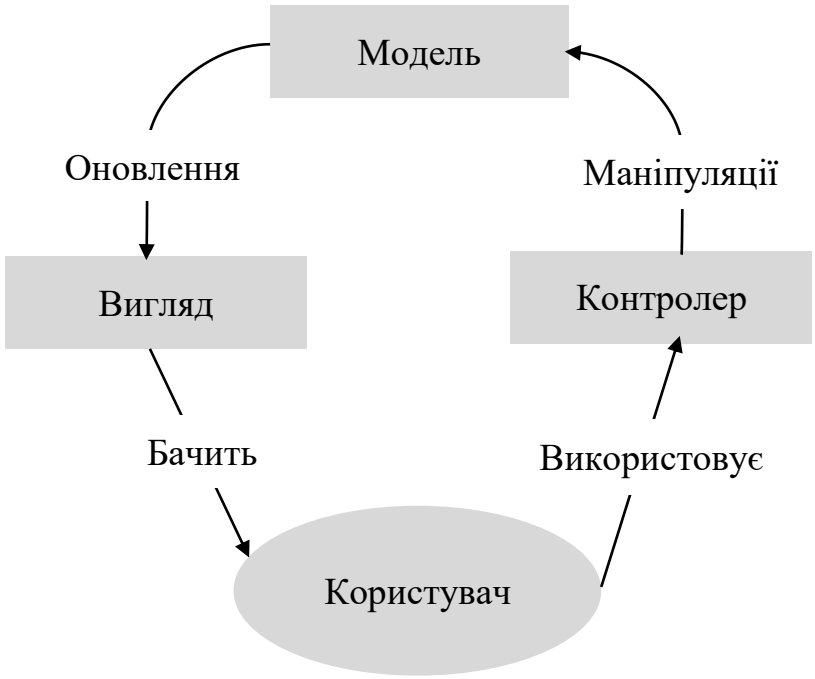


Рисунок 2.5 – Використання архітектурного шаблону MVC

Для правильної роботи застосунку побудуємо алгоритм роботи інформаційної технології для проходження лабіринту на базі генетичного алгоритму, який зображений на рисунку 2.6.

Алгоритм складається з таких кроків:

Блок 1. Створення вікна користувача.

Блок 2. Створення лабіринту.

Блок 3. Перевірка на натиснення кнопки.

Блок 3.1 Якщо нажата кнопка «Налаштування», перейти до Блоку 4.

Блок 3.2 Якщо нажата кнопка «Приклад», перейти до Блоку 5.

Блок 3.3 Якщо нажата кнопка «Нове покоління», перейти до Блоку 6.

Блок 3.4 Якщо нажата кнопка «Наступне покоління», перейти до Блоку 7.

Блок 3.5 Якщо нажата кнопка «Створити 200 поколінь», a = 0, перейти до Блоку 7.1.

Блок 4. Відкриття вікна налаштування алгоритму.

Блок 5. Відтворення заданого просування створіння по лабіринту.

Блок 6. Створення нового покоління в кількості 100 створінь.

Блок 7. Створення ще одного покоління в кількості 100 створінь.

Блок 7.1 Якщо a < 200, a=a+1, перейти до Блоку 7.

Блок 8. Збереження результату проходження створінь по лабіринту в список.

Блок 9. Якщо вибрано створіння зі списку результатів, перейти до Блоку 5.

Блок 10. Якщо створіння знаходиться на клітинці виходу, показати шлях та зупинити програму.

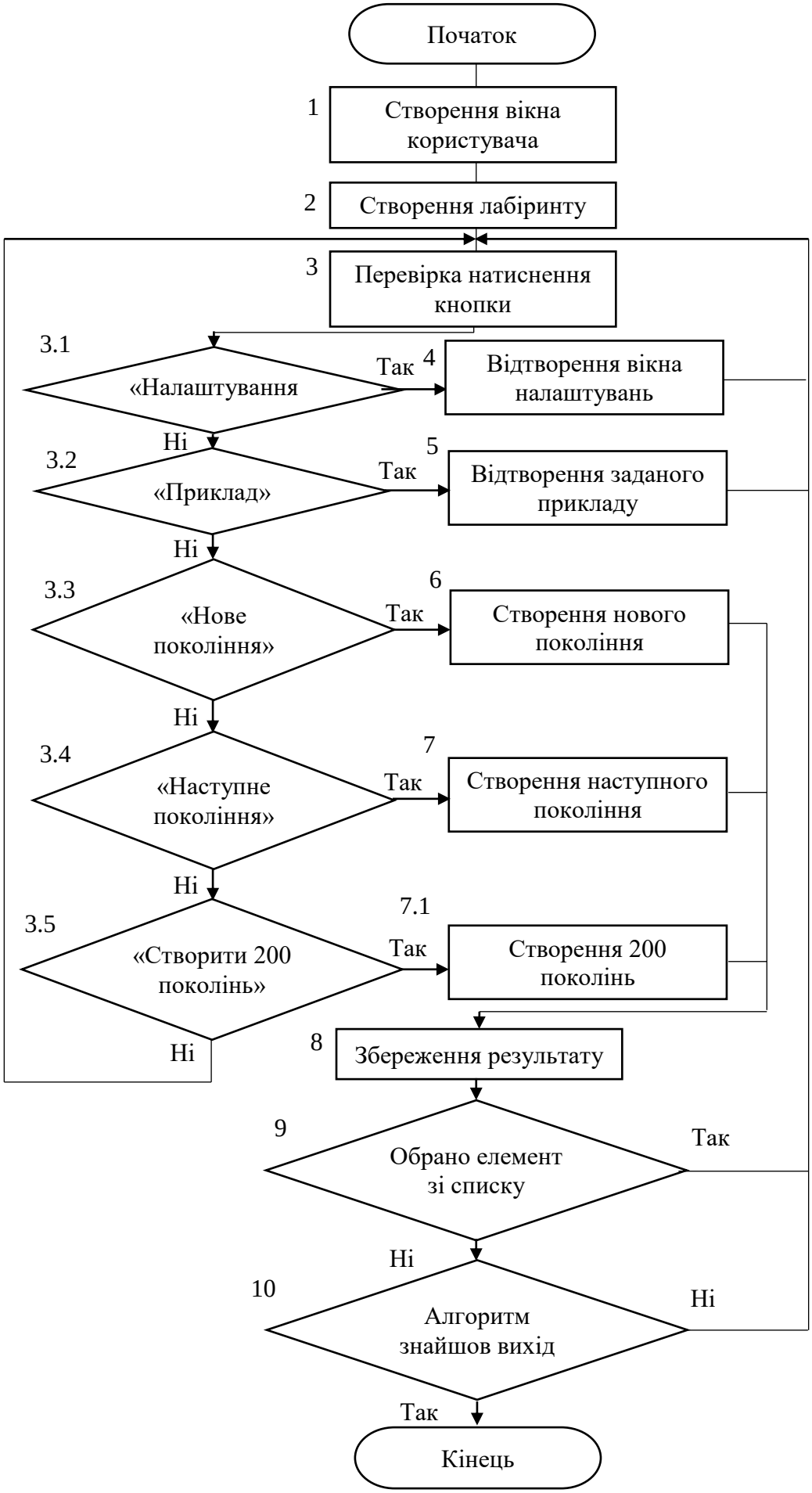


Рисунок 2.6 – Схема загального алгоритму функціонування системи

2.3 Розробка структури інформаційної технології проходження лабіринту на базі генетичного алгоритму

Після створення вікна програми, буде створено лабіринт для проходження. Мінімальна розмірність алгоритму становить 10x10. Сам лабіринт буде виконаний в 2D вимірі. Для вірної роботи інформаційної технології користувачу надана можливість налаштовування вхідних параметрів для роботи генетичного алгоритму. Після введення цих параметрів генетичний алгоритм створить першу початкову популяцію, яка буде рухатись по лабіринту впродовж свого життя.

Далі буде розрахована фітнес-функція індивідів популяції. Далі циклічно буде проводитись відбір найбільш пристосованих індивідів з поточної популяції, які будуть робити потомство. Після, оцінюється фітнес-функція кожного нащадка та виконується заміна найменш пристосованих індивідів з поточної популяці згенерованої нащадками.

На основі отриманих даних від проходження лабіринту популяціями, буде складатись шлях від входу до виходу. Кроки кожної популяції записуються в окремий список для знаходження найкоротшого шляху після знаходження виходу [16].

2.4 Висновок до розділу 2

Обгрунтовано вибір алгоритму для проходження лабіринту. Проаналізовано роботу генетичного алгоритму. Для вірної роботи генетичного алгоритму потрібно пройти такі кроки:

1. Створити вихідну випадкову популяцію.
2. Розрахувати фітнес-функцію (ФФ) індивідів популяції.
3. Повторити наступні кроки, поки не буде досягнуто термінальне умова:

3.1. Вибрати найбільш пристосованих індивідів з поточної популяції і зробити потомство.

3.2. Оцінити ФФ кожного нащадка.

3.3. Замінити найменш пристосованих індивідів з поточної популяції згенерованої нащадками.

Розроблено структуру майбутнього програмного забезпечення проходження лабіринту з урахуванням генетичного алгоритму. Для цього програмного забезпечення використаний архітектурний зразок модель-вид-контролер, у якому програма ділиться на 3 окремих, але взаємозалежних елементів із поділом функцій між компонентами.

Розроблено алгоритм роботи програмного модуля та проаналізовано його функціонування. Також розроблено та описано структуру інформаційної технології проходження лабіринту на базі генетичного алгоритму.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПРОХОДЖЕННЯ ЛАБІРИНТУ

3.1 Обґрунтування вибору мови та середовища програмування для проходження лабіринту

Синтаксис C# близький до C++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML.

Перейнявши багато від своїх попередників – мов C++, Object Pascal, Модула і Smalltalk – C#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова C#, на відміну від C++, не передбачає множинне успадкування класів [17].

C# працює за принципом «будь-яка сутність є об'єкт». Його зараховують до об'єктно-орієнтованим, а точніше об'єктним, мов програмування. «Мова заснована на суворості компонентної архітектури і реалізує передові механізми забезпечення безпеки коду» - так прийнято характеризувати її.

Прихильники C# називають його самим мультипарадигменним, універсальним, просунутим і зручним у використанні мовою програмування. З огляду на той факт, що за ним стоїть платформа Microsoft .NET, число таких прихильників досить велике.

Ще одним нововведенням платформи .NET була технологія активних серверних сторінок ASP.NET (Active Server Page). З її допомогою можна було відносно швидко розробити веб-додатки, які взаємодіють з базами даних.

Мова програмування C# була створена спеціально для ASP.NET. На C # повністю була написана і сама ASP.NET [18].

С # підтримує всі три принципи об'єктно-орієнтованого програмування: інкапсуляцію, успадкування і поліморфізм. Крім того, додано реалізацію автоматичного «прибирання сміття», обробки виключень, динамічне зв'язування.

Таблиця 3.1 – порівнянно мов об’єктно-орієнтованого програмування

Характеристика порівняння	C#	Java	C++
Швидкодія	+	-	-
Ручне управління пам’яттю	+	-	+
Перевантаження функцій	+	+	+
ООП	+	+	+
Шаблони	+	+	+
Умовна компіляція	+	-	+
Визначення макросів процесора	+	-	+
Значення параметрів за замовчуванням	+	-	+
Багатовимірні масиви	+	+	+
Динамічні масиви	+	+/-	+/-
Множинне наслідування	+	-	+
Особисте знання	+	-	+

Завдяки порівнянню було визначено, що мова програмування C# має такі переваги: підтримування всіх принципів об'єктно-орієнтованого програмування, автоматичне «прибирання сміття», обробку виключень,

динамічне зв'язування і є найбільш оптимальною для реалізації курсової роботи.

Для розробки заданого програмного забезпечення розглядалося кілька інтегрованих середовищ розробки, саме: CLion, Microsoft Visual Studio, IntelliJ IDEA, Rider. Через істотні переваги, описані нижче, було вибрано середовище розробки Microsoft Visual Studio. Visual Studio є інтегрованим середовищем розробки програмного забезпечення багатьох мов програмування. Наприклад, C#, Java, JavaScript, Python, розроблена корпорацією Microsoft [19].

Visual Studio є високотехнологічним комплексом тісно інтегрованих інструментів програмування, що включає розумовий редактор початкових текстів з розвиненими засобами автоматизації, сильні інструменти рефакторингу коду, вбудовану підтримку спец спецтехнологій J2EE, механізми інтеграції з середовищем тестування Ant/JUnit і системами управління версіями, неповторним перевіркам коду Code Inspection та інноваційний візуальний конструктор графічних інтерфейсів. Visual Studio в першу чергу вважається середовищем розробки для C #. З цією мовою мова середовище працює найкраще.

Не менш важлива допомога їхньої коробки таких передових технологій, як Groovy, Scala та інших. Вони поєднують різні можливості, як C# спільно з функціоналом Ruby, Smalltalk і т.д. Для розробки мовою C# також застосовують такі середовища як. IntelliJ IDEA та JetBrains Rider. Visual Studio була обрана, тому що вона є особливо еластичною та комфортною для розробки додатків з підтримкою мови програмування C# [20].

Microsoft Visual Studio – серія продуктів Майкрософт, що містять інтегроване середовище розробки програмного забезпечення та низку інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні додатки, так і програми з графічним інтерфейсом, включаючи підтримку технології Windows Forms, а також сайти, вебдодатки, вебслужба як в рідному, так і в керованому кодах для всіх платформ. підтримуються Microsoft Windows,

Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Високопрофесійні розробники програмного забезпечення .NET напевно мають у своєму розпорядженні найсерйозніший у цій сфері продукт виробництва Microsoft, який називається Visual Studio. Даний продукт є найбільш функціонально інтенсивним і особливо пристосованим під застосування на підприємствах IDE-середовище. Така потужність, безперечно, має свою ціну, яка варіюється залежно від версії Visual Studio. Як не складно здогадатися, будь-яка версія поставляється зі своїм унікальним комплектом функціональних перспектив [21].

Visual Studio є абсолютно вбудованою середовищем розробки. Вона спроєктований таким чином, щоб робити процес написання коду, його налагодження та компіляції у складання для постачання фінальним покупцям якомога простіше. Насправді це означає, що Visual Studio є дуже складний додаток з багатодокументним інтерфейсом, у якому можна робити практично все, що стосується розробки коду. Нижче перераховані основні можливості Visual Studio: Текстовий редактор З підтримкою цього редактора можна готувати тексти додатків мовою C# (а ще Visual Basic і C).

Текстовий редактор має досить сильні перспективи. Наприклад, при введенні тексту програми він механічно компонує його на сторінці, створюючи між рядками потрібні відступи, вирівнюючи фігурні дужки блоків коду, що відкривають і закривають, і виділяючи ключові слова кольором. Крім того, у міру введення коду він виконує його перевірку на предмет синтаксичних помилок і підкреслює фрагменти, які викликатимуть помилки при компіляції, що також називається налагодженням на стадії проєктування. У редакторі реалізовано засіб IntelliSense, що забезпечує механічне зображення імен класів, полів або методів на початку їх введення, а також списки параметрів, що підтримують всі доступні перевантажені версії методів на початку введення параметрів для методів.

Візуальний редактор форм.

Даний редактор дозволяє розміщувати бажані елементи управління для інтерфейсу користувача і доступу до даних у плані, а Visual Studio після цього механічно додає у вихідні файли код мовою C#, який потрібен для виробництва екземплярів цих елементів у плані.

(Це можливо тому, що всі елементи керування в .NET є екземпляри певних базових класів.)

Допоміжні вікна.

Ці вікна дозволяють переглядати та змінювати різні аспекти проєктного плану, як класів у вихідному коді, а також характеристики (та їх початкові значення), доступні для класів Windows Forms і Web Forms. До того ж такі вікна можуть використовуватися для вказівки параметрів компіляції, наприклад, які збірка повинен посилатися код.

Можливість компіляції серед розробки.

Замість того щоб виконувати компіляцію проєктного плану, запускаючи компілятор C# з командного рядка, можна віддати перевагу відповідному пункту меню в середовищі розробки. Visual Studio самостійно викликає компілятор і передає йому всі необхідні параметри командного рядка, що вказують на які збірки повинен посилатися код і який вид повинен мати збірка на виході (наприклад, файл або бібліотека *.dll). За бажання Visual Studio може також механічно запускати скомпільований виконуваний файл на виконання, дозволяючи перевірити його роботу [22].

Інтегрований відладчик.

За природою програмування код рідко виконується правильно з першого разу. Visual Studio забезпечує гладке підключення відладчика, дозволяючи створювати зупинкові точки та відстежувати значення змінних, не залишаючи середовище розробки.

Доступ до інших програм. Visual Studio надає доступ до низки інших утиліт, що дозволяють переглядати та змінювати різні аспекти комп'ютера чи

мережі, не залишаючи середовища розробки. Завдяки цим інструментам можна переглядати виконувані служби та активні з'єднання з базами даних, переглядати в таблиці на сервері SQL Server і навіть відвідувати вебсайт із застосуванням вікна Internet Explorer.

Вбудована довідкова система MSDN.

Visual Studio дозволяє отримувати доступ до документації MSDN безпосередньо з середовища IDE. У разі, наприклад, появи коливань з приводу призначення тієї чи іншої ключового слова під час роботи з текстовим редактором можна назвати це ключове слово і натиснути клавішу <F1>, у результаті Visual Studio автоматично під'єднатися до MSDN і зобразити необхідні розділи довідки.

Подібно, якщо необхідно подивитися, що означає той чи інших помилок компіляції, необхідно виділяти повідомлення з помилкою і натиснути <F1>.
Також Visual Studio містить графічні редактори та конструктори XML, забезпечує підтримку розробки програм Windows, націлених на мобільні пристрої, підтримку розробки програм Microsoft Office та Windows Workflow Foundation, містить вбудовану підтримку рефакторингу коду та інструменти конструювання зорових класів [23].

3.2 Вибір спеціалізованої бібліотеки для програмної реалізації проходження лабіринту

Основною та найбільш корисною бібліотекою для програмної реалізації проходження лабіринту на базі генетичного алгоритму буде бібліотека від .NET Framework яка називається Windows Forms.

Windows Forms – це технологія інтерфейсу для .NET, що представляє собою набір керованих бібліотек, які спрощують виконання стандартних завдань, таких як читання з файлової системи і запис в неї. За допомогою середовища розробки, такого як Visual Studio, можна створювати

інтелектуальні клієнтські програми Windows Forms, які відображають інформацію, вимагають введення користувача та взаємодіють з віддаленими комп'ютерами по мережі.

Програма Windows Forms є подієво-орієнтована програма, що підтримується Microsoft .NET Framework. На відміну від пакетних програм, більшість часу витрачається на очікування від користувача будь-яких дій, як наприклад, введення тексту в текстове поле або кліка мишкою по кнопці.

У Windows Forms форма – це візуальна поверхня, на якій відображається інформація для користувача.

Зазвичай програма Windows Forms будується шляхом додавання елементів керування у форми та створення коду для реагування на дії користувача, такі як клацання миші або натискання клавіш. Елемент управління – це окремий елемент інтерфейсу користувача, призначений для відображення або введення даних.

При виконанні користувачем будь-якої дії з формою або одним з елементів управління створюється подія. Програма реагує на ці події, як задано в коді, та обробляє події при їх виникненні.

У Windows Forms передбачено безліч елементів керування, які можна додавати до форм. Наприклад, елементи керування можуть відображати текстові поля, кнопки, списки, перемикачі і навіть веб-сторінки. Якщо передбачені елементи керування не підходять для ваших цілей, у Windows Forms можна створювати власні елементи керування за допомогою класу UserControl.

У Windows Forms є багатофункціональні елементи керування інтерфейсу користувача, що дозволяють емулювати функції таких складних програм, як Microsoft Office. За допомогою елементів керування ToolStrip та MenuStrip ви можете створювати панелі інструментів та меню, які містять текст та зображення, відображають підменю та розміщують інші елементи керування, такі як текстові поля та поля зі списками [24].

Використовуючи функцію перетягування конструктора Windows Forms у Visual Studio, можна легко створювати програми Windows Forms. Просто виділіть елемент керування за допомогою курсору та помістіть його на потрібне місце у формі. Для подолання труднощів, пов'язаних із вирівнюванням елементів керування, конструктор надає такі засоби, як лінії сітки та лінії прив'язки. За допомогою елементів керування FlowLayoutPanel, TableLayoutPanel та SplitContainer можна набагато швидше створювати складні макети форм.

Нарешті, якщо потрібно створити свої власні елементи інтерфейсу користувача, простір імен System.Drawing містить широкий набір класів, необхідних для малювання ліній, кіл та інших фігур безпосередньо на формі.

Отже, дана бібліотека має всі ключові складові для створення програмного забезпечення для проходження лабіринту на базі генетичного алгоритму [25].

3.3 Розробка UML-діаграми класів програми проходження лабіринту на базі генетичного алгоритму

Розроблено класи Dna, Maze, Population, Creature, Sketch, Interface, Form.

Класи Dna, Maze, Population, Creature, Interface, Form слугують для маніпуляції даних, які отримані з БД. Всі ці класи наслідуються від класу Entity, який реалізує маркерний інтерфейс Serializable.

Завдяки цьому класи можна серіалізувати та зберігати поточний стан системи.

Клас Form створює та запускає вікно роботи програми, використовуючи бібіліотеку Windows.Forms.

Клас Interface використовує клас Form для того, щоб створити інтерфейс користування програмою. Саме в цьому класі зазначені поля для введення

вхідних параметрів для вище зазначених класів та кнопки керування програмою.

Клас Settings записує та передає вхідні параметри які ввів користувач для роботи генетичного алгоритму.

Клас Maze використовує задану в програмі інформацію і будує сам лабіринт. За допомогою класу Interface вставляє його у визначене місце у вікні побудованому класом Form.

Клас Dna вказує як саме має створюватись та змінюватись Creature та Population.

Клас Creature будує створіння з певним набором даних з класу Dna.

Клас Population за допомогою класу Creature та інформації з класу Dna, створює популяцію створінь, які повинні рухатись по лабіринту та знайти вихід.

На рисунку 3.1 зображено загальну UML-діаграму класів системи проходження лабіринту на базі генетичного алгоритму.

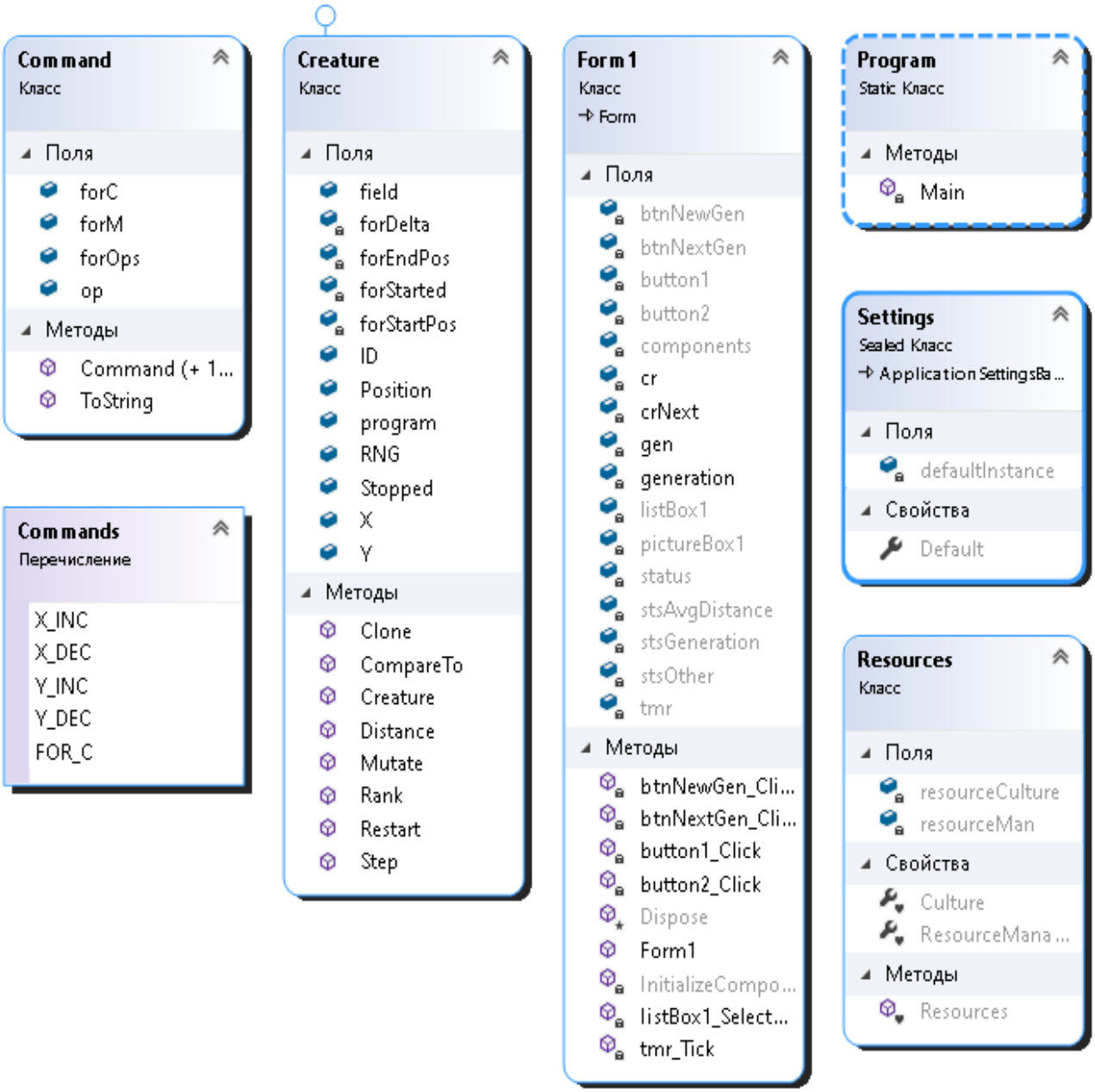


Рисунок 3.1 – Загальна UML-діаграма класів

3.4 Програмна реалізація інформаційної технології проходження лабіринту

Для того щоб правильно показати роботу генетичного алгоритму мені потрібно було звернути увагу на чотири основні елементи додатку. Це створення лабіринту, істоти, популяції та мутації.

Першим основним елементом додатку є лабіринт по якому буде просувати створінь генетичний алгоритм. Тому почнемо зі створення самого лабіринту. Це можна записати у вигляді коду:

```
public Form1()
    {
      InitializeComponent();
      int[,] field = new int[10, 10] {
        {0,0,1,0,1,0,0,1,0,1},
        {0,0,1,0,1,0,1,1,0,1},
        {0,0,0,0,1,0,0,0,0,0},
        {1,1,0,1,1,0,1,1,1,0},
        {1,0,0,0,0,0,0,0,1,0},
        {0,0,1,0,1,0,1,0,1,1},
        {1,1,1,0,1,0,0,0,0,1},
        {0,0,0,0,1,0,0,1,0,0},
        {1,1,0,1,1,0,0,1,1,0},
        {1,0,0,0,0,1,0,0,1,0}
      };
      Creature.field = field;
      Image I = new Bitmap(pictureBox1.Width, pictureBox1.Height);
      Graphics G = Graphics.FromImage(I);
      float w = I.Width / (Creature.field.GetUpperBound(0) + 1);
      float h = I.Height / (Creature.field.GetUpperBound(1) + 1);
      G.Clear(Color.White);
      G.FillRectangle(Brushes.Yellow, 0, 0, w, h);
```

```
            G.FillRectangle(Brushes.LimeGreen, w * field.GetUpperBound(0), h * field.GetUpperBound(0), w, h);
            for (int i = 0; i < field.GetUpperBound(0)+1; i++)
            {
               G.DrawLine(Pens.Black, 0, h * i, I.Width, h * i);
               G.DrawLine(Pens.Black, w * i, 0, w * i, I.Height);
               for (int j = 0; j < field.GetUpperBound(0)+1; j++)
               {
                  if (Creature.field[i, j] == 1) G.FillRectangle(Brushes.Black, i * w, j * h, w, h);
               }
            }
            pictureBox1.Image = I;
         }
```

Після цього я перейшов до розробки методу наповнення лабіринту створіннями. Для того щоб вони з’явились у лабіринті, ми повинні їм вказати де початок лабіринту та звідки потрібно починати свій шлях. Відбувається це за таким методом:

```
public Creature(int id)
      {
         ID = id;
         program = new List<Command>();
         int num = RNG.Next(4, 10);
         for (int i = 0; i < num; i++)
         {
            int op = RNG.Next(0, 5);
            if (op == 4)
            {
               int forM = RNG.Next(2, 5);
               int forC = RNG.Next(1, 3);
               Command C = new Command(Commands.FOR_C);
               C.forM = forM;
               C.forOps = forC;
               program.Add(C);
               for (int j = 0; j < forC; j++)
               {
                  op = RNG.Next(0, 4);
                  program.Add(new Command(op));
               }
               continue;
            }
```

```
            program.Add(new Command(op));
        }
        X = 0;
        Y = 0;
        Position = 0;
    }
```

Створення популяції розробляється на основі створення одного створіння та його клонів. Для вирішення даної задачі я написав такий метод:

```
private void btnNewGen_Click(object sender, EventArgs e)
    {
        crNext = 0;
        gen = 1;
        stsGeneration.Text = "Generation: 1";
        generation = new Creature[100];
        listBox1.Enabled = true;
        tmr.Stop();
        listBox1.Items.Clear();
        double sum = 0;
        for (int i=0; i<100; i++)
        {
            generation[i] = new Creature(crNext++);
            generation[i].Mutate();
            for (int j = 0; j < generation[i].program.Count; j++) generation[i].Step();
            listBox1.Items.Add("Creation #" + generation[i].ID + " (reached " +
generation[i].Distance() + ")");
            sum += generation[i].Distance();
            generation[i].Restart();
        }
        stsAvgDistance.Text = "Average Distance: " + Math.Round(sum) / 100;
    }
```

Останнім основним елементом роботи додатку є мутація істот. Метод, зображений нижче, проводить мутацію створінь в лабіринті та допомагає створенню наступної популяції.

```
  public void Mutate()
    {
        int numberOfMutations =  RNG.Next(1, 4);
        for (int i = 0; i < numberOfMutations; i++)
        {
            int mutateType = RNG.Next(0, 100);
            if (mutateType < 60)
            {
                int pos = RNG.Next(program.Count);
                int newOp = RNG.Next(5);
                if (newOp == 4)
                {
```

```
                int forM = RNG.Next(2, 5);
                int forC = RNG.Next(1, 3);
                Command C = new Command(Commands.FOR_C);
                C.forM = forM;
                C.forOps = forC;
                program[pos]=(C);
                for (int j = 0; j < forC; j++)
                {
                   newOp = RNG.Next(0, 4);
                   program.Insert(pos,new Command(newOp));
                }
                continue;
             }
             program[pos] = new Command(newOp);
          }
          else if (mutateType < 75)
          {
             int pos = RNG.Next(program.Count);
             int newOp = RNG.Next(4);
             program.Insert(pos, new Command(newOp));
          }
          else
          {
             int pos = RNG.Next(program.Count);
             if (program.Count > 2) program.RemoveAt(pos);
          }
       }
    }
```

Отже, розроблені методи дозволяють нам повноцінно показати роботу генетичного алгоритму під час проходження лабіринту та знаходження найкоротшого шляху в ньому.

3.5 Тестування та аналіз результатів роботи програми проходження лабіринту на базі генетичного алгоритму

Для розробленого додатку для проходження лабіринту на базі генетичного алгоритму було проведено тестування, яке підтвердило коректність його роботи та відповідність вимогам висунутим у постановці задачі.

Для того, щоб протестувати додаток його було запущено 1000 разів. Це дало змогу повністю оцінити його працездатність, виявити недоліки, які необхідно доопрацювати в майбутньому.

Тестування проводилось на персональному комп’ютері, на якому відбувалась розробка додатку. Воно починається з запуску самої програми. Після чого створюється лабіринт для проходження. Потім він наповнюється створіннями які його будуть проходити. Далі, для того, щоб протестувати програму, створювались різні популяції та змінювалось місце входу та виходу з лабіринту. Після того як алгоритм знаходив вихід з лабіринту, перевірялось чи дійсно шлях є найкоротшим.

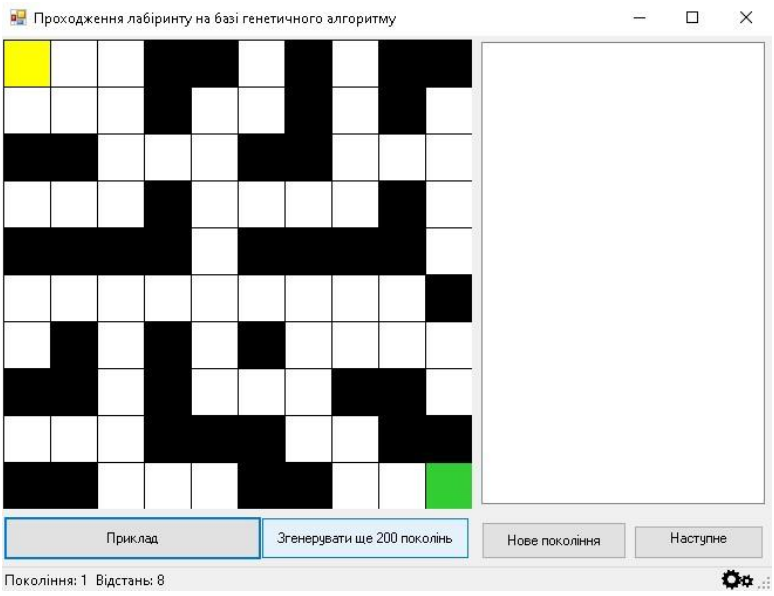


Рисунок 3.2 − Вікно програми

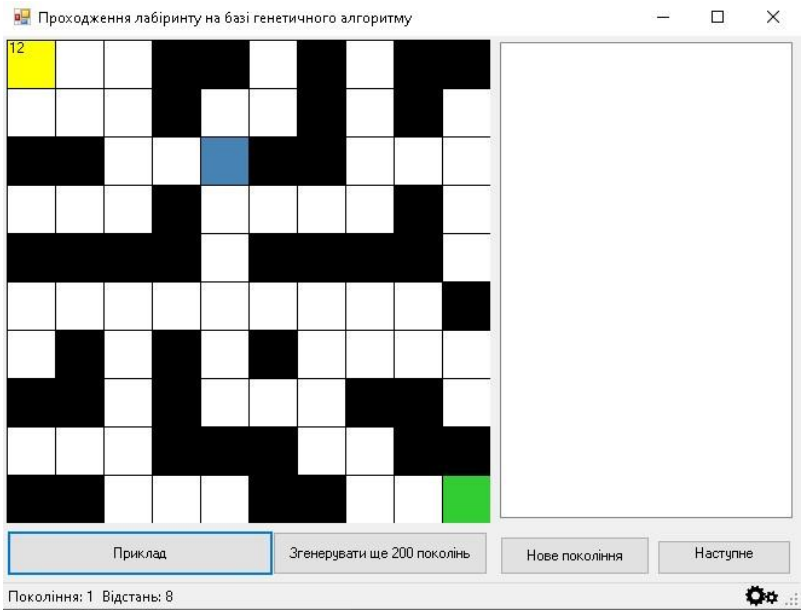


Рисунок 3.3 – Приклад проходження лабіринту

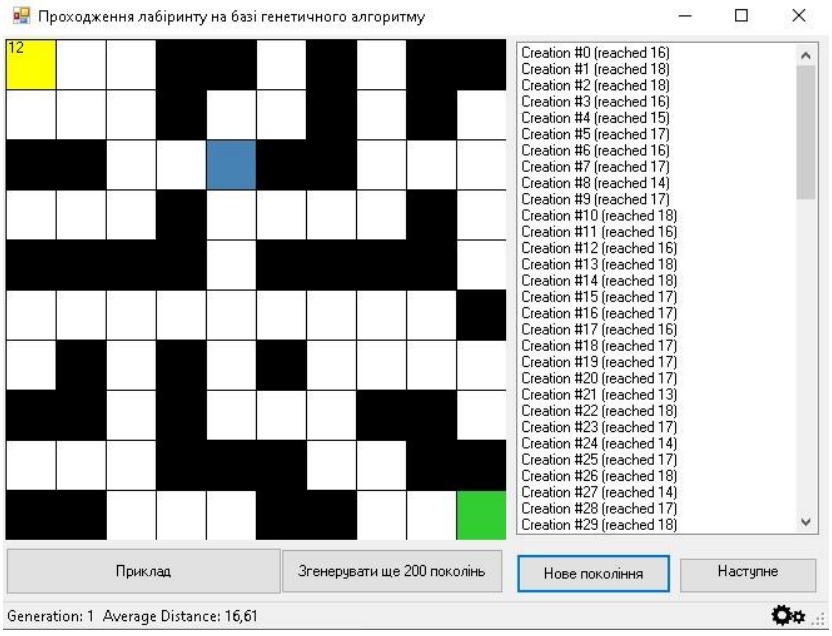


Рисунок 3.4 – Створене нове покоління

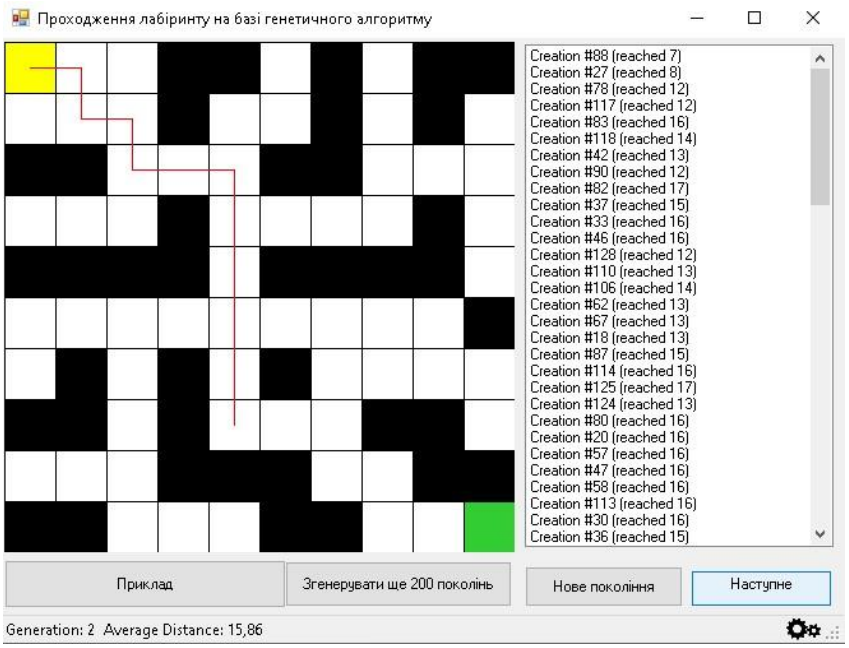


Рисунок 3.5 – Створене наступне покоління

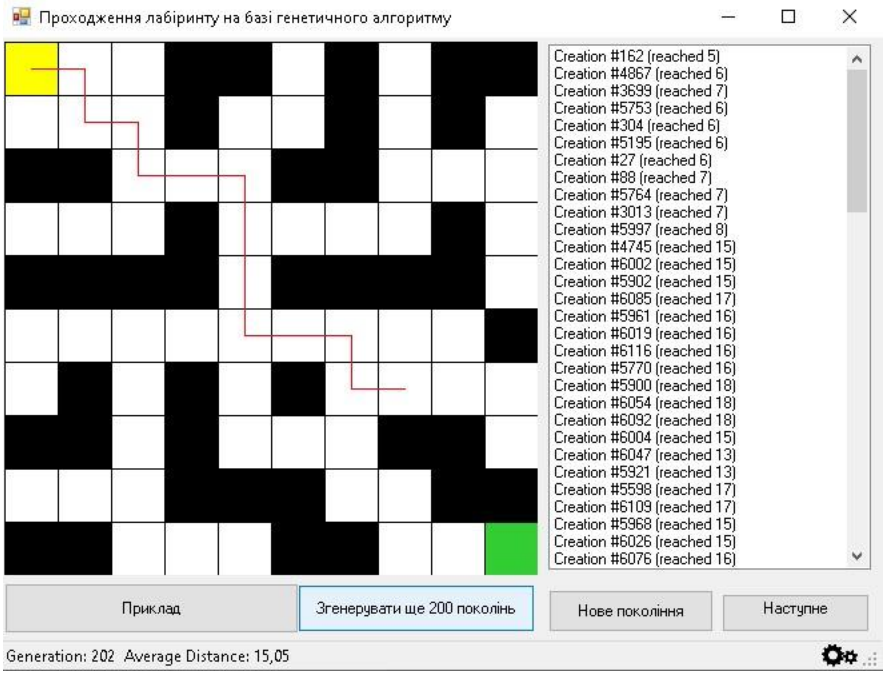


Рисунок 3.6 – Створено ще 200 поколінь

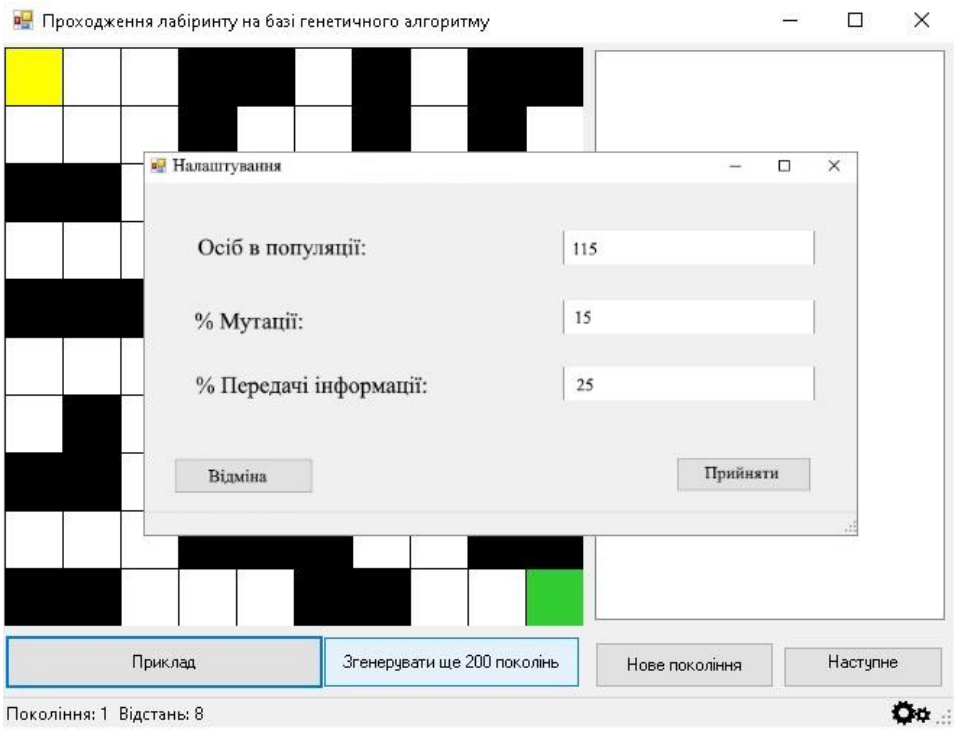


Рисунок 3.7 – Вікно налаштувань

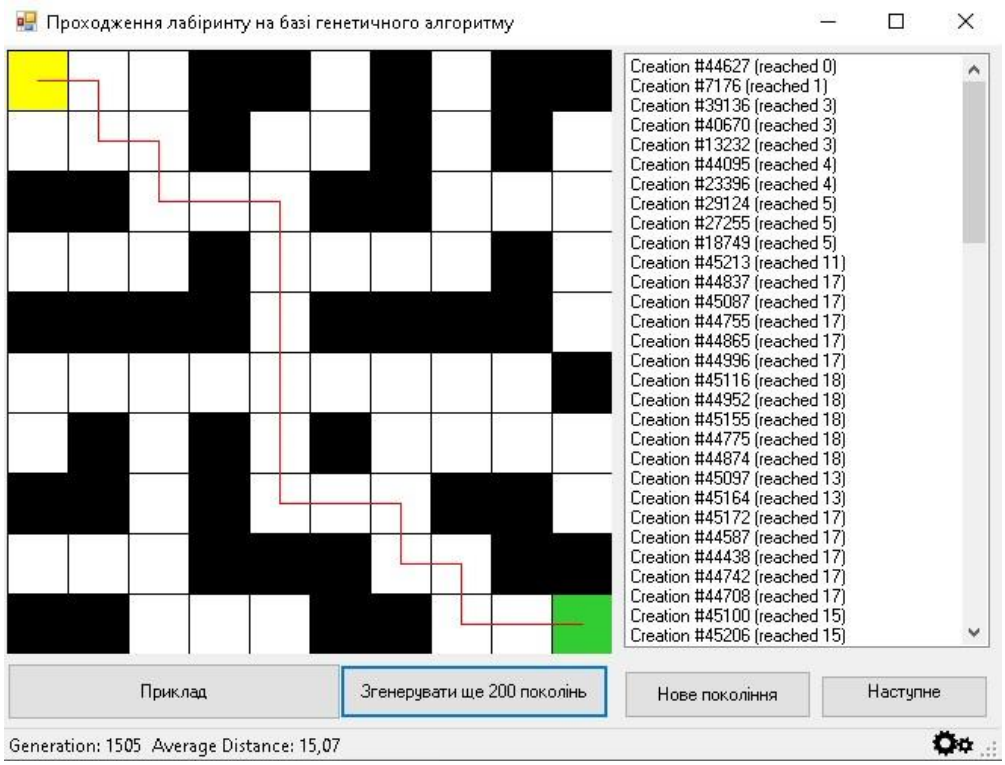


Рисунок 3.8 – Результат проходження лабіринту

Після проведених операцій вирішено, що тестування розробленого додатку пройшло успішно.

У процесі тестування не було виявлено похибок у роботі програмного додатку. Користувачі, які проходили тестування відзначили, що з додатком легко працювати, він інтуїтивно зрозумілий, дизайн стриманий та лаконічний.

Далі у таблиці 3.1 наведено порівняння результатів тестування з алгоритмами-аналогами. Тестування проводилось з застосування різних типів лабіринту. В лабіринтах можлива наявність двох виходів. Стандартна розмірність лабіринту 100x100.

Таблиця 3.2 – Порівняння основних результатів тестування з алгоритмами аналогами

Алгоритм	Рішення	Гарантія	Незалежний від проходів
Пошук найкоротшого шляху	1 найкоротше	Так	Так
Random Mouse	1	Ні	Ні
Wall Follower	1	Ні	Так
Алгоритм Пледжа	1	Ні	Так
Алгоритм ланцюгів	1	Так	Так
Recursive Backtracker	1	Так	Так
Алгоритм Тремо	1	Так	Ні
Генетичний алгоритм	Всі, найкоротші	Так	Так

У таблиці 3.2 наведено порівняння результатів тестування інших відомих програмних засобів для проходження лабіринту на базі генетичного алгоритму відносно розробленого програмного засобу в даній магістерській кваліфікаційній роботі.

Тестування проводилось з використанням приблизно однакових вхідних параметрів та однакових лабіринтів розмірністю 100x100.

Таблиця 3.3 – Порівняння основних результатів тестування з програмами аналогами

Назва	Осіб в популяції	% мутації	% передачі і-ції	Кількість проходжень	Швидкість
A-Mazer	110	15%	25%	130	1 хв., 36 с.
MazeT	110	13%	26%	135	1 хв., 43 с.
Maze Navigation	125	16%	24%	133	1 хв., 40 с.
Codebox Maze	95	15%	25%	137	1 хв., 45 с.
Розроблений ПЗ	105	15%	25%	132	1 хв., 38 с.

Отже, виходячи з результатів які наведені вище в таблиці 3.2, можемо підсумувати, що якщо вхідні параметри генетичного алгоритму відрізняються мінімально, то результат буде майже однаковим. Тому, для покращення результатів в розроблений програмний засіб було додано функціонально можливість гнучкого налаштування вхідних параметрів для роботи генетичного алгоритму.

У таблиці 3.3 наведено порівняння результатів тестування зі зміненими вхідними параметрами для роботи генетичного алгоритму, проте лабіринт залишається однаковим для всіх програм.

Таблиця 3.4 – Порівняння основних результатів тестування з програмами аналогами зі зміненими вхідними параметрами

Назва	Осіб в популяції	% мутації	% передачі і-ції	Кількість проходжень	Швидкість
A-Mazer	110	15%	25%	130	1 хв., 36 с.
MazeT	110	13%	26%	135	1 хв., 43 с.
Maze Navigation	125	16%	24%	133	1 хв., 40 с.
Codebox Maze	95	15%	25%	137	1 хв., 45 с.
Розроблений ПЗ	120	18%	24%	115	1 хв., 5 с.

Після збільшення осіб в популяції, % мутації та зменшення % передачі інформації, ми можемо спостерігати, що кількість проходжень по лабіринту зменшилась, в результаті чого й зменшилась швидкість знаходження шляху.

Отже, провівши тестування розробленого додатку, можна зробити висновок, що він працює швидко, помилок не виявлено; порівняно з програмами-аналогами покращено функціонал, в результаті чого показано кращі результати за характеристиками.

Користувачами було відзначено, що програма легка та інтуїтивно зрозуміла у користуванні. За обсягом ОП розроблений додаток використовує близько 40 до 60 Мб при створенні лабіринту розмірністю від 10x10 до 100x100.

3.6 Висновок до розділу 3

Обґрунтовано вибір програмних засобів реалізації додатку для проходження лабіринту на базі генетичного алгоритму. В результаті для реалізації додатку обрано середовище програмування Visual Studio та мову програмування C#.

Розроблено програмні рішення які передбачають виконання таких додаткових функцій як: побудова лабіринтів з різними особливостями та властивостями, пошуку усіх шляхів проходження лабіринту та найкоротшого з них, а також просування популяції по лабіринту. Наведено окремі частини коду з відповідними коментарями.

Тестування розробленого продукту пройшло успішно та підтвердило, що розроблений додаток відповідає поставленим вимогам. Здійснено порівняння роботи додатку з його аналогами за його основними можливостями та наведено його додаткові функції. Додатки порівнювались за такими критеріями: кількість рішень, гарантія знаходження шляху, можливість пройти лабіринт. Додаток може працювати в двох режимах: 1) враховувати довільне розташування створіння в межах лабіринту; 2) враховувати розташування входу та виходу в лабіринті.

Покращено функціональні можливості відомих програмних засобів. У розробленому продукті користувач має можливість налаштовувати вхідні параметри для роботи генетичного алгоритму. Таким чином гнучко підлаштовувати алгоритм для роботи з різними видами лабіринтів. Що дає змогу швидше розв’язувати задачу проходження лабіринту.

У порівнянні з іншими алгоритмами, генетичний алгоритм має перевагу у кількості рішень, гарантії знаходження виходу та незалежності від. Доцільно зазначити, що за обсягом пам’яті розроблений додаток використовує від 40 до 60 Мб при розмірності лабіринту від 10x10 до 100x100.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Комерційний та технологічний аудит науково-технічної розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено двох незалежних експертів. Такими експертами будуть Арсенюк І.Р. та Маслій Р. В..

Оцінювання комерційного потенціалу розробки буде здійснено за 12-ма критеріями за 5-ти бальною шкалою. Результати оцінювання комерційного потенціалу розробки наведено в таблиці 4.1.

Таблиця 4.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	Арсенюк І.Р.	Маслій Р. В.
	Бали, виставлені експертами:	
1	4	4
2	3	3
3	3	4
4	4	3
5	3	4
6	4	
7	3	3
8	4	4
9	4	3
10	4	3
11	3	4
12	3	4
Сума балів	$СБ_1 = 43$	$СБ_2 = 43$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 43$	

Отже, з отриманих даних таблиці 4.1 видно, що нова розробка має високий рівень комерційного потенціалу.

4.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (4.1):

$$З_о = \frac{M}{T_p} \cdot t, \quad (4.1)$$

де M – місячний посадовий оклад конкретного розробника;

T_p – кількість робочих днів у місяці, T_p = 22 дні;

t – число днів роботи розробника, t = 50 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 4.2.

Таблиця 4.2 – Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	10000	454,55	5	2272,75
Інженер-програміст	6000	272,73	50	13636,36
Всього:				15909,11

Розрахуємо додаткову заробітну плату:

$$З_{дод} = 0,1 \cdot 15909.11 = 1590,91 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 22% від суми їхньої основної та додаткової заробітної плати:

$$Н_{зп} = (З_о + З_р) \cdot \frac{\beta}{100}, \quad (4.2)$$

$$Н_{зп} = (15909,11+1590,91) \cdot \frac{22}{100} = 3850.00 (\text{грн.}).$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12}, \quad (4.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 4.3 – Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань,грн
Персональний комп'ютер	10000	25	3	625
Всього:				625

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n Н_і \cdot Ц_і \cdot К_і, \quad (4.4)$$

де *n* – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 4.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	180	1	180
Пачка паперу	уп.	130	1	130
Ручка	шт.	10	1	10
Всього з урахуванням транспортних витрат				352

Витрати на силову електроенергію розраховуються за формулою:

$$B_e = B \cdot П \cdot Ф \cdot K_n , \quad (4.5)$$

де *В* – вартість 1кВт-години електроенергії (В=1,68 грн/кВт);

П – установлена потужність комп’ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп’ютера (Ф=200 год.);

$К_п$ – коефіцієнт використання потужності ($К_п < 1$, $К_п = 0,7$).

$$B_e = 1,68 \cdot 0,6 \cdot 200 \cdot 0,7 = 141,12 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (1..3) \cdot (З_о + З_р). \quad (4.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1 * (15909,11+1590,91) = 17500,02 \text{ (грн).}$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

$$В = 15909,11+1590,91+3850,51+625+352+141,12+17500,02 = 39968,67 \text{ (грн.)}$$

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha}, \quad (4.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{39968,67}{1} = 39968,67.$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta}, \quad (4.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{39968,67}{0,9} = 44409.63 \text{ (грн.)}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$вкладених інвестицій розраховується так:

$$Е_{абс} = (ПП - PV), \quad (4.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до т. 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, зображений на рисунку 4.1.

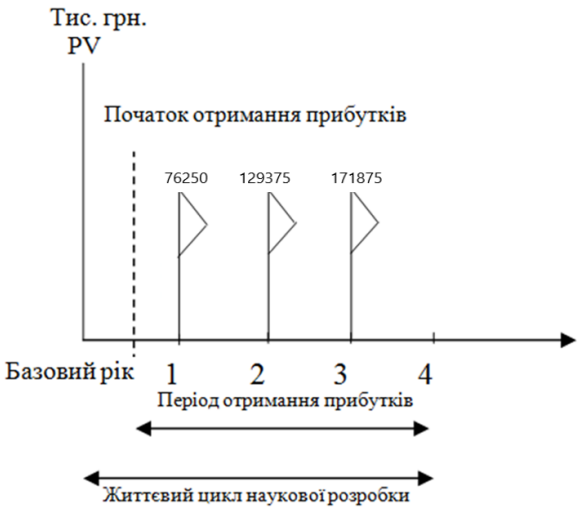


Рисунок 4.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (4.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

τ – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{44409,63}{(1+0,1)^0} + \frac{76250}{(1+0,1)^2} + \frac{129375}{(1+0,1)^3} + \frac{171875}{(1+0,1)^4} = 322020,45(\text{грн.})$$

Тоді розрахуємо $E_{абс}$:

$$Е_{абс} = 322020,45 - 44409,63 = 277610,82 \text{ грн.}$$

Оскільки $E_{\text{абс}} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $\text{Е}_\text{в}$ за формулою:

$$\text{Е}_\text{в} = \sqrt[\text{Т}]{1 + \frac{\text{Е}_\text{абс}}{\text{PV}}} - 1, \qquad (4.12)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$T_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$\text{Е}_\text{в} = \sqrt[3]{1 + \frac{277610{,}82}{44409{,}63}} - 1 = 1{,}69 \text{ або } 169\ \%.$$

Далі, розраховану величина $\text{Е}_\text{в}$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{\text{мін}}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{\text{мін}}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0{,}2 + 0{,}1 = 0{,}3$$

Оскільки $E_в$ = 169% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в},$$

$$T_{ок} = \frac{1}{1,69} = 0,69 \text{ року}.$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

4.4 Висновок до розділу 4

У розділі було виконано розрахунок витрат на розробку та виготовлення нового технічного рішення, сума яких складає 44409.63 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розаховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, знайдено термін окупності витрат для виробника та економічний ефект для споживача при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розробка у виробництві та використання дешевша за аналог і є більш конкурентоспроможною. Період окупності складе близько 0,69 роки.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи розроблено інформаційну технологію проходження лабіринту на базі генетичного алгоритму та розширено функціонал відомих програмних аналогів.

У першому розділі магістерської роботи було проведено аналіз предметної області та досліджені основні алгоритми проходження лабіринту. Наведено поняття лабіринту, розглянуто існуючі класи лабіринтів та їх особливості. Проаналізовані основні алгоритми проходження лабіринту. Були знайдені їх переваги та недоліки, а також визначено, що аналізовані алгоритми використовуються головним чином знаходження шляху в певних видах лабіринту. Крім того, більшість алгоритмів не пристосовані для гарантованого пошуку виходу. Виконано огляд та аналіз існуючих програмних засобів, для проходження лабіринту за допомогою генетичного алгоритму. В наведених існуючих програмних засобах відсутня можливість налаштування вхідних параметрів для роботи з генетичним алгоритмом, що значно зменшує швидкість розв’язання задачі проходження лабіринту. Наведено і обґрунтовано вимоги до розроблюваного програмного продукту.

У другому розділі магістерської роботи було обгрунтовано вибір алгоритму для проходження лабіринту. Проаналізовано роботу генетичного алгоритму. Для вірної роботи генетичного алгоритму потрібно пройти такі кроки:

1. Створити вихідну випадкову популяцію.
2. Розрахувати фітнес-функцію (ФФ) індивідів популяції.
3. Повторити наступні кроки, поки не буде досягнуто термінальне умова:
    3.1. Вибрати найбільш пристосованих індивідів з поточної популяції і зробити потомство.
    3.2. Оцінити ФФ кожного нащадка.

3.3. Замінити найменш пристосованих індивідів з поточної популяції згенерованої нащадками.

Розроблено структуру майбутнього програмного забезпечення проходження лабіринту з урахуванням генетичного алгоритму. Для цього програмного забезпечення використаний архітектурний зразок модель-вид-контролер, у якому програма ділиться на 3 окремих, але взаємозалежних елементів із поділом функцій між компонентами. Розроблено алгоритм роботи програмного модуля та проаналізовано його функціонування. Також розроблено та описано структуру інформаційної технології проходження лабіринту на базі генетичного алгоритму.

У третьому розділі магістерської роботи було Обґрунтовано вибір програмних засобів реалізації додатку для проходження лабіринту на базі генетичного алгоритму. В результаті для реалізації додатку обрано середовище програмування Visual Studio та мову програмування C#.

Розроблено програмні рішення які передбачають виконання таких додаткових функцій як: побудова лабіринтів з різними особливостями та властивостями, пошуку усіх шляхів проходження лабіринту та найкоротшого з них, а також просування популяції по лабіринту. Наведено окремі частини коду з відповідними коментарями.

Тестування розробленого продукту пройшло успішно та підтвердило, що розроблений додаток відповідає поставленим вимогам. Здійснено порівняння роботи додатку з його аналогами за його основними можливостями та наведено його додаткові функції. Додатки порівнювались за такими критеріями: кількість рішень, гарантія знаходження шляху, можливість пройти лабіринт. Додаток може працювати в двох режимах: 1) враховувати довільне розташування створіння в межах лабіринту; 2) враховувати розташування входу та виходу в лабіринті. Покращено функціональні можливості відомих програмних засобів. У розробленому продукті користувач має можливість налаштовувати вхідні параметри для роботи генетичного алгоритму. Таким чином гнучко

підлаштовувати алгоритм для роботи з різними видами лабіринтів. Що дає змогу швидше розв’язувати задачу проходження лабіринту. У порівнянні з іншими алгоритмами, генетичний алгоритм має перевагу у кількості рішень, гарантії знаходження виходу та незалежності від. Доцільно зазначити, що за обсягом пам’яті розроблений додаток використовує від 40 до 60 Мб при розмірності лабіринту від 10x10 до 100x100.

У четвертому розділі магістерської роботи було виконано розрахунок витрат на розробку та виготовлення нового технічного рішення, сума яких складає 44409.63 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розаховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, знайдено термін окупності витрат для виробника та економічний ефект для споживача при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розробка у виробництві та використання дешевша за аналог і є більш конкурентоспроможною. Період окупності складе близько 0,69 роки.

Отже, можна констатувати, що програма покращує можливості своїх аналогів. У результаті виконання даної кваліфікаційної роботи поставлені задачі було виконано у повному обсязі. Отже, мета магістерської роботи досягнута.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Канаєв Є. Ю., Арсенюк І. Р. Дослідження оптимальності використання генетичного алгоритму для проходження лабіринту. // Modern science: innovations and prospects. Proceedings of the 16th International scientific and practical conference. SSPG Publish. Stockholm, Sweden. 2022. Pp. 162-167. [Електронний ресурс] – Режим доступу до ресурсу: https://sci-conf.com.ua/xvi-mizhnarodna-naukovo-praktichna-konferentsiya-modern-science-innovations-and-prospects-11-13-12-2022-stokgolm-shvetsiya-arhiv/

2. Свідоцтво про реєстрацію авторського права на твір № 115466. Комп`ютерна програма "Програмний модуль подолання лабіринту на базі генетичного алгоритму" / Арсенюк І. Р., Канаєв Є. Ю. Дата реєстрації 26.10.2022.

3. Канаєв Є. Ю. Обґрунтування доцільності застосування генетичного алгоритму для задачі проходження лабіринтів. / Є. Ю. Канаєв, І. Р. Арсенюк, В. І. Месюра // Тези доповідей L науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії. – Вінниця: ВНТУ, 2021. Електронний ресурс (режим доступу https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/view/12522/10611).

4. Алгоритм розв'язування лабіринтів [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Алгоритм_розв%27язування_лабіринтів#:~:text=Існує%20ряд%20різних%20алгоритмів%20для,)%20та%20алгоритм%20Тремо%20.

5. Германн, Керн. Лабиринт: основные принципы, гипотезы, интерпретации // Керн Г. Лабиринты мира. – СПб.: Азбука-классика, 2007 – с. 7-33/

6. Лабіринт [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Лабіринт.

7. Krasner, G.E. and S.T. Pope, A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80, Journal of Object-Oriented Programming, 1(3), pp. 26-49, August/September 1988, SIGS Publications, New York, NY, USA, 1988.

8. Maze Classification [Електронний ресурс] – 2021. – Режим доступу до ресурсу: https://www.astrolog.org/labyrnth/algrithm.htm.

9. Алгоритм вирішення лабіринту - Maze solving algorithm [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikiqube.net/wiki/Maze_solving_algorithm.

10. Maze Algorithms [Електронний ресурс] – Режим доступу до ресурсу: https://prezi.com/cflqtcfxovpw/maze-algorithms/.

11. Популярно про генетичні алгоритми [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20160612014837/http://www.victoria.lviv.ua/html/oio/html/theme10.htm.

12. P.Radhakrishnan, Dr. V.M.Prasad, Dr. V.M.Prasad «Inventory Optimization in Supply Chain Management using Genetic Algorithm».

13. Yandra, Marimin, Irawadi Jamaran, Eriyatno, Hiroyuki Tamura «An integration of multi-objective genetic algorithm and fuzzy logic for optimization of agroindustrial supply chain design»

14. Алгоритм створення лабіринту [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Алгоритм_створення_лабіринту.

15. GAUL Documentation: Contents [Електронний ресурс]. – Режим доступу: http://gaul.sourceforge.net/documentation.html.

16. Riccardo Poli, William B. Langdon, Nicholas Freitag McPhee, A Field Guide to Genetic Programming Paperback – March 26, 2008.

17. Van Vliet, D.Improved shortest path algorithms for transport networks [Електронний ресурс] режим доступу http://www.sciencedirect.com/science/article/pii/0041164778901028.

18. Малихіна М.П., Частікова В.А., Власов К.А., Дослідження ефективності роботи модифікованого генетичного алгоритму в задачах комбінаторики // Сучасні проблеми науки та освіти. – 2013. – № 3.

19. Субботін С. О., Олійник А. О., Олійник О. О. Неітеративні, еволюційні та мультиагентні методи синтезу нечіткологічних і нейромережних моделей: Монографія / Під заг. ред. С. О. Субботіна. — Запоріжжя: ЗНТУ, 2009. — 375 с.

20. Генетичні алгоритми. Ключові поняття і методи реалізації [Електронний ресурс]. – Режим доступу: http://www.znannya.org/?view=ga_general.

21. ВИКОРИСТАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ ПРИ СТВОРЕННІ АГЕНТНО-ОРІЄНТОВАНИХ СИСТЕМ В ЛОГІСТИЦІ [Електронний ресурс]. – Режим доступу: https://www.problecon.com/export_pdf/problems-of-economy-2012-2_0-pages-62_66.pdf.

22. Visual Studio IDE [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ru/.

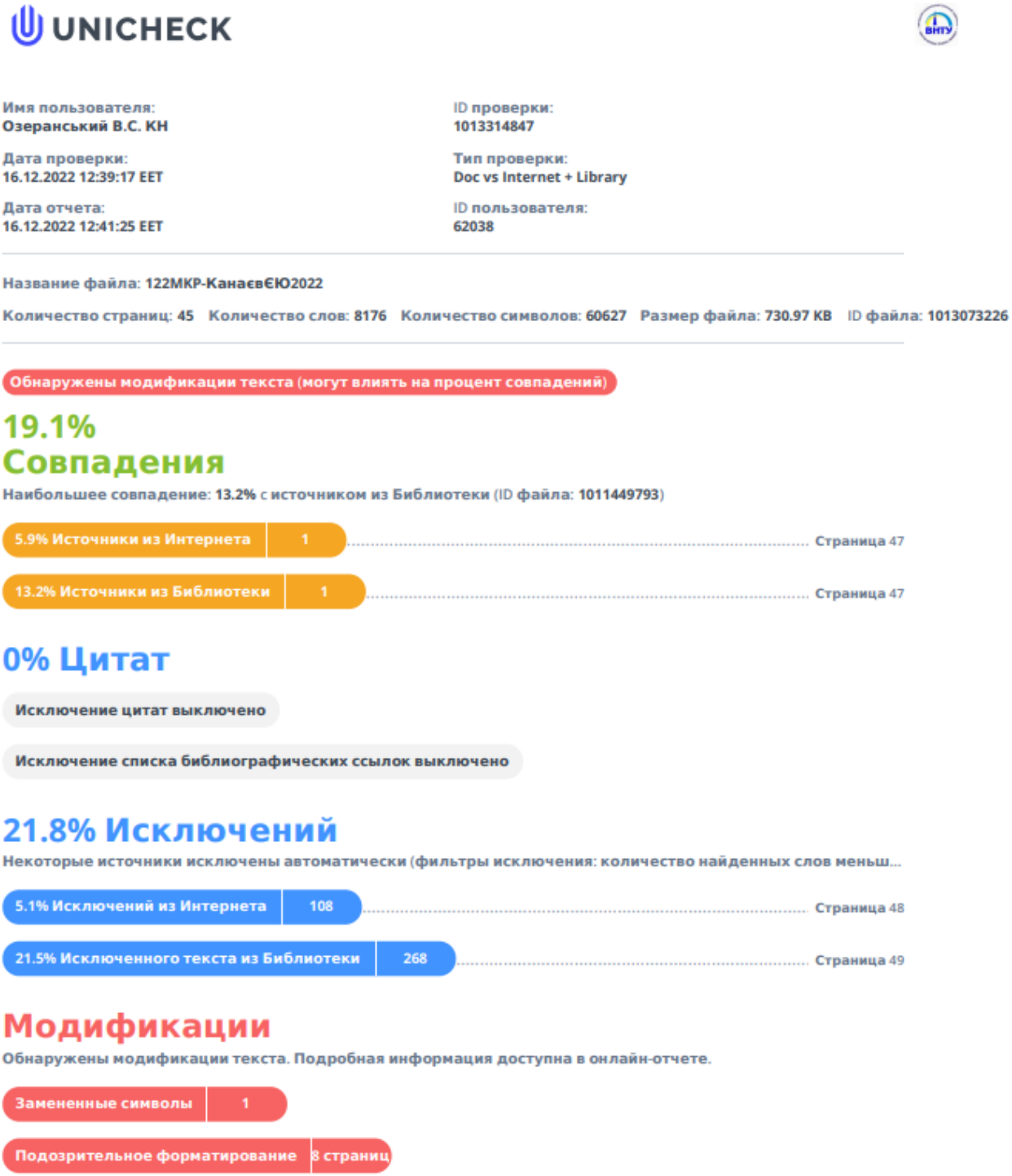
23. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com.

24. Microsoft Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio#:~:text=Microsoft%20Visual%20Studio%20—%20серія%20продуктівта%20низку%20інших%20інструментальних%20засобів.

ДОДАТКИ

Додаток А (обов'язковий)

Результат перевірки на плагіат в онлайн-системі UNICHECK



UNICHECK

Имя пользователя:
Озеранський В.С. КН

ID проверки:
1013314847

Дата проверки:
16.12.2022 12:39:17 EET

Тип проверки:
Doc vs Internet + Library

Дата отчета:
16.12.2022 12:41:25 EET

ID пользователя:
62038

Название файла: 122МКР-КанаєвЄЮ2022

Количество страниц: 45 Количество слов: 8176 Количество символов: 60627 Размер файла: 730.97 KB ID файла: 1013073226

Обнаружены модификации текста (могут влиять на процент совпадений)

19.1%
Совпадения

Наибольшее совпадение: 13.2% с источником из Библиотеки (ID файла: 1011449793)

5.9% Источники из Интернета 1 ..... Страница 47

13.2% Источники из Библиотеки 1 ..... Страница 47

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

21.8% Исключений

Некоторые источники исключены автоматически (фильтры исключения: количество найденных слов меньш...

5.1% Исключений из Интернета 108 ..... Страница 48

21.5% Исключенного текста из Библиотеки 268 ..... Страница 49

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Замененные символы 1

Подозрительное форматирование 8 страниц

Додаток Б (обов'язковий)

Лістинг програми

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GeneticTest
{
   static class Program
   {
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}

namespace GeneticTest
{
   partial class Form1
   {
      private System.ComponentModel.IContainer components = null;

      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }

      #region Код, автоматически созданный конструктором форм Windows

      private void InitializeComponent()
      {
         this.components = new System.ComponentModel.Container();
         this.pictureBox1 = new System.Windows.Forms.PictureBox();
         this.button1 = new System.Windows.Forms.Button();
         this.tmr = new System.Windows.Forms.Timer(this.components);
         this.button2 = new System.Windows.Forms.Button();
         this.listBox1 = new System.Windows.Forms.ListBox();
         this.btnNewGen = new System.Windows.Forms.Button();
         this.btnNextGen = new System.Windows.Forms.Button();
```

```
        this.status = new System.Windows.Forms.StatusStrip();
        this.stsGeneration = new System.Windows.Forms.ToolStripStatusLabel();
        this.stsAvgDistance = new System.Windows.Forms.ToolStripStatusLabel();
        this.stsOther = new System.Windows.Forms.ToolStripStatusLabel();
        ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).BeginInit();
        this.status.SuspendLayout();
        this.SuspendLayout();
        //
        // pictureBox1
        //
        this.pictureBox1.Location = new System.Drawing.Point(3, 3);
        this.pictureBox1.Name = "pictureBox1";
        this.pictureBox1.Size = new System.Drawing.Size(400, 400);
        this.pictureBox1.TabIndex = 0;
        this.pictureBox1.TabStop = false;
        //
        // button1
        //
        this.button1.Location = new System.Drawing.Point(3, 409);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(221, 38);
        this.button1.TabIndex = 1;
        this.button1.Text = "Приклад";
        this.button1.UseVisualStyleBackColor = true;
        this.button1.Click += new System.EventHandler(this.button1_Click);
        //
        // tmr
        //
        this.tmr.Interval = 200;
        this.tmr.Tick += new System.EventHandler(this.tmr_Tick);
        //
        // button2
        //
        this.button2.Location = new System.Drawing.Point(223, 410);
        this.button2.Name = "button2";
        this.button2.Size = new System.Drawing.Size(178, 36);
        this.button2.TabIndex = 2;
        this.button2.Text = "Згенерувати ще 200 поколінь";
        this.button2.UseVisualStyleBackColor = true;
        this.button2.Click += new System.EventHandler(this.button2_Click);
        //
        // listBox1
        //
        this.listBox1.FormattingEnabled = true;
        this.listBox1.Location = new System.Drawing.Point(411, 5);
        this.listBox1.Name = "listBox1";
        this.listBox1.Size = new System.Drawing.Size(242, 394);
        this.listBox1.TabIndex = 3;
        this.listBox1.SelectedIndexChanged += new
System.EventHandler(this.listBox1_SelectedIndexChanged);
        //
```

```
// btnNewGen
//
this.btnNewGen.Location = new System.Drawing.Point(411, 414);
this.btnNewGen.Name = "btnNewGen";
this.btnNewGen.Size = new System.Drawing.Size(124, 32);
this.btnNewGen.TabIndex = 4;
this.btnNewGen.Text = "Нове покоління";
this.btnNewGen.UseVisualStyleBackColor = true;
this.btnNewGen.Click += new System.EventHandler(this.btnNewGen_Click);
//
// btnNextGen
//
this.btnNextGen.Location = new System.Drawing.Point(541, 417);
this.btnNextGen.Name = "btnNextGen";
this.btnNextGen.Size = new System.Drawing.Size(111, 29);
this.btnNextGen.TabIndex = 5;
this.btnNextGen.Text = "Наступне покоління";
this.btnNextGen.UseVisualStyleBackColor = true;
this.btnNextGen.Click += new System.EventHandler(this.btnNextGen_Click);
//
// status
//
this.status.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
this.stsGeneration,
this.stsAvgDistance,
this.stsOther});
this.status.Location = new System.Drawing.Point(0, 452);
this.status.Name = "status";
this.status.Size = new System.Drawing.Size(660, 22);
this.status.TabIndex = 6;
this.status.Text = "statusStrip1";
//
// stsGeneration
//
this.stsGeneration.Name = "stsGeneration";
this.stsGeneration.Size = new System.Drawing.Size(77, 17);
this.stsGeneration.Text = "Покоління: 1";
//
// stsAvgDistance
//
this.stsAvgDistance.Name = "stsAvgDistance";
this.stsAvgDistance.Size = new System.Drawing.Size(110, 17);
this.stsAvgDistance.Text = "Відстань: 8";
//
// stsOther
//
this.stsOther.Name = "stsOther";
this.stsOther.Size = new System.Drawing.Size(0, 17);
//
// Form1
//
```

```
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(660, 474);
            this.Controls.Add(this.status);
            this.Controls.Add(this.btnNextGen);
            this.Controls.Add(this.btnNewGen);
            this.Controls.Add(this.listBox1);
            this.Controls.Add(this.button2);
            this.Controls.Add(this.button1);
            this.Controls.Add(this.pictureBox1);
            this.Name = "Form1";
            this.Text = "Проходження лабіринту на базі генетичного алгоритму";
            ((System.ComponentModel.ISupportInitialize)(this.pictureBox1)).EndInit();
            this.status.ResumeLayout(false);
            this.status.PerformLayout();
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.PictureBox pictureBox1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Timer tmr;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.ListBox listBox1;
        private System.Windows.Forms.Button btnNewGen;
        private System.Windows.Forms.Button btnNextGen;
        private System.Windows.Forms.StatusStrip status;
        private System.Windows.Forms.ToolStripStatusLabel stsGeneration;
        private System.Windows.Forms.ToolStripStatusLabel stsAvgDistance;
        private System.Windows.Forms.ToolStripStatusLabel stsOther;
    }
}


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace GeneticTest
{
    public partial class Form1 : Form
    {
```

```
        public Form1()
        {
            InitializeComponent();
            int[,] field = new int[10, 10] {
                {0,0,1,0,1,0,0,1,0,1},
                {0,0,1,0,1,0,1,1,0,1},
                {0,0,0,0,1,0,0,0,0,0},
                {1,1,0,1,1,0,1,1,1,0},
                {1,0,0,0,0,0,0,0,1,0},
                {0,0,1,0,1,0,1,0,1,1},
                {1,1,1,0,1,0,0,0,0,1},
                {0,0,0,0,1,0,0,1,0,0},
                {1,1,0,1,1,0,0,1,1,0},
                {1,0,0,0,0,1,0,0,1,0}
            };
            Creature.field = field;
            Image I = new Bitmap(pictureBox1.Width, pictureBox1.Height);
            Graphics G = Graphics.FromImage(I);
            float w = I.Width / (Creature.field.GetUpperBound(0) + 1);
            float h = I.Height / (Creature.field.GetUpperBound(1) + 1);
            G.Clear(Color.White);
            G.FillRectangle(Brushes.Yellow, 0, 0, w, h);
            G.FillRectangle(Brushes.LimeGreen, w * field.GetUpperBound(0), h *
field.GetUpperBound(0), w, h);
            for (int i = 0; i < field.GetUpperBound(0)+1; i++)
            {
                G.DrawLine(Pens.Black, 0, h * i, I.Width, h * i);
                G.DrawLine(Pens.Black, w * i, 0, w * i, I.Height);
                for (int j = 0; j < field.GetUpperBound(0)+1; j++)
                {
                    if (Creature.field[i, j] == 1) G.FillRectangle(Brushes.Black, i * w, j * h, w, h);
                }
            }
            pictureBox1.Image = I;
        }
        Creature cr;
        Creature[] generation;
        int crNext = 0;
        int gen;
        private void button1_Click(object sender, EventArgs e)
        {
            cr = new Creature(-1);
            cr.program.Clear();
            Command F = new Command(4);
            F.forM = 2;
            F.forOps = 1;
            cr.program.Add(F);
            cr.program.Add(new Command(Commands.X_INC));
            cr.program.Add(F);
            cr.program.Add(new Command(Commands.Y_INC));
            cr.program.Add(F);
```

```
            cr.program.Add(new Command(Commands.X_INC));
            foreach (Command c in cr.program) { Console.Write(c.ToString() + "; "); }
            Console.WriteLine();
            tmr.Start();
        }

        private void tmr_Tick(object sender, EventArgs e)
        {
            Image I = new Bitmap(pictureBox1.Width, pictureBox1.Height);
            Graphics G = Graphics.FromImage(I);
            float w = I.Width / (Creature.field.GetUpperBound(0) + 1);
            float h = I.Height / (Creature.field.GetUpperBound(1) + 1);
            G.Clear(Color.White);
            G.FillRectangle(Brushes.Yellow, 0, 0, w, h);
            G.FillRectangle(Brushes.LimeGreen, w * Creature.field.GetUpperBound(0), h *
Creature.field.GetUpperBound(1), w, h);
            G.FillRectangle(Brushes.SteelBlue, cr.X * w, cr.Y * h, w, h);
            for (int i = 0; i < Creature.field.GetUpperBound(0)+1; i++)
            {
                G.DrawLine(Pens.Black, 0, h * i, I.Width, h * i);
                G.DrawLine(Pens.Black, w * i, 0, w * i, I.Height);
                for (int j = 0; j < Creature.field.GetUpperBound(1) + 1; j++)
                {
                    if (Creature.field[i, j] == 1) G.FillRectangle(Brushes.Black, i * w, j * h, w, h);
                }
            }
            G.DrawString(cr.Distance() + "", Font, Brushes.Black, 0, 0);
            pictureBox1.Image = I;
            cr.Step();
            if (cr.Stopped) { listBox1.Enabled = true; tmr.Stop(); }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < 200; i++)
            {
                btnNextGen_Click(sender, e);
                Array.Sort(generation);
                if (generation[0].Distance() == 0) break;
            }
        }

        private void btnNewGen_Click(object sender, EventArgs e)
        {
            crNext = 0;
            gen = 1;
            stsGeneration.Text = "Generation: 1";
            generation = new Creature[100];
            listBox1.Enabled = true;
            tmr.Stop();
            listBox1.Items.Clear();
```

```
            double sum = 0;
            for (int i=0; i<100; i++)
            {
                generation[i] = new Creature(crNext++);
                generation[i].Mutate();
                for (int j = 0; j < generation[i].program.Count; j++) generation[i].Step();
                listBox1.Items.Add("Creature #" + generation[i].ID + " (reached " +
generation[i].Distance() + ")");
                sum += generation[i].Distance();
                generation[i].Restart();
            }
            stsAvgDistance.Text = "Average Distance: " + Math.Round(sum) / 100;
        }

        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            listBox1.Enabled = false;
            cr = generation[listBox1.SelectedIndex];
            cr.Restart();
            foreach (Command c in cr.program) { Console.Write(c.ToString() + "; "); }
            Console.WriteLine();
            tmr.Start();
        }

        private void btnNextGen_Click(object sender, EventArgs e)
        {
            gen++;
            stsGeneration.Text = "Generation: " + gen;
            Array.Sort(generation);
            for (int i = 10; i < 100; i++)
            {
                generation[i].Mutate();
            }
            for (int i = 70; i < 100; i++)
            {
                generation[i] = new Creature(crNext++);
                generation[i].Mutate();
            }
                listBox1.Items.Clear();
            Array.Sort(generation);
            double sum = 0;
            foreach (Creature c in generation)
            {
                c.Restart();
                while(!c.Stopped) c.Step();
                listBox1.Items.Add("Creature #" + c.ID + " (reached " + c.Distance() + ")");
                sum += c.Distance();
            }
            Image I = new Bitmap(pictureBox1.Image);
            Graphics G = Graphics.FromImage(I);
            float w = I.Width / (Creature.field.GetUpperBound(0) + 1);
```

```
            float h = I.Height / (Creature.field.GetUpperBound(1) + 1);
            G.Clear(Color.White);
            G.FillRectangle(Brushes.Yellow, 0, 0, w, h);
            G.FillRectangle(Brushes.LimeGreen, w * Creature.field.GetUpperBound(0), h *
Creature.field.GetUpperBound(1), w, h);
            for (int i = 0; i < Creature.field.GetUpperBound(0)+1; i++)
            {
               G.DrawLine(Pens.Black, 0, h * i, I.Width, h * i);
               G.DrawLine(Pens.Black, w * i, 0, w * i, I.Height);
               for (int j = 0; j < Creature.field.GetUpperBound(1)+1; j++)
               {
                  if (Creature.field[i, j] == 1) G.FillRectangle(Brushes.Black, i * w, j * h, w, h);
               }
            }
            Creature C = generation[0];
            int old_X = 0, old_Y = 0;
            C.Restart();
            while (!C.Stopped)
            {
               C.Step();
               G.DrawLine(Pens.Red, w / 2 + w * old_X, h / 2 + h * old_Y, w / 2 + w * C.X, h / 2 + h *
C.Y);
               old_X = C.X;
               old_Y = C.Y;
            }
            pictureBox1.Image = I;
            stsAvgDistance.Text = "Average Distance: " + Math.Round(sum) / 100;
            this.Refresh();
         }
      }
   }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GeneticTest
{
   class Creature: IComparable<Creature>
   {
      public int X;
      public int Y;
      public int ID;
      public int Position;
      public bool Stopped = false;
      public List<Command> program;
      public static int[,] field;
      public static Random RNG = new Random((int)DateTime.Now.Ticks);
```

```
        private int forStartPos;
        private bool forStarted;
        private int forDelta;
        private int forEndPos;
        public Creature(int id)
        {
            ID = id;
            program = new List<Command>();
            int num = RNG.Next(4, 10);
            for (int i = 0; i < num; i++)
            {
                int op = RNG.Next(0, 5);
                if (op == 4)
                {
                    int forM = RNG.Next(2, 5);
                    int forC = RNG.Next(1, 3);
                    Command C = new Command(Commands.FOR_C);
                    C.forM = forM;
                    C.forOps = forC;
                    program.Add(C);
                    for (int j = 0; j < forC; j++)
                    {
                        op = RNG.Next(0, 4);
                        program.Add(new Command(op));
                    }
                    continue;
                }
                program.Add(new Command(op));
            }
            X = 0;
            Y = 0;
            Position = 0;
        }
        public Creature Clone()
        {
            Creature x = new Creature(ID);
            x.program = this.program;
            x.ID = this.ID;
            for (int i = 0; i < x.program.Count; i++) x.Step();
            return x;
        }
        public void Step()
        {
            if (Position >= program.Count() || X==field.GetUpperBound(0) &&
Y==field.GetUpperBound(1)) { Stopped = true; return; }
            switch (program[Position].op)
            {
                case Commands.X_INC:
                    if (X < field.GetUpperBound(0))
                        if (field[X + 1, Y] == 0)
```

```
                    X++;
                break;
            case Commands.X_DEC:
                if (X > 0)
                    if (field[X - 1, Y] == 0)
                        X--;
                break;
            case Commands.Y_INC:
                if (Y < field.GetUpperBound(1))
                    if (field[X, Y + 1] == 0)
                        Y++;
                break;
            case Commands.Y_DEC:
                if (Y > 0)
                    if (field[X, Y - 1] == 0)
                        Y--;
                break;
            case Commands.FOR_C:
                forStarted = true;
                forDelta = program[Position].forM;
                forStartPos = Position;
                forEndPos = Position + program[Position].forOps;
                //Console.WriteLine("FOR from " + forStartPos + " to " + forEndPos + " " + forDelta +
" times");
                break;
        }
        if (forStarted && Position == forEndPos)
        {
            forDelta--;
            if (forDelta == 0) forStarted = false;
            else Position = forStartPos;
        }
        Position++;
    }
    public void Restart()
    {
        X = 0;
        Y = 0;
        Position = 0;
        Stopped = false;
    }
    public void Mutate()
    {
        int numberOfMutations =  RNG.Next(1, 4);
        for (int i = 0; i < numberOfMutations; i++)
        {
            int mutateType = RNG.Next(0, 100);
            if (mutateType < 60)
            {
                int pos = RNG.Next(program.Count);
                int newOp = RNG.Next(5);
```

```
                if (newOp == 4)
                {
                    int forM = RNG.Next(2, 5);
                    int forC = RNG.Next(1, 3);
                    Command C = new Command(Commands.FOR_C);
                    C.forM = forM;
                    C.forOps = forC;
                    program[pos]=(C);
                    for (int j = 0; j < forC; j++)
                    {
                        newOp = RNG.Next(0, 4);
                        program.Insert(pos,new Command(newOp));
                    }
                    continue;
                }
                program[pos] = new Command(newOp);
            }
            else if (mutateType < 75)
            {
                int pos = RNG.Next(program.Count);
                int newOp = RNG.Next(4);
                program.Insert(pos, new Command(newOp));
            }
            else
            {
                int pos = RNG.Next(program.Count);
                if (program.Count > 2) program.RemoveAt(pos);
            }
        }
    }
    public int Distance()
    {
        //return Math.Round(Math.Sqrt(Math.Pow(field.GetUpperBound(0) - X - 1, 2d) +
Math.Pow(field.GetUpperBound(1) - 1 - Y, 2d)) * 100) / 100d;
        return (field.GetUpperBound(0) + field.GetUpperBound(1)-X - Y);
    }
    public int Rank()
    {
        return (int)(Math.Pow((Distance() + 1), 4) * (Position) / 3);
    }

    public int CompareTo(Creature obj)
    {
        Restart(); obj.Restart();
        while(!Stopped) Step();
        while (!obj.Stopped) obj.Step();
        return (int)(this.Rank() - obj.Rank());
    }
  }
  class Command {
    public Commands op;
```

```
        public int forC = 0;
        public int forM;
        public int forOps;
        public Command(Commands cmd)
        {
            op = cmd;
        }
        public Command(int cmd)
        {
            op = (Commands)cmd;
        }
        public new string ToString()
        {
            if (op != Commands.FOR_C)
                return op.ToString();
            else
                return "FOR_"+forOps+"x"+forM;
        }
    }
    enum Commands
    {
        X_INC=0,
        X_DEC=1,
        Y_INC=2,
        Y_DEC=3,
        FOR_C=4
    }
}



namespace GeneticTest.Properties
{


    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.Sett
ingsDesigner.SettingsSingleFileGenerator", "11.0.0.0")]
    internal sealed partial class Settings : global::System.Configuration.ApplicationSettingsBase
    {

        private static Settings defaultInstance =
((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized(new Settings())));

        public static Settings Default
        {
            get
            {
                return defaultInstance;
            }
```

```
        }
    }
}

namespace GeneticTest.Properties
{


[global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyT
ypedResourceBuilder", "4.0.0.0")]
    [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
    internal class Resources
    {

        private static global::System.Resources.ResourceManager resourceMan;

        private static global::System.Globalization.CultureInfo resourceCulture;


[global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance",
"CA1811:AvoidUncalledPrivateCode")]
        internal Resources()
        {
        }


[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.Edi
torBrowsableState.Advanced)]
        internal static global::System.Resources.ResourceManager ResourceManager
        {
            get
            {
                if ((resourceMan == null))
                {
                    global::System.Resources.ResourceManager temp = new
global::System.Resources.ResourceManager("GeneticTest.Properties.Resources",
typeof(Resources).Assembly);
                    resourceMan = temp;
                }
                return resourceMan;
            }
        }


[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.Edi
torBrowsableState.Advanced)]
        internal static global::System.Globalization.CultureInfo Culture
        {
            get
            {
```

```
            return resourceCulture;
        }
        set
        {
            resourceCulture = value;
        }
    }
  }
}
```

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Reflection;
using System.Text;

namespace Evolution
{

    public sealed class Generator<T> where T : class, IFitnessProvider, ICodeProvider<T>
  {

        private readonly Random _simpleRandomGenerator = new Random();

        private Func<double> _randomFunction;

        private List<MethodInfo> _decisions;

        private List<MethodInfo> _terminals;

        public Func<double> RandomFunction
        {
            [Pure] get { return _randomFunction; }
            set
            {
                Contract.Requires(value != null);
                _randomFunction = value;
            }
        }

        public Generator()
        {
            DetermineFunctionsAndTerminals();
        }

        #region Zufallszahlen

        private double GetRandomValue()
```

```
		{
			Contract.Ensures(Contract.Result<double>() >= 0 &&
Contract.Result<double>() <= 1);

			double value = (_randomFunction ??
_simpleRandomGenerator.NextDouble)();
			return Math.Max(0, Math.Min(1, value));
		}

		private int GetRandomValue(int minValue, int maxValue)
		{
			Contract.Requires(minValue <= maxValue);
			Contract.Ensures(Contract.Result<int>() >= minValue &&
Contract.Result<int>() <= maxValue);

			double randomValue = GetRandomValue();
			return (int)Math.Round(randomValue*(maxValue - minValue) + minValue);
		}

		#endregion Zufallszahlen

		#region Ermitteln der Funktionen und Terminals

		internal void DetermineFunctionsAndTerminals()
		{
			Contract.Ensures(_decisions != null);
			Contract.Ensures(_terminals != null);

			Type type = typeof(T);
			if (type.GetCustomAttributes(typeof(EvolutionaryClassAttribute),
true).Length == 0)
				throw new ClassDecorationException("Klasse ist nicht mit dem
EvolutionaryClass-Attribut markiert");

			IEnumerable<MethodInfo> methods =
				from method in type.GetMethods(BindingFlags.FlattenHierarchy |
BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.Instance)
				where method.GetCustomAttributes(typeof
(EvolutionaryMethodAttribute), true).Length > 0
				select method;

			_decisions = new List<MethodInfo>();
			_terminals = new List<MethodInfo>();
			List<string> invalidMethodList = new List<string>();
			foreach (MethodInfo method in methods)
			{
				if (method.GetParameters().Length > 0)
invalidMethodList.Add(method.Name + ": Keine Parameter erlaubt.");
				if (method.ReturnType == typeof(void)) _terminals.Add(method);
				else if (method.ReturnType == typeof(bool))
_decisions.Add(method);
```

```
                    else invalidMethodList.Add(method.Name + ": Nur Rückgabewerte
bool und void erlaubt.");
                }

                if (invalidMethodList.Count > 0)
                {
                    StringBuilder builder = new StringBuilder();
                    builder.AppendLine("Eine oder mehrere Methoden des Objektes
haben eine ungültige Signatur:");
                    for (int i = 0; i < invalidMethodList.Count; ++i )
                    {
                        builder.AppendLine("- " + invalidMethodList[i]);
                    }
                    throw new MethodSignatureException(builder.ToString());
                }
                if (_terminals.Count == 0) throw new GeneratorException("Es wurden keine
Terminalmethoden ermittelt.");
            }

            #endregion Ermitteln der Funktionen und Terminals

            #region Bauen des Ausdrucksbaumes

            internal CodeExpression<T> BuildRandomExpressionTree()
            {
                Contract.Ensures(Contract.Result<CodeExpression<T>>() != null);

                int complexityHelperValue = Math.Min(_terminals.Count,
_decisions.Count);
                int randomComplexity = GetRandomValue(1 + complexityHelperValue, 1 +
complexityHelperValue*4);

                Contract.Assert(randomComplexity >= 1);
                Contract.Assume(_terminals.Count > 0);

                CodeExpression<T> action = BuildExpressionTreeRecursive(null,
randomComplexity, _decisions.Count > 0);
                return action;
            }

            private CodeExpression<T> BuildExpressionTreeRecursive(CodeExpression<T>
parent, int complexity, bool forceDecision)
            {
                Contract.Requires(complexity > 0);
                Contract.Requires(_terminals.Count > 0);
                Contract.Requires(forceDecision && _decisions.Count > 0 ||
!forceDecision);
                Contract.Ensures(Contract.Result<CodeExpression<T>>() != null);

                bool isDecisionNode = forceDecision || (GetRandomValue() > 0.5);
```

```
                    if (complexity == 1 || _decisions.Count == 0 || !isDecisionNode)
                    {
                        MethodInfo method = SelectRandomTerminal();
                        return new CodeExpression<T>(parent, method);
                    }

                    MethodInfo decisionMethod = SelectRandomDecision();
                    Contract.Assume(decisionMethod != null);
                    ConditionalCodeExpression<T> decisionFunc = new
ConditionalCodeExpression<T>(parent, decisionMethod);
                    decisionFunc.LeftAction = BuildExpressionTreeRecursive(decisionFunc,
complexity - 1, false);
                    decisionFunc.RightAction = BuildExpressionTreeRecursive(decisionFunc,
complexity - 1, false);
                    return decisionFunc;
                }

                private MethodInfo SelectRandomTerminal()
                {
                    Contract.Requires(_terminals.Count > 0);
                    Contract.Ensures(_terminals.Count ==
Contract.OldValue(_terminals.Count));
                    Contract.Ensures(Contract.Result<MethodInfo>() != null);

                    int methodIndex = GetRandomValue(0, _terminals.Count - 1);
                    MethodInfo method = _terminals[methodIndex];
                    Contract.Assume(method != null);
                    return method;
                }

                private MethodInfo SelectRandomDecision()
                {
                    Contract.Requires(_decisions.Count > 0);
                    Contract.Ensures(_decisions.Count ==
Contract.OldValue(_decisions.Count));
                    Contract.Ensures(Contract.Result<MethodInfo>() != null);

                    int methodIndex = GetRandomValue(0, _decisions.Count - 1);
                    MethodInfo method = _decisions[methodIndex];
                    Contract.Assume(method != null);
                    return method;
                }

                #endregion Bauen des Ausdrucksbaumes

                #region Genetische Methoden

                internal void Crossover(ref CodeExpression<T> left, ref CodeExpression<T> right)
                {
                    Contract.Requires(left != null && left != null && !ReferenceEquals(left,
right));
```

```
                IList<CodeExpression<T>> leftNodes = left.GetChildNodes().ToList();
                IList<CodeExpression<T>> rightNodes = right.GetChildNodes().ToList();

                int leftIndex = GetRandomValue(0, leftNodes.Count-1);
                int rightIndex = GetRandomValue(0, rightNodes.Count-1);

                CodeExpression<T> leftSubNode = leftNodes[leftIndex];
                CodeExpression<T> rightSubNode = rightNodes[rightIndex];

                ConditionalCodeExpression<T> leftParent = leftSubNode.Parent as
ConditionalCodeExpression<T>;
                if (leftParent != null)
                {
                        if (leftParent.LeftAction == leftSubNode)
                                leftParent.LeftAction = rightSubNode;
                        else
                                leftParent.RightAction = rightSubNode;
                }
                else
                {
                        left = rightSubNode;
                        left.Parent = null;
                }

                ConditionalCodeExpression<T> rightParent = rightSubNode.Parent as
ConditionalCodeExpression<T>;
                if (rightParent != null)
                {
                        if (rightParent.LeftAction == rightSubNode)
                                rightParent.LeftAction = leftSubNode;
                        else
                                rightParent.RightAction = leftSubNode;
                }
                else
                {
                        right = leftSubNode;
                        right.Parent = null;
                }

                rightSubNode.Parent = leftParent;
                leftSubNode.Parent = rightParent;
        }

        internal void Mutate(ref CodeExpression<T> tree)
        {
                IList<CodeExpression<T>> nodes = tree.GetChildNodes().ToList();

                int leftIndex = GetRandomValue(0, nodes.Count-1);

                CodeExpression<T> subnode = nodes[leftIndex];
```

```
                        ConditionalCodeExpression<T> parent = subnode.Parent as
ConditionalCodeExpression<T>;
                        if (parent != null)
                        {
                                int depth = tree.GetDepth();
                                int randomComplexity = GetRandomValue(Math.Max(1, depth/2),
depth*2);

                                if (parent.LeftAction == tree)
                                        parent.LeftAction = BuildExpressionTreeRecursive(parent,
randomComplexity, false);
                                else
                                        parent.RightAction = BuildExpressionTreeRecursive(parent,
randomComplexity, false);
                        }
                }

                #endregion Genetische Methoden

                #region Erzeugen von Generationen

        public delegate T CreateElement(CodeExpression<T> newCode);


                public delegate T CreateElementByMutation(T reference, CodeExpression<T>
newCode);

        public delegate T CreateElementByCrossover(T parentA, T parentB, CodeExpression<T>
newCode);

        public IList<T> CreateGeneration(int generationSize, CreateElement regularCreator)
        {
           Contract.Requires(generationSize >= 0);
           Contract.Requires(regularCreator != null, "Erzeugerfunktion darf nicht null sein.");

           List<T> elements = new List<T>(generationSize);
           for (int i = 0; i < generationSize; ++i)
           {
              CodeExpression<T> expression = BuildRandomExpressionTree();
              elements.Add(regularCreator(expression));
           }
           return elements;
        }

        public GenerationReport<T> EvolveGeneration(int newGenerationSize, IList<T> fitnesses,
CreateElement regularCreator, CreateElementByCrossover crossoverCreator,
CreateElementByMutation mutationCreator)
                {
           Contract.Requires(newGenerationSize > 0, "Größe der Generation muss positiv sein");
           Contract.Requires(fitnesses != null);
```

```
            Contract.Requires(regularCreator != null, "Erzeugerfunktion darf nicht null sein.");
            Contract.Requires(crossoverCreator != null, "Erzeugerfunktion darf nicht null sein.");
            Contract.Requires(mutationCreator != null, "Erzeugerfunktion darf nicht null sein.");

            return EvolveGeneration(newGenerationSize, fitnesses, regularCreator, crossoverCreator,
mutationCreator, 0.1D, 0.1d, 0.05d);
                        }

        public GenerationReport<T> EvolveGeneration(int newGenerationSize, IList<T> fitnesses,
CreateElement regularCreator)
        {
            Contract.Requires(newGenerationSize > 0, "Größe der Generation muss positiv sein");
            Contract.Requires(fitnesses != null);
            Contract.Requires(regularCreator != null, "Erzeugerfunktion darf nicht null sein.");

            return EvolveGeneration(newGenerationSize, fitnesses, regularCreator, (ignored1, ignored2,
code) => regularCreator(code), (ignored, code) => regularCreator(code), 0.1D, 0.1d, 0.05d);
        }

        public GenerationReport<T> EvolveGeneration(int newGenerationSize, IList<T> fitnesses,
CreateElement regularCreator, double keepPercentage, double crossoverPercentage, double
mutationPercentage)
        {
            Contract.Requires(newGenerationSize > 0, "Größe der Generation muss positiv sein");
            Contract.Requires(fitnesses != null);
            Contract.Requires(regularCreator != null, "Erzeugerfunktion darf nicht null sein.");

            return EvolveGeneration(newGenerationSize, fitnesses, regularCreator, (ignored1, ignored2,
code) => regularCreator(code), (ignored, code) => regularCreator(code), keepPercentage,
crossoverPercentage, mutationPercentage);
        }

        public GenerationReport<T> EvolveGeneration(int newGenerationSize, IList<T> fitnesses,
CreateElement regularCreator, CreateElementByCrossover crossoverCreator,
CreateElementByMutation mutationCreator, double keepPercentage, double crossoverPercentage,
double mutationPercentage)
                        {
            Contract.Requires(newGenerationSize > 0, "Größe der Generation muss positiv sein");
            Contract.Requires(regularCreator != null, "Erzeugerfunktion darf nicht null sein.");
            Contract.Requires(crossoverCreator != null, "Erzeugerfunktion darf nicht null sein.");
            Contract.Requires(mutationCreator != null, "Erzeugerfunktion darf nicht null sein.");
                                    Contract.Requires(fitnesses != null);
                                    Contract.Requires(keepPercentage > 0 && keepPercentage <= 1);
                                    Contract.Requires(crossoverPercentage >= 0 && crossoverPercentage <= 1);
                                    Contract.Requires(mutationPercentage >= 0 && mutationPercentage <= 1);

            List<T> fitnessSorted = new List<T>(fitnesses.OrderBy(element => -1*
element.GetFitness()).ThenBy(element => element.GetCode().GetDepth()));

            Func<int, int, double, double> selectionProbability = (index, maxCount, percentage) =>
Math.Exp(-((1.0D / percentage) * index) / maxCount); // TODO: Sigmoid-Funktion verwenden
```

```
List<T> selectedElements = new List<T>();
List<T> deceasedElements = new List<T>();
for (int i = 0; i < fitnessSorted.Count; ++i)
{
   double value = GetRandomValue();
   double judge = selectionProbability(i, fitnessSorted.Count, keepPercentage);
   if (value > judge || fitnessSorted[i].IsStarved)
   {
      deceasedElements.Add(fitnessSorted[i]);
      continue;
   }
   selectedElements.Add(fitnessSorted[i]);
}
int selectedItemCount = selectedElements.Count;

            List<Tuple<T, T, T>> crossoverResults = new List<Tuple<T, T, T>>();
            Func<int, int, double, double> crossoverProbability = selectionProbability;
for (int i = 0; i < selectedItemCount; ++i)
                  {
                        if (GetRandomValue() > crossoverPercentage) continue;

   List<T> crossoverCandidates = new List<T>();
   for (int c = 0; c < selectedItemCount; ++c)
   {
      double value = GetRandomValue();
      double judge = crossoverProbability(i, fitnessSorted.Count, keepPercentage * 0.5);
      if (value > judge) continue;
      crossoverCandidates.Add(selectedElements[c]);
   }

                        int selectedCandidateIndex = GetRandomValue(0,
crossoverCandidates.Count - 1);
                        T elementA = selectedElements[i];
   T elementB = crossoverCandidates[selectedCandidateIndex];

                        CodeExpression<T> codeA = elementA.GetCode().Clone();
                        CodeExpression<T> codeB = elementB.GetCode().Clone();

                              Crossover(ref codeA, ref codeB);

   T newElementA = crossoverCreator(elementA, elementB, codeA);
   T newElementB = crossoverCreator(elementA, elementB, codeB);
   crossoverResults.Add(new Tuple<T, T, T>(elementA, elementB, newElementA));
   crossoverResults.Add(new Tuple<T, T, T>(elementA, elementB, newElementB));
                  }

List<Tuple<T, T>> mutationResults = new List<Tuple<T, T>>();
for (int i = 0; i < selectedItemCount; ++i)
                  {
                        if (GetRandomValue() > mutationPercentage) continue;
   T element = selectedElements[i];
```

```
                                CodeExpression<T> a = element.GetCode().Clone();

                                Mutate(ref a);

            T newElement = mutationCreator(element, a);
            mutationResults.Add(new Tuple<T, T>(element, newElement));
                        }

        List<T> newGeneration = new List<T>();
        newGeneration.AddRange(selectedElements);

        List<T> newElements = new List<T>();
        newElements.AddRange(crossoverResults.Select(element => element.Item3));
        newElements.AddRange(mutationResults.Select(element => element.Item2));
        newGeneration.AddRange(newElements.OrderBy(r => GetRandomValue()));

        IList<T> newlyGenerated;
        if (newGeneration.Count < newGenerationSize)
        {
            int count = newGenerationSize - newGeneration.Count;
                                newlyGenerated = CreateGeneration(count, regularCreator);
                                newGeneration.AddRange(newlyGenerated);
        }
        else
        {
                                newlyGenerated = new List<T>(0);
            while (newGeneration.Count > newGenerationSize)
            {
                newGeneration.RemoveAt(newGeneration.Count - 1);
            }
        }

                        GenerationReport<T> report = new GenerationReport<T>(fitnessSorted,
selectedElements, deceasedElements, mutationResults, crossoverResults, newlyGenerated,
newGeneration);
        return report;
                }

                #endregion Erzeugen von Generationen
        }
}
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОХОДЖЕННЯ ЛАБІРИНТУ НА БАЗІ ГЕНЕТИЧНОГО АЛГОРИТМУ

Виконав: студент 2-го курсу,
групи 2КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Канаєв Є. Ю.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
_____ Арсенюк І. Р.
(прізвище та ініціали)
«_____»_____2022 р.

Вінниця ВНТУ – 2022 рік

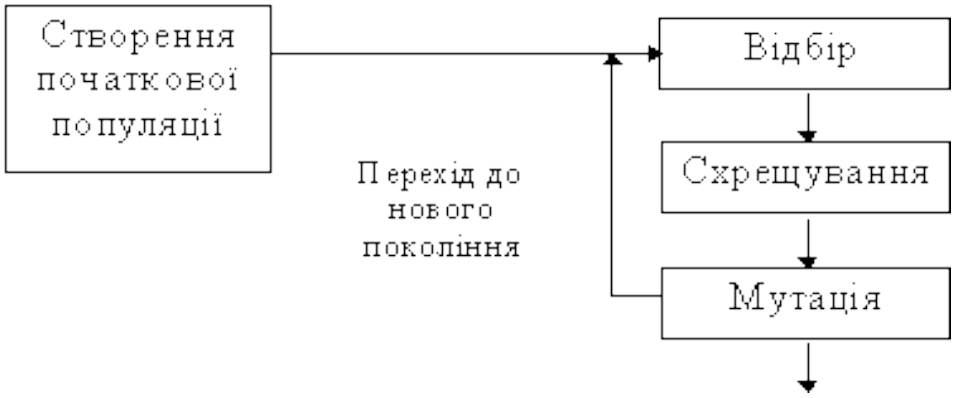


Рисунок В.1 – Схема основних вузлів роботи генетичного алгоритму

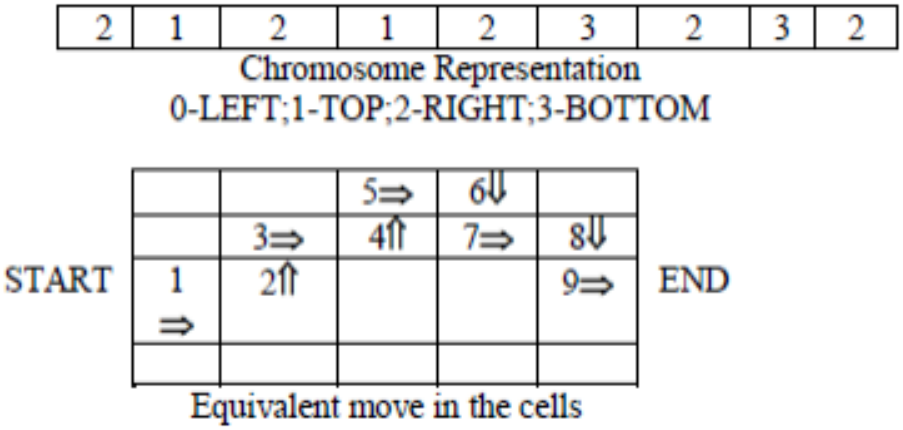


Рисунок В.2 – Представлення хромосоми

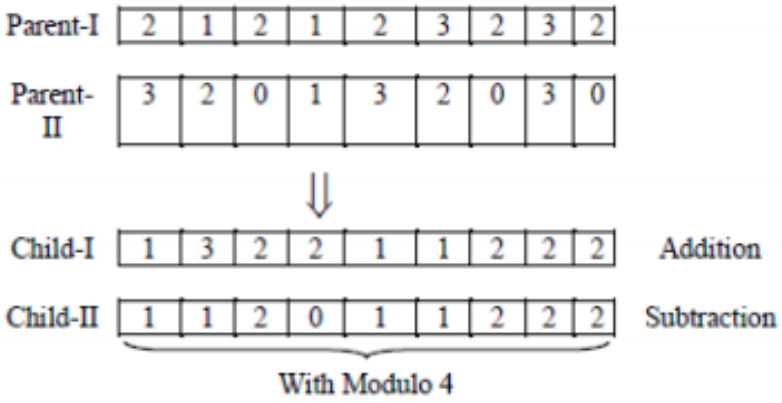


Рисунок В.3 – Операція кросовера

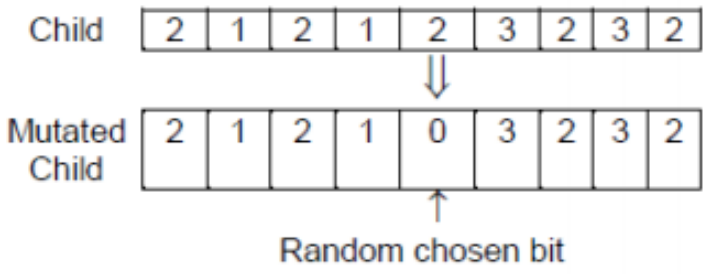


Рисунок В.4 – Операція мутації

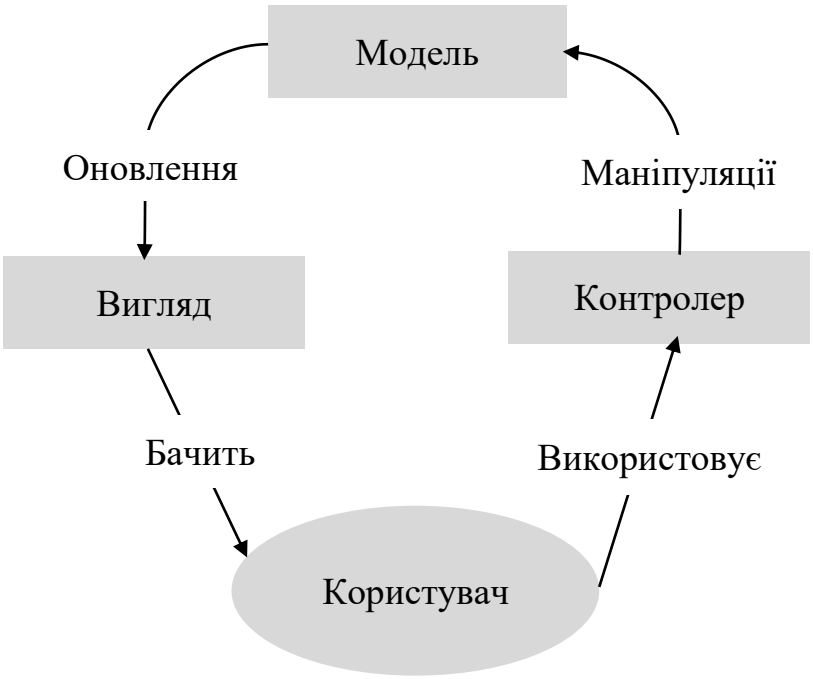


Рисунок В.5 – Використання архітектурного шаблону MVC

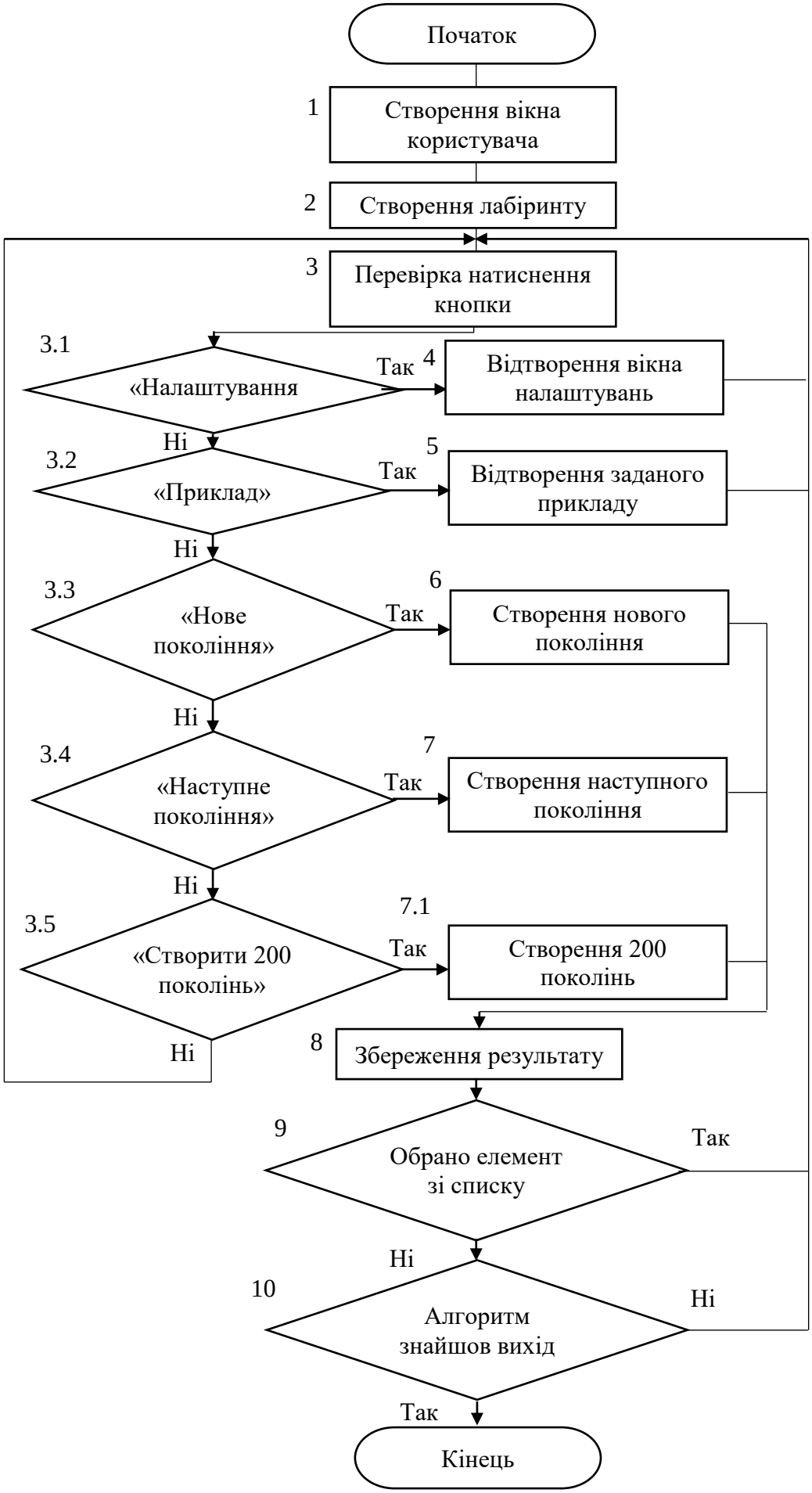


Рисунок В.6 – Схема загального алгоритму функціонування системи

Характеристика порівняння	C#	Java	C++
Швидкодія	+	-	-
Ручне управління пам’яттю	+	-	+
Перевантаження функцій	+	+	+
ООП	+	+	+
Шаблони	+	+	+
Умовна компіляція	+	-	+
Визначення макросів процесора	+	-	+
Значення параметрів за замовчуванням	+	-	+
Багатовимірні масиви	+	+	+
Динамічні масиви	+	+/-	+/-
Множинне наслідування	+	-	+
Особисте знання	+	-	+

Рисунок В.7 – порівнянно мов об’єктно-орієнтованого програмування

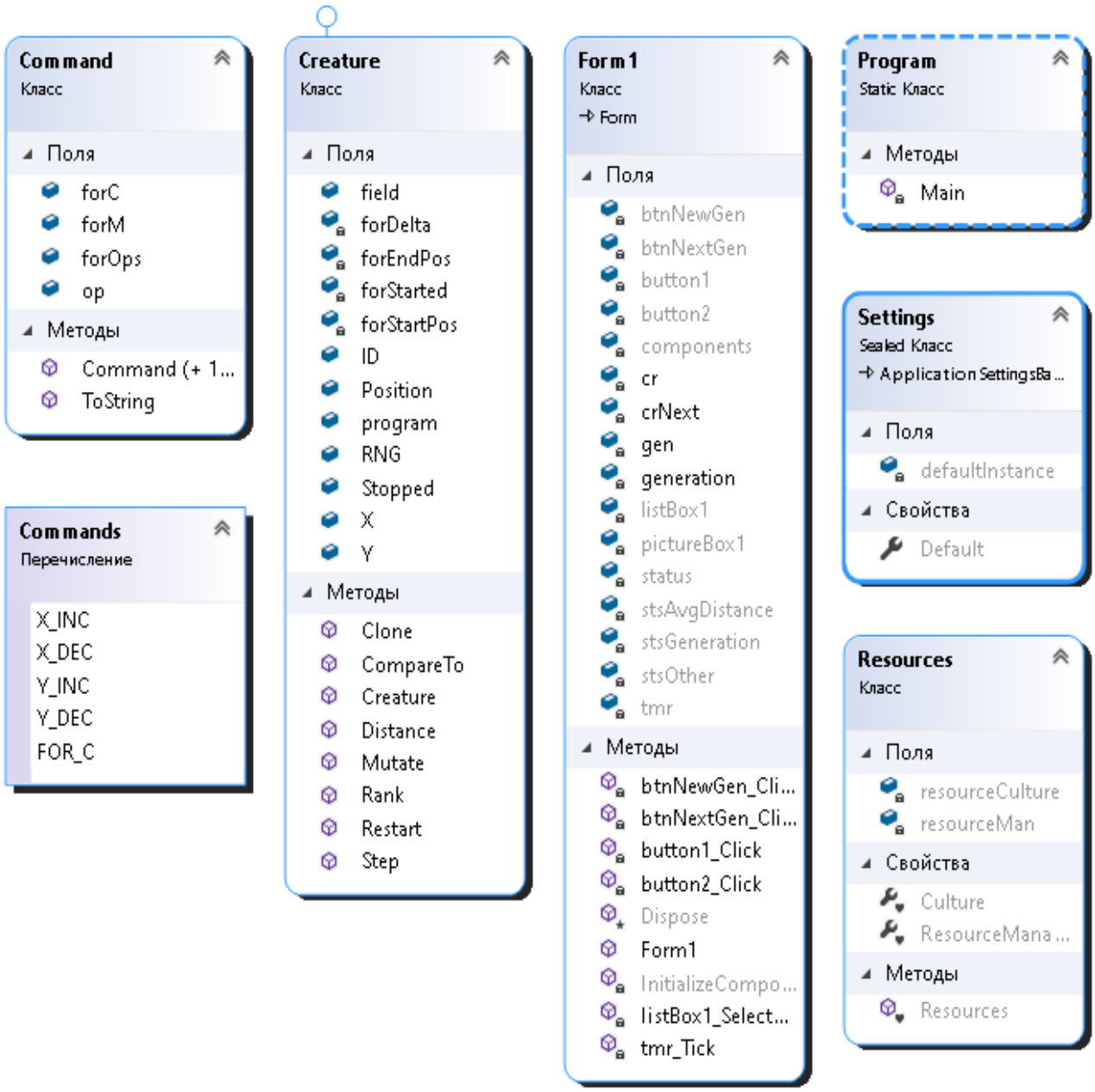


Рисунок В.8 – Загальна UML-діаграма класів

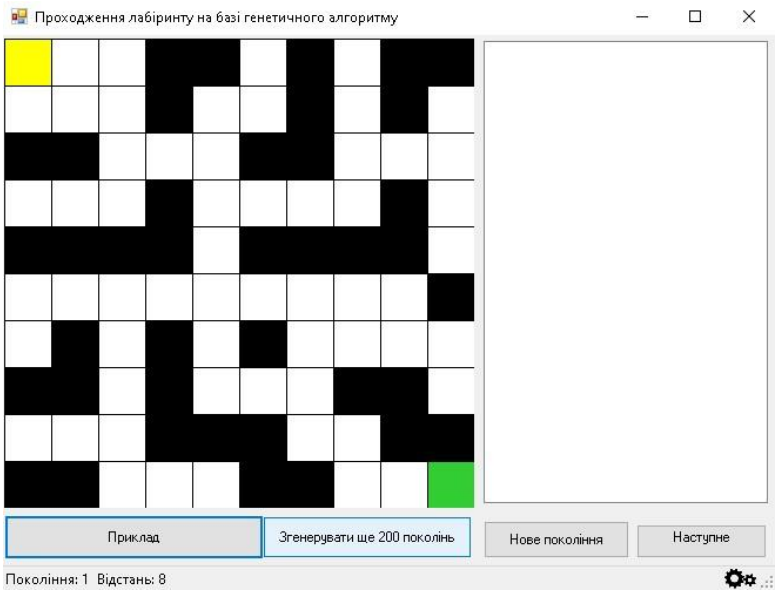


Рисунок В.9 − Вікно програми

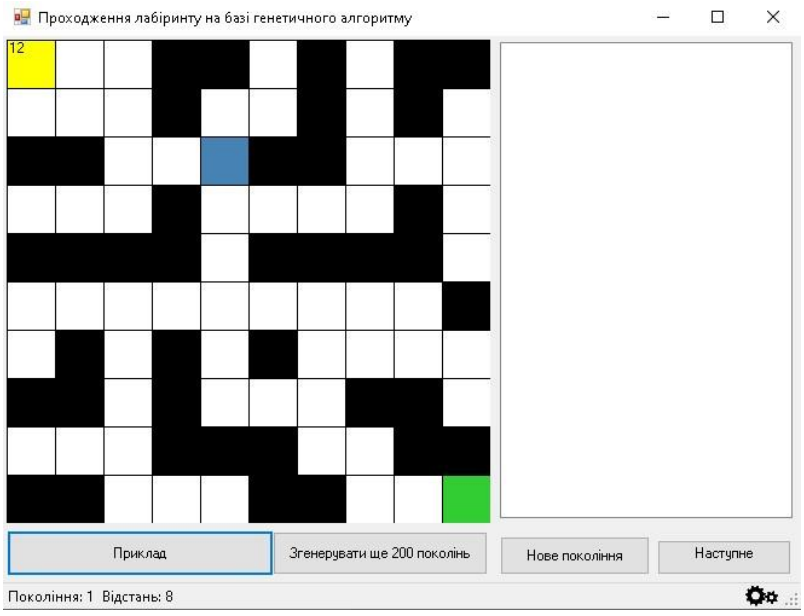


Рисунок В.10 – Приклад проходження лабіринту

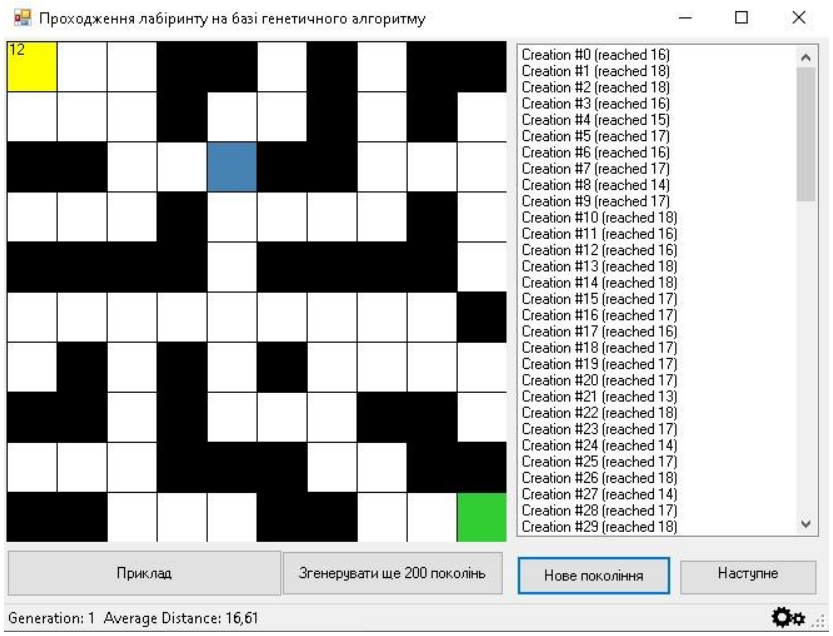


Рисунок В.11 – Створене нове покоління

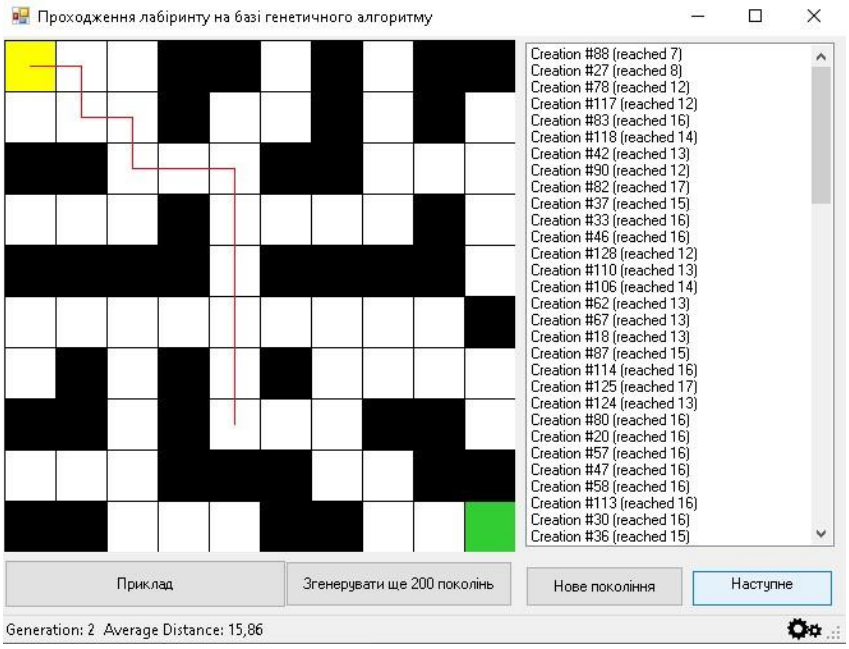


Рисунок В.12 – Створене наступне покоління

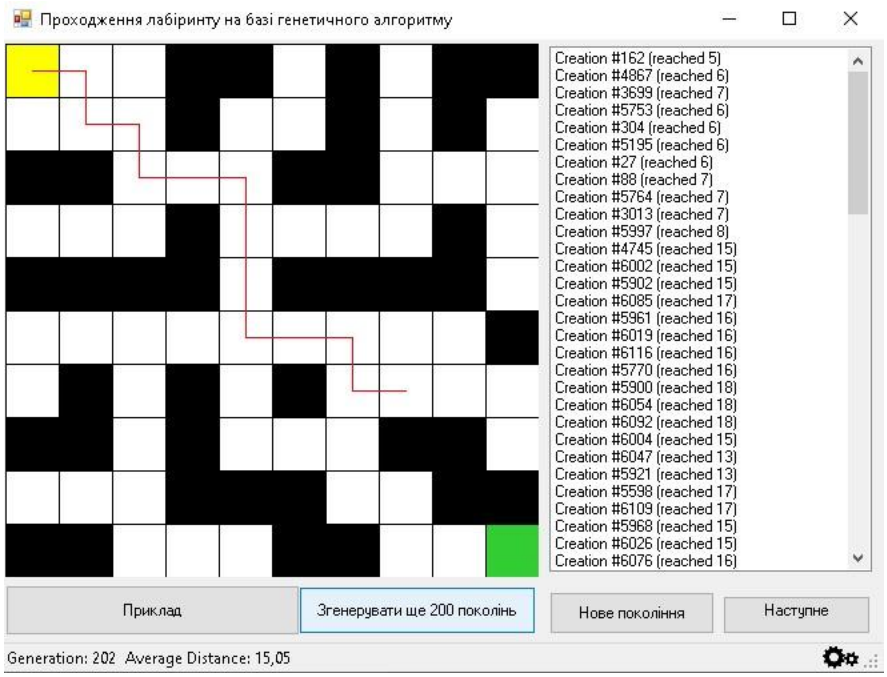


Рисунок В.13 – Створено ще 200 поколінь

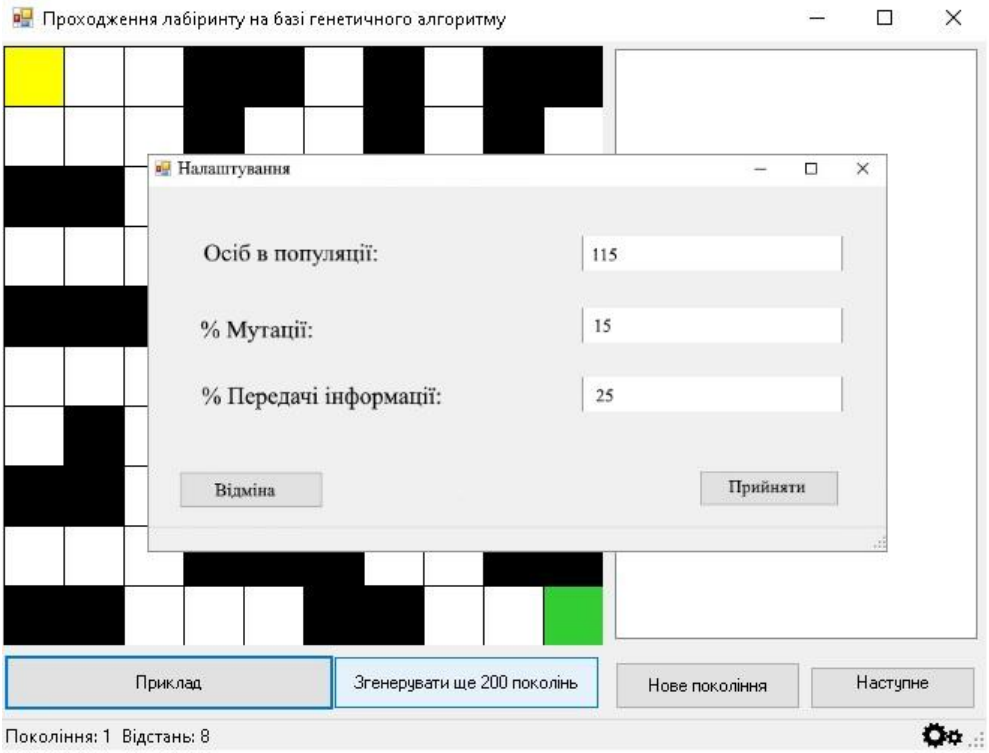


Рисунок В.14 – Вікно налаштувань

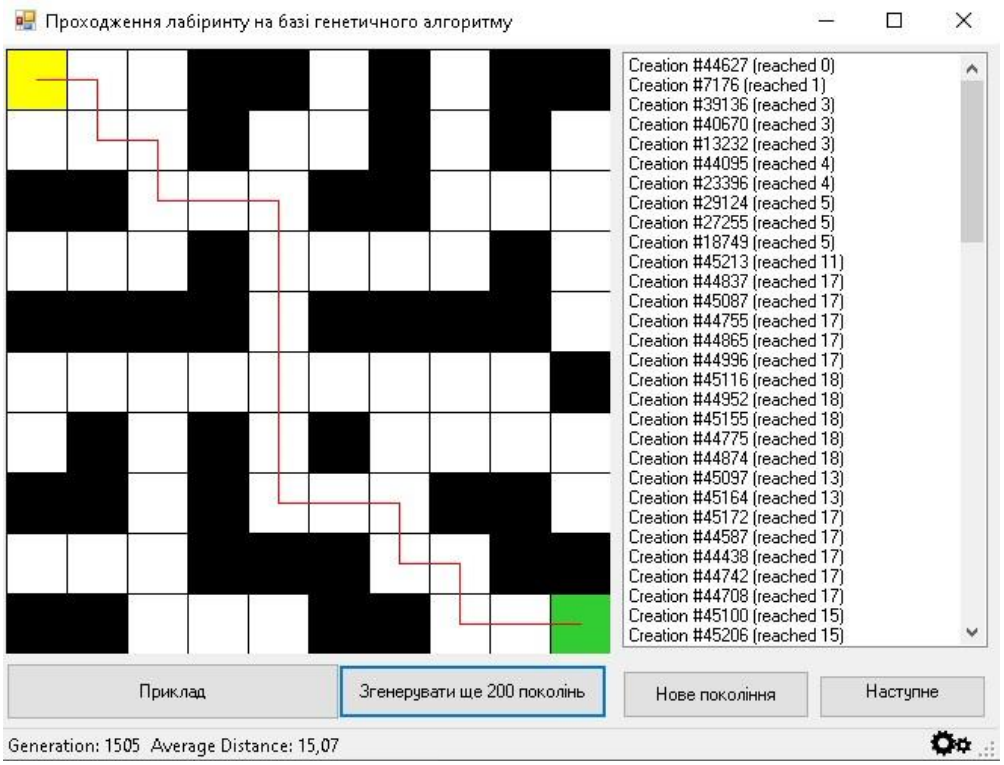


Рисунок В.15 − Результат проходження лабіринту

Алгоритм	Рішення	Гарантія	Незалежний від проходів
Пошук найкоротшого шляху	1 найкоротше	Так	Так
Random Mouse	1	Ні	Ні
Wall Follower	1	Ні	Так
Алгоритм Пледжа	1	Ні	Так
Алгоритм ланцюгів	1	Так	Так
Recursive Backtracker	1	Так	Так
Алгоритм Тремо	1	Так	Ні
Генетичний алгоритм	Всі, найкоротші	Так	Так

Рисунок В.16 – Порівняння основних результатів тестування з алгоритмами аналогами

Назва	Осіб в популяції	% мутації	% передачі і-ції	Кількість проходжень	Швидкість
A-Mazer	110	15%	25%	130	1 хв., 36 с.
MazeT	110	13%	26%	135	1 хв., 43 с.
Maze Navigation	125	16%	24%	133	1 хв., 40 с.
Codebox Maze	95	15%	25%	137	1 хв., 45 с.
Розроблений ПЗ	105	15%	25%	132	1 хв., 38 с.

Рисунок В.17 – Порівняння основних результатів тестування з програмами аналогами

Додаток Г (довідниковий)

Інструкція користувача

Для запуску розробленої програми проходження лабіринту на базі генетичного алгоритму потрібно відкрити файл MazeEvolution.exe.

Після запуску програми відкривається вікно інтерфейсу користувача (рисунок Г.1).

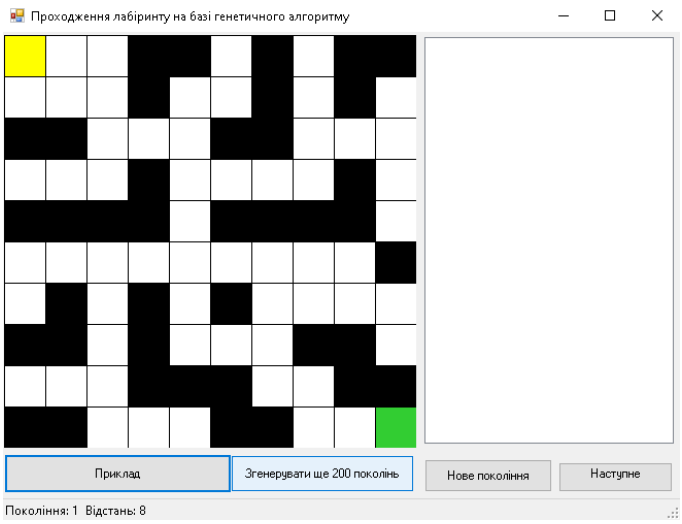


Рисунок Г.1 – Вікно програми

Якщо користувач натиснув кнопку «Приклад» (рисунок Г.2).

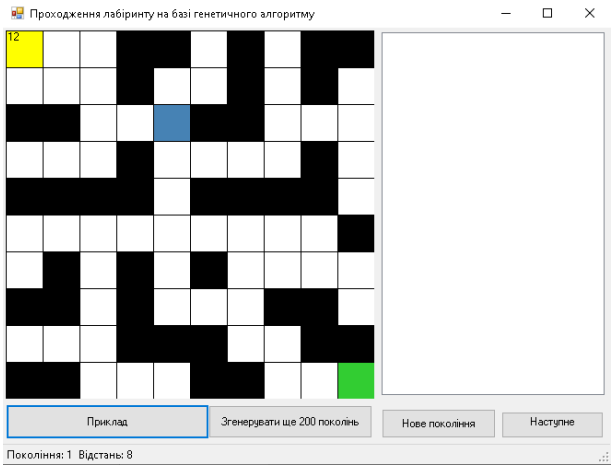


Рисунок Г.2 – Натиснута кнопка Приклад

Якщо користувач натиснув кнопку Нове покоління (рисунок Г.3).

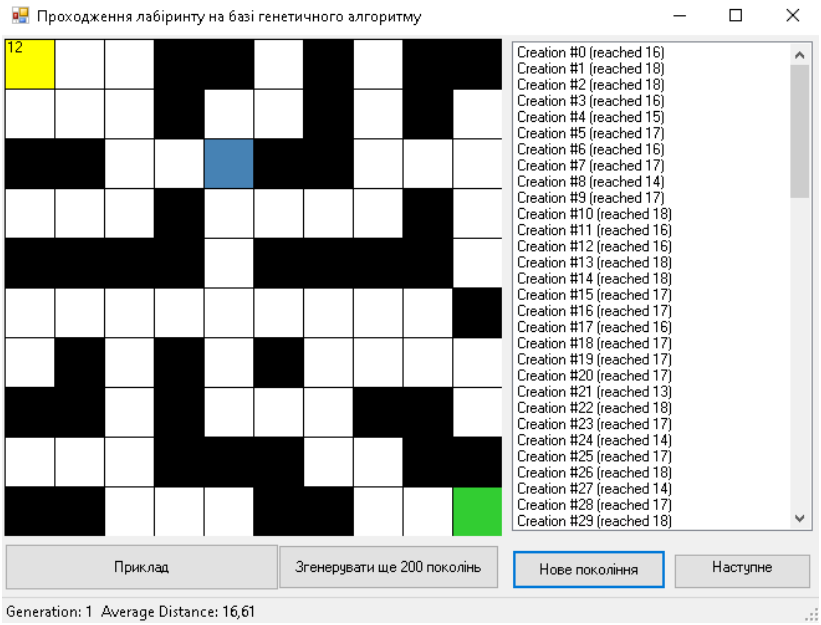


Рисунок Г.3 – Натиснута кнопка Нове покоління

Якщо користувач натиснув кнопку Наступне покоління (рисунок Г.4).

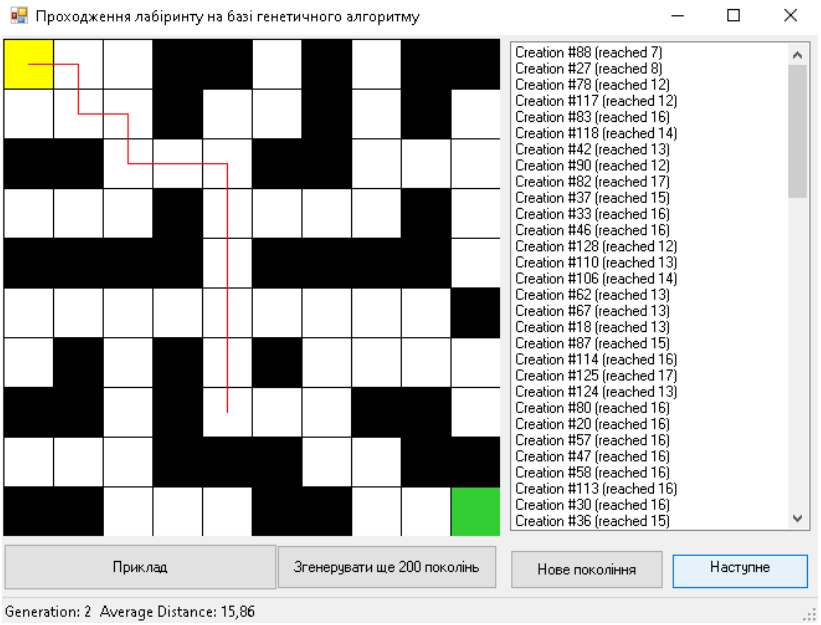


Рисунок Г.4 – Натиснута кнопка Наступне покоління

Якщо користувач натиснув кнопку Згенерувати ще 200 поколінь (рисунок Г.5).

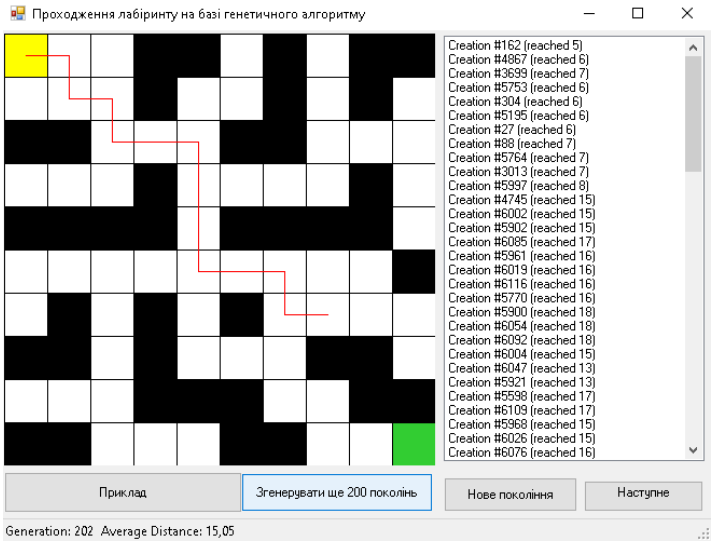


Рисунок Г.5 – Натиснута кнопка Згенерувати ще 200 поколінь

Якщо користувач натиснув іконку Налаштувань, відкривається вікно налаштувань вхідних параметрів генетичного алгоритму (рисунок Г.6).

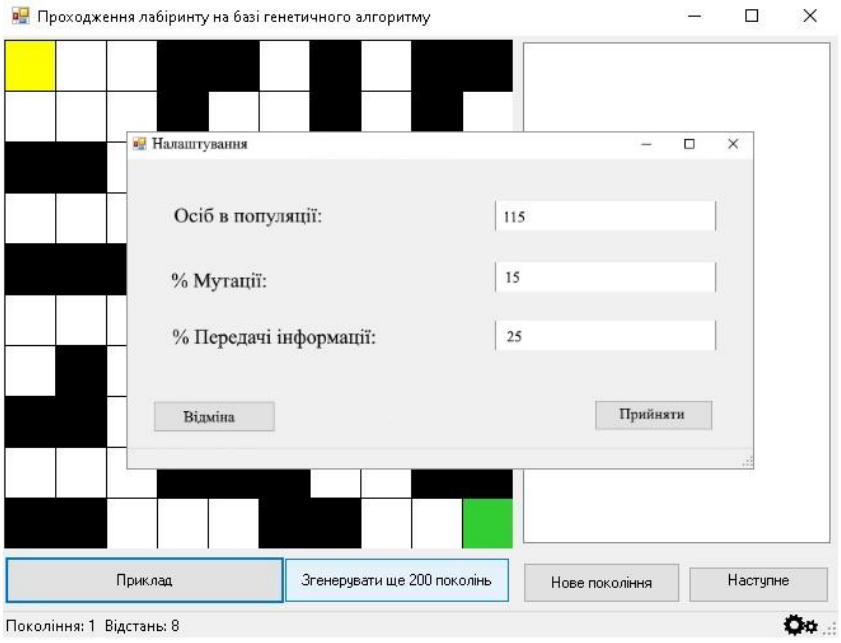


Рисунок Г.6 – Вікно налаштувань

Якщо алгоритм знайшов вихід з лабіринту (рисунок Г.7).

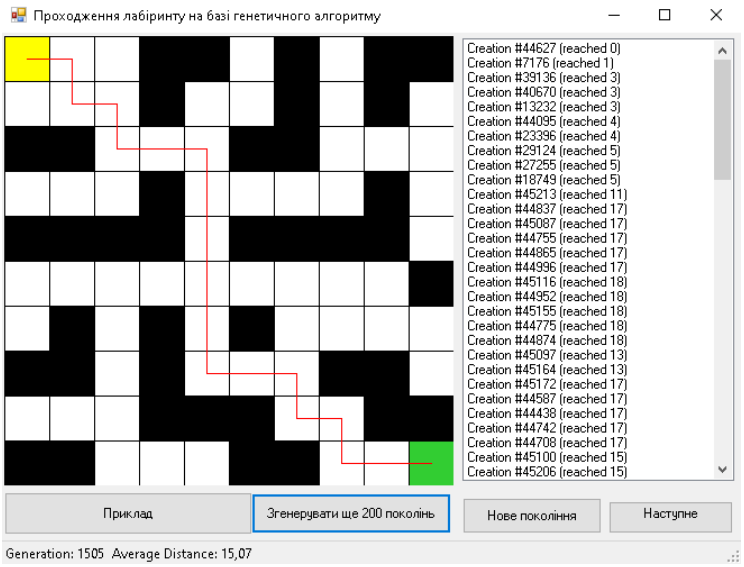


Рисунок Г.7 – Результат роботи алгоритму