Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи бакалавра

на тему: «Моделювання поведінки масових персонажів комп'ютерних ігор на базі клітинних автоматів»
за освітньою програмою: «Інженерія програмного забезпечення»
зі спеціальності: «121 Інженерія програмного забезпечення»
Виконав: студент групи «ПЗ1811»

_____ /Тімур КДИРОВ/
(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: _____ /ст. викл. Ірина ШАПОВАЛ/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: _____ /доц. Олена КУРОП'ЯТНИК/
(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Дніпро – 2022 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note
to Bachelor's Thesis

on the topic: «Computer games characters mass behavior modeling on the basis of cellular automata»
according to educational curriculum « Software engineering »
in the Speciality: «121 Software engineering»

Done by the student of the group PZ1811: /Timur KDYROV/

Scientific Supervisor: /Iryna SHAPOVAL/

Normative controller: /Olena KUROPIATNYK/

Dnipro – 2022

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

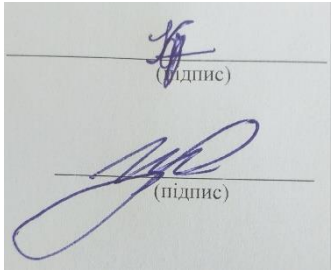
ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)
Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра
студенту Кдирову Тімуру Айткалійовичу
1. Тема роботи: «Моделювання поведінки масових персонажів комп'ютерних ігор на базі клітинних автоматів»
Керівник роботи: Шаповал Ірина Вікторівна, старший викладач
затверджені наказом № 77 ст від 08.12.2021
2. Строк подання студентом роботи: 11.06.2022 р.
3. Вихідні дані до роботи:
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати): вступ, збір вимог до програмного забезпечення, зовнішнє і внутрішнє проектування, тестування та налагодження, висновки, література.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): презентація, відео-демонстрація роботи програми.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі	31.01.2022 – 05.01.2022	
2	Огляд літератури та аналіз аналогів	16.01.2022 – 24.01.2022	
3	Розробка структур вхідних і вихідних даних	25.01.2022 – 02.02.2022	
4	Визначення вимог до програми. Вибір та обґрунтування мови програмування	03.02.2022 – 10.02.2022	
5	Узгодження та затвердження ТЗ	11.02.2022 – 18.02.2022	
6	Розробка та програмування логіки програми	19.02.2022 – 01.03.2022	
7	Розробка і реалізація інтерфейсу користувача	02.03.2022 – 20.03.2022	
8	Відлагодження програми	21.03.2022 – 24.03.2022	
9	Розробка, узгодження та затвердження програмної документації	25.03.2022 – 03.04.2022	
10	Подання кваліфікаційної роботи до кафедри	04.04.2022– 12.06.2022	
11	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	21.06.2022	

Студент  Тімур КДИРОВ
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник роботи (підпис) ст. викл. Ірина ШАПОВАЛ
(Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка складається з 7 розділів:

– вступ – в даному розділі описується сутність розробки, її актуальність. Складається з 1 сторінки;

– збір вимог до програмного забезпечення – у цьому розділі описуються аналоги програми та література по даній предметній області, а також проводиться опит зацікавлених сторін для формування найбільш повних вимог до програмного забезпечення. Складається з 10 сторінок;

– зовнішнє і внутрішнє проектування – у цьому розділі проведений огляд вхідних і вихідних даних, формалізація задачі, розробка фізичного проекту, приводиться опис об’єктно-орієнтованого проектування, проектування інтерфейсу користувача, ескізи форм, аналіз проекту, проектування динаміки системи, вибір мови програмування. Складається з 31 сторінки;

– тестування та налагодження – включає в себе вибір стратегії тестування, опис тестів методами «чорної» та «білої» скриньки. Також аналіз помилок їх вплив на систему та вирішення проблеми. Складається з 18 сторінок;

– висновки. Складається з 2 сторінок;

– список літератури – включає в себе бібліографічний список використаної літератури. Складається з 1 сторінки;

– додатки – містить робочий проект додаткового без .txt файлу та зображень для форм.

Кількість таблиць: 23 штуки.

Кількість рисунків: 8 штук.

Ключові слова: клітинний автомат, імітація, моделювання.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

КА (Клітинний автомат) – дискретна математична модель, яка визначає сукупність та описується набором клітинок, що утворюють періодичну решітку, та заданими правилами переходу, що визначають стан клітини за теперішнім станом самої клітинки та тих її сусідів, що знаходяться від неї на певній відстані;

ЕОМ (електронно-обчислювальна машина) – загальна назва для обчислювальних машин, що є електронними (починаючи з перших ламповых машин, включаючи напівпровідникові тощо) на відміну від електромеханічних (на електричних реле тощо) та механічних обчислювальних машин;

НП (неігровий персонаж) – у комп’ютерних на настільних рольових іграх, терміном НП позначають персонажів, що (частіше за все) спілкуються із гравцем. Вони можуть бути як нейтральними так і дружелюбними і навіть ворожими, по відношенню до гравця. Якщо казати простіше, до НП належать усі персонажі (до котрих відносяться і всі створіння у грі) за котрі виконує дії система (комп’ютер), а не гравець.

ВСТУП

У наш час комп'ютерних технологій, прогресує дослідження інформаційних моделей. Дослідження клітинних автоматів і є одним з методів комп'ютерного моделювання. Завдяки йому, можна змоделювати найрізноманітніші процесі та об'єкти, як наприклад: розвиток різних колоній, поширення мікроорганізмів, та навіть космічних процесів. Завдяки використанню саме клітинних автоматів ми маємо свободу у виборі правил та структур розвитку системи.

На даний час існує дуже багато найрізноманітніших програмних засобів, що надаюсь можливість використовувати метод моделювання клітинних автоматів. У той же час, саме через поширення комп'ютерних технологій, всі ці програмні засоби включають і багато інших задач, підзадач та функціоналу, що призводить до важкого розуміння усіх можливостей, та необхідність у написанні частини із еволюціонуванням клітинного автомату з боку користувача. Підсумувавши, все сказане раніше, я вирішив розробити свій програмний засіб, що буде спрямований на надання необхідного функціоналу користувачеві для моделювання клітинними автоматами, а все що буде потрібно від користувача, це запустити програму та вибрати певну модель.

Задачею даного дипломного проекту є проектування програмного продукту, що буде моделювати протікання процесів поведінки певного характеру: гра «хижак-жертва», гра «життя», гра «мурахи» та гру «Арена». Все що потребується від користувача – це вибрати певну модель що описує необхідний процес. Кінцевим користувачем програмного засобу може бути як розробник ігор так і проста зацікавлена особа.

1. ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Огляд програмних аналогів

Серед програмних засобів що використовуються для розробки імітаційних моделей, можна виділи програмний продукт «Simulink» та платформу «TerraME».

Імітаційні середовища не потребують програмування у виді послідовності команд. Замість написання програми користувачі складають модель з бібліотечних графічних модулів і заповнюють спеціальні форми. Як правило імітаційне середовище забезпечує можливість візуалізації процесу імітації, а також дозволяє сценарний аналіз і пошук оптимальних рішень [1].

1.1.1. Огляд програмного продукту «Simulink»

Simulink – інтерактивний інструмент для моделювання, імітації та аналізу динамічних систем, включаючи дискретні, неперервні та гібридні, нелінійні та розривні системи. Розроблений компанією The MathWorks. Дає можливість будувати графічні блок-діаграми, імітувати динамічні системи, досліджувати працездатність систем і вдосконалювати проекти.

Програма Simulink – це додаток до пакету MATLAB. При моделюванні з використанням Simulink реалізується принцип візуального програмування, відповідно до якого, користувач на екрані з бібліотеки стандартних блоків створює модель пристрою і здійснює розрахунки. При цьому, користувачеві не потрібно досконально вивчати мову програмування і чисельні методи математики, а досить загальних знань потрібних при роботі на комп'ютері і знань тієї предметної області в якій він працює [2].

Simulink є досить самостійним інструментом MATLAB і при роботі з ним зовсім не потрібно знати сам MATLAB і інші його додатки. З іншого боку доступ до функцій MATLAB і іншим його інструментам залишається відкритим і їх можна використовувати в Simulink.

При роботі з Simulink користувач має можливість модернізувати бібліотечні блоки, створювати свої власні, а також складати нові бібліотеки блоків.

При моделюванні користувач може вибирати метод розв'язання диференціальних рівнянь, а також спосіб зміни модельного часу (з фіксованим або змінним кроком). У ході моделювання є можливість стежити за процесами, що відбуваються в системі. Для цього використовуються спеціальні пристрої спостереження, входять до складу бібліотеки Simulink. Результати моделювання можуть бути представлені у вигляді графіків або таблиць [2].

Перевага Simulink полягає також у тому, що він дозволяє поповнювати бібліотеки блоків за допомогою підпрограм написаних як мовою MATLAB, так і на мовах C++, Fortran і Ada .

1.1.2. Огляд платформи «TerraME»

Серед програмних продуктів, які використовуються для розробки моделей складних систем на основі клітинних автоматів, можна виділити TerraME.

TerraME – платформа для моделювання екологічних систем, яка пропонує концептуальну основу, і послуги зі створення екологічних моделей за допомогою мови програмування високого рівня – Terra Modeling Language. Використання мови програмування все ще є обмеженням, оскільки основні користувачі – це дослідники, яким зазвичай не вистачає базових знань алгоритмів і методів програмування.

TerraME підтримує концепцію вкладених клітинних автоматів. Просторової динамічної моделлю є модель, у якій час і місце є незалежними змінними. Результати цих моделей – карти, які зображують просторовий розподіл рисунка або безперервної змінної. TerraME дозволяє симулювати в двовимірних клітинних просторах і використовувати просторові дані для зберігання і пошуку. TerraME надає інтерфейс для TerraLib – географічних баз даних. Terra Modeling Language має вбудовані функції, що робить його

легшим для розробки різного масштабу. Серед типових застосувань TerraME є земельні зміни та гідрологічні моделі [3].

1.2. Огляд літератури

1.2.1. Клітинні автомати

Клітинні автомати є дискретними динамічними системами, поведінка яких повністю визначається в термінах локальних залежностей [4]. Клітинний автомат (КА) був запропонований фон-Нейманом в середині минулого століття [5]. На основі цієї моделі підтверджувалася думка про те, що людина може створити пристрій, який буде володіти притаманній їй самій властивостями, зокрема, здатністю відтворювати собі подібного. Тому КА довгий час розглядався як модель самовідтворення і тлумачився як спрощена модель деякого біологічного співтовариства, що складається з безлічі клітин. Кожній клітці ставиться у відповідність кінцевий автомат, званий елементарним автоматом, який може знаходитися в одному з двох станів: 0 і 1 (чорне або біле) і змінювати цей стан $0 \rightarrow 1$ або $1 \rightarrow 0$ залежно від станів клітин деякого свого оточення, званого сусідством.

Алгоритм обчислення наступного стану в залежності від станів сусідів (функція переходів елементарного автомата) у всіх клітин однаковий. Всі клітини виконують перехід у новий стан одночасно (синхронно та паралельно). При цьому спостерігається зміна глобальної чорно-білої картини розподілу станів простору КА. Така картина називається конфігурацією КА.

Ітеративна зміна конфігурації при переходах всіх елементарних станів в нові стани називається еволюцією КА. Еволюціонуючи КА моделює просторову динаміку, яка може мати завершення, повторюватися періодично або змінюватися хаотично.

За своїми поведінковими властивостями КА розбиваються на 4 класи [6], що розрізняються типами конфігурацій, до яких еволюціонує КА.

Анопрієнко [7] пропонує класифікацію, яка базується на принципі варіювання таких параметрів КА, як стан елементів, геометрія КА, тип

сусідства, локальні правила, способи організації автомата за часом. Деякі типи КА виходять шляхом поєднування різних варіантних параметрів.

За просторовими характеристиками КА діляться на одномірні і багатовимірні залежно від кількості просторових координат сітки автомата.

З геометричних характеристик дуже умовно виділені около і ізометрія. Около, яке складається з клітин, що мають спільну вершину з даною, називається около Мура. Около, яке складається з клітин, що мають спільну сторону з даної, називається около фон Неймана.

За способом формування правил переходу КА можна розділити на:

– детерміністичні (класичний). Стан комірки у наступний момент часу однозначно визначається станом цієї комірки і станом її найближчих сусідів у попередній момент часу. У цьому випадку стан даного елемента в момент часу $n+1$ є однозначною функцією F двох змінних – стану цього елемента і суми станів його найближчих сусідів у попередній момент часу n. При такому визначенні клітинний автомат не має пам'ять;
– ймовірнісні. Стан комірки у наступний момент часу визначається на основі деяких ймовірностей. У таких автоматах при моделюванні деякого процесу для кожної комірки автомата датчиком випадкових чисел генерується випадкове число $\theta(0<\theta<1)$, яке порівнюється з ймовірністю W реалізації цього процесу. Якщо $\theta<W$, то процес реалізується. У імовірнісних КА замість функції F необхідно задати набір імовірностей зміни стану клітини, які показують, якою буде ймовірність переходу i-го елемента із стану в n-й момент часу в стан в наступний $n+1$- й момент часу за умови, що стану його найближчих сусідів у n-й момент часу брали певні значення;
– узагальнюючі. Правила залежать тільки від загального числа значень сусідніх комірок.

За ознакою однорідності КА поділяють на:

– однорідні. Локальні функції переходів і індекси сусідства однакові для

кожної клітини;

– неоднорідні. Клітини можуть мати різні функції переходу або різні індекси сусідства.

Основні переваги КА- моделей:

– відсутність помилок округлення. Оскільки стани клітин представлені булевими векторами, і обчислення ведуться над ними, всі розряди в них рівноправні, округлення не проводиться. При ітеративних обчисленнях це буває важливо;
– економія пам'яті при великих обсягах даних, так як булеві дані розміщуються у всіх розрядах машинних комірок;
– віртуальна необмеженість паралелізму. Допускається розрізання області моделювання будь-яким чином. Розпаралелювання на будь-яку кількість процесорів не призводить до деградації ефективності розпаралелювання;
– простота завдання граничних умов. Межі областей моделювання представлені особливими клітинами, які мають граничні правила переходів. Навіть при дуже складних формах кордонів (пористі середовища, вигнуті трубки) алгоритм моделювання практично не ускладнювати в порівнянні з тим, коли перешкод немає.

Безсумнівно, КА – моделі і засновані на них алгоритми мають і недоліки, з якими доводиться боротися. Головними з них є наступні:

– автоматний шум, пов'язаний з дискретним поданням даних. Уникнути його неможливо, але при досить великому радіусі усереднення, він стає непомітним;
– відсутність формальних методів «передбачення» поведінки КА, по заданих функцій переходу. Звідси відсутність методів синтезу функції переходу КА по заданих властивостям його еволюції.

1.2.2. Імітаційне моделювання

Моделювання починається з формування предмета дослідження – системи понять, які відображають істотні для моделювання характеристики об’єкта.

Спочатку розберемося з фундаментальними поняттями, такими як система, модель, моделювання, імітація.

На даний час при аналізі і синтезі складних систем отримав розвиток системний підхід, який передбачає послідовний перехід від загального до часткового, коли в основі розгляду лежить мета, причому об’єкт дослідження вирізняється з навколишнього середовища [8].

Моделлю є представлення об’єкта, системи або поняття у деякій формі, відмінній від форми їх реального існування [9]. Модель – це опис системи, що служить зазвичай засобом, який допомагає нам у поясненні, зрозумінні або вдосконаленні цієї системи. Модель будь-якого об’єкта може бути або точною копією цього об’єкта, або відображати деякі характерні властивості об’єкта в абстрактній формі. Так як представлення об’єкта, системи або поняття за допомогою моделі носить загальний характер, дати повну класифікацію всіх функцій моделі складно.

Моделі використовуються, як засіб професійної підготовки і навчання. Також моделі використовують для прогнозування поведінки об’єктів моделювання. Використання моделей дозволяє проводити контрольовані експерименти в ситуаціях, де експериментування на реальних об’єктах було б практично неможливим або економічно недоцільно. Безпосередньо експериментування з системою зазвичай складається у варіюванні її деяких параметрів; при цьому, підтримуючи всі інші параметри незмінними, спостерігають результати експерименту.

Модель може слугувати для досягнення однієї з двох основних цілей: або описової, якщо модель слугує для пояснення і кращого зрозуміння об’єкта, або предписуючої, коли модель дозволяє передбачити і відтворити характеристики об'єкта, які визначають його поведінку. Модель предписуючого типу зазвичай є і описуючою, але не навпаки.

Сам процес дослідження об'єктів або систем на їх моделях, побудови моделей реально існуючих об'єктів, процесів або явищ з метою отримання пояснень цих явищ, а також для передбачення явищ, що цікавлять

дослідника, називається, моделювання. У науці не існує єдиної класифікації видів моделювання системи і виділяють близько 15. Детально їх Совєтов [10].

Розглянемо коротку класифікацію видів моделювання систем (рис.1.1) [10]:

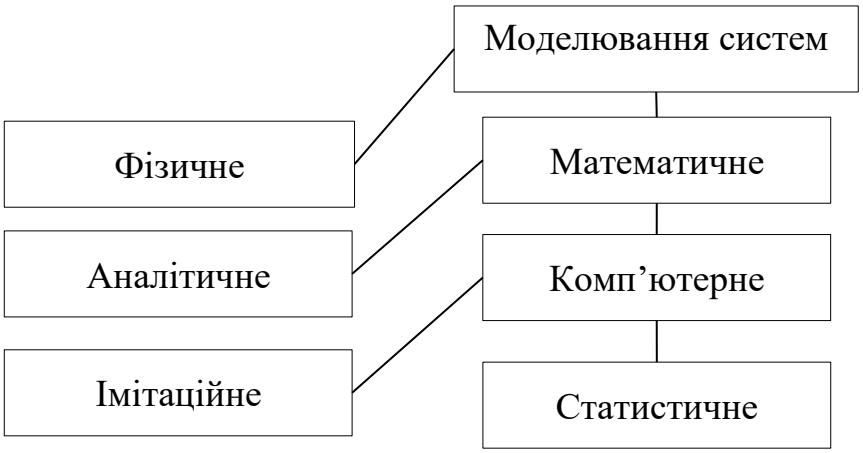


Рисунок 1.1 – Класифікація видів

Фізичне моделювання передбачає, що у якості моделі використовується або сама досліджувана система, або інша система с тією же або подібною фізичною природою.

Під математичним моделюванням розуміють процес встановлення відповідності даної реальної системи S деякій математичній моделі M і дослідження цієї моделі, яке дозволяє отримати характеристики реальної системи.

Для аналітичного моделювання характерне те, що процеси функціонування елементів системи записуються у виді деяких математичних співвідношеннях або логічних умов.

Для комп'ютерного моделювання характерно, що математична модель системи представлена у виді програми для ЕОМ – комп'ютерній моделі, яка дозволяє проводити з нею обчислювальні експерименти.

Імітаційне моделювання – це вид комп'ютерного моделювання, для якого є характерним відтворення на ЕОМ процесу функціонування досліджуваної складної системи. При цьому імітуються елементарні явища, які складають

процес, зі збереженням їх логічної структури, послідовності протікання у часі, що дозволяє отримати інформацію о стані системи S у задані моменти часу.

Статистичне моделювання – це вид комп'ютерного моделювання, яке дозволяє отримати статистичні дані о процесах у модельованої системі S.

Імітаційною моделлю складної системи S називаються машині програми (або алгоритми), які позволяють імітувати на ЕОМ поведінку окремих елементів системи S і зв'язків між ними у перебігу заданого часу моделювання.

Особливості імітаційних моделей:

– під час створення імітаційної моделі, достатньо знань алгоритмів, що описують поведінку та зв'язки окремих елементів системи (бо у цілому закон функціонування всієї системи може бути невідомим);
– в імітаційній моделі значення характеристик, що досліджуються визначаються у ході самого експерименту імітації на ЕОМ, а зв'язки характеристик та параметрів системи виявляються.

Способи взаємодії дослідника із моделлю у ході моделювання імітаційні моделі поділяються на діалогові та автоматичні. Діалоговими називаються такі імітаційні моделі, які позволяють користувачу керувати ходом моделювання більш активно, тобто: призупиняти сеанс моделювання, вносити зміни у модель та інше. До автоматичних відносяться імітаційні моделі, взаємодія котрого зводиться тільки до вводу користувачем вхідної інформації і управління початком і закінченням роботи моделі.

Частіше за все, складна система «S», складається з багатьох елементів всі елементи котрої функціонують одночасно. Але нажаль на даний час у більшості сучасних ЕОМ неможливе паралельне виконання декількох програм, що імітують поведінку окремих чи певних елементів системи.

1.3. Різновиди персонажів комп’ютерних ігор та їх поведінка

Більшість ігрових нетрадиційних (людських) персонажів, є надприродні людиноподібні створіння. Розглянемо найвідоміших та тих, що використовуються найчастіше:

Гоблін – Найпоширеніший НП ворог у комп'ютерних іграх. Переважно уявлявся як потворний карлик, найчастіше зустрічається у групах або навіть у натовпах, що може перевищувати кількість у сотні одиниць персонажів. Вони не відрізняються розумом і основними діями у процесі життя є слідування первинним інстинктам, а саме у розмноженні та вбивствах. Характеристики гоблінів відносно людських – дуже низькі, приблизно четверть від середньостатистичної людини, але через перевагу у кількості та несподіваним атакам зазвичай переважають над середньостатистичними людьми. Тривалість життя у звичайних гоблінів, становить приблизно 1 рік, через що в них й високий рівень підвищення популяції;

Орк – Зазвичай зображується кремезним варваром із звіриними рисами. Зустрічаються у меншій кількості за гоблінів, та відрізняються більш розвиненим інтелектом на відміну від гоблінів та кобольдів, та є більш наближений до людського, що призводить до появи певних координаційних дій у боях та полюванні. Характеристики Орків переважають над людськими у три–чотири рази. Тривалість життя дуже близька до людського але через агресивну натуру та дикість, частіше всього вмирають у бою;

Кобольд – Найчастіше зображуються спотвореними людиноподібними тваринами–рудокопами, що переважно більшістю знаходяться у шахтах, та тільки маленькими групами поряд із ними. Мають характеристики, що більш походять на гоблінів, але більш розвинена комунікація та стадні інстинкти. Характеристики кобольдів лише трішки більше за гоблінів. Тривалість їх життя у середньому один–два десятка років.

Висновки до пункту 1.

1) КА «Життя» – є найбільш стандартною моделлю КА, що може зобразити майже все, але в той же час потребує більш детальних коригувань. Надану

модель можна розглядати в якості основи для створення як повсякдення селищ персонажів, що бігають, роблять та будуть щось, так і для моделювання ігрових локацій якими буде пересуватися основний персонаж;

2) КА «Мурахи» – є прикладом моделювання цілком сталого повсякдення певних персонажів, як на приклад ті самі кобольди, що копаються у своїх шахтах, розкопуючи собі житло;
3) КА «Хижак і здобич» – модель, що представляє моделювання архетипів двох персонажів, один з котрих є «пасивним» (або як приклад, боягузливим) що при змозі лише тікає від загрози, котру представляє другий персонаж. Він відрізняється від першого, агресивністю, та бажанням знищити, з'їсти, тощо першого персонажа. До прикладу таких персонажів можна підставити тих самих Гоблінів та Орків. Гобліни навіть у кількісній перевазі не матимуть хоробрості протистояти Оркам, котрі бачать у гоблінах якщо не їжу то ціль, яку вони матимуть здолати;
4) КА «Арена» – моделювання може підходити під демонстрацію поведінки найрізноманітніших персонажів ігор. В більшості випадків, а саме – обмежений простір, та присутність інших видів/груп персонажів – призводить до агресивної поведінки із можливістю вмерти як у бою, так і від віку, що й реалізується у наданій моделі.

2. ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЕКТУВАННЯ

2.1. Зовнішнє проектування

2.1.1. Функціональне призначення

Функціональним призначенням програми є вибір із декількох запропонованих моделей на основі клітинного автомату для моделювання двовимірної системи поведінки масових персонажів комп’ютерних ігор.

Також функціональним призначенням програми є зміна положення відображення, а також декілька інших параметрів масштабування та відображення, що можна змінити шляхом попереднього вводу у .txt файл: масштаби екрану, максимальну кількість кадрів в секунду, розмір клітин та кількість персонажів на старті гри.

2.1.2. Експлуатаційне призначення

Експлуатаційне призначення програми:

– створення більш відповідної для розробників ігор імітаційної моделі поведінки;
– збільшення зацікавленості із боку розробників комп’ютерних ігор цілеспрямованістю програмного засобу;
– збільшення продуктивності в цілому роботи розробників чи зацікавлених осіб у аналізі моделей поведінки, що виражається конкретизацією програми у певному напрямі із певним (мінімально необхідним) інструментарієм на відміну від аналогів.

2.1.3. Функціональні вимоги

Програма повинна надавати можливість:

– завантажувати з файлу:
    – розгортка екрану;
    – розмір клітин у пікселях;
    – кількість персонажів на старті моделювання.
– редагування параметрів у файлі;
– зупиняти та відновлювати моделювання;

– отримати візуалізацію процесу імітації.

2.1.4. Вхідні та вихідні дані

2.1.4.1. Вхідні дані

Вхідними даними є:

1) m_livingWarriors – кількість живих персонажів на дискретний момент часу моделювання, значення – натуральне число;
2) m_ants – кількість персонажів мурах, значення – натуральне число;
3) m_drawGrid – перемикач відображення сітки;
4) change – зміщення положення екрану по осі ординат, значення – ціле число;
5) zoom – масштабування положення екрану, значення у десяткових дробах;
6) m_pause та pause – стан моделювання, одне з двох значень false чи true;
7) Значення що отримуються з файлу «config.txt» (наступні значення з поміткою «Тегами» із наведеного переліку):
   – fps – обмеження кількості кадрів за секунду, має один параметр, значення – натуральне число, мінімальне значення 10, максимальне відсутнє;
   – winx – параметр ширини екрану відображення – має один параметр, значення – натуральне число, мінімальне значення 100, максимальне 1920;
   – winy – параметр висоти екрану відображення – має один параметр, значення – натуральне число, мінімальне значення 100, максимальне 1080;
   – cellsize – розмір клітини, має один параметр, значення – натуральне число, мінімальне значення 2, максимальне 30;
   – WarriorsAtStart – зберігає кількість персонажів на початок моделювання гри Арена, один параметр, значення – натуральне число, мінімальне відсутнє, максимальне 1000;

Формат файлу «Config.txt» :

Тег

Значення

Тег

Значення

…

Розміщення порядку Тегів допускається у довільному порядку. Для кращої читабельності допускається відокремлення Тегів від Значення пустим рядком.

2.1.4.2. Вихідні дані

Вихідними даними є:

– fps – кількість кадрів в секунду, що відображаються у ході моделювання у виді надпису та самого лічильника по верх моделі;
– year – лічильник покоління, що також відображається у ході моделювання у виді надпису та самого лічильника по верх моделі;
– getNbrWarriors – певна кількість живих персонажів, що знаходиться в певний час на екрані у виді надпису та самого лічильника по верх моделі;
– m_BuildGrid – відображення сітки КА в двовимірній матриці у виді ліній що розділяють поле моделювання на клітини.

2.1.5. Опис зовнішнього інформаційного середовища

Дані файлу Config.txt, що наводяться в пункті [2.1.4.1]. Для функціонування системи потрібна операційна система Windows 8 та новіше, наявність стандартних бібліотек та додаткової бібліотеки SFML 2.5.1 та новіше.

Розглянемо головні задачі та сценарії, що їх реалізують:

1) підготувати файл конфігурації: якщо не проводилося ніяких замін або збійних ситуацій (файл не відкрився, дані неправильно тощо) – задаються параметри за замовченням у іншому випадку параметри завантажуються з файлу;

2) контроль моделювання: обов’язковою частиною моделювання є його візуалізація. Під процесу моделювання, користувач може контролювати стан моделювання: зупиняти та відновлювати процес моделювання, масштабувати та рухати зону візуалізації відносно вікна.

Специфікація функціональних вимог виконана у вигляді діаграми прецедентів (рис. 2.1).

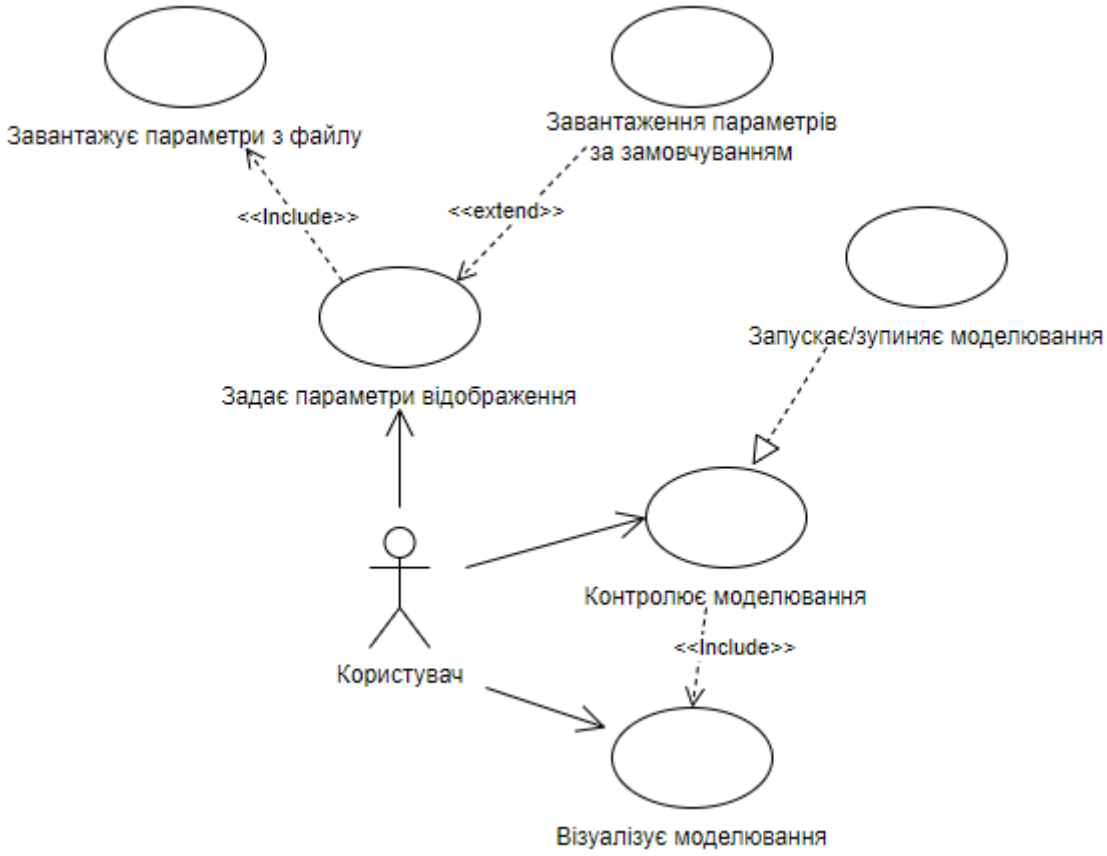


Рисунок 2.1 – Діаграма прецедентів

2.2. Внутрішнє проектування

2.2.1. Аналіз зовнішніх специфікацій систем

2.2.1.1. Моделювання словника системи

Розглянувши математичні моделі та сценарії взаємодії користувача із системою, можна виділити базові сутності:

Ідентифіковані сутності: Нова Мураха, Мураха, Додаток, Клітинний Автомат, Конфігурація, Створіння, Менеджер суб’єкта, Лічильник кадрів,

Гра, Гра Життя, Хижак і здобич, Випадковість, Власник ресурсів, Менеджер ресурсів, Воїн, Всесвіт.

Ідентифіковані обов'язки:

1) Нова Мураха – додає та оновлює мурах у моделюванні;
2) Мураха – містить основну частину поведінки мурах для моделювання;
3) Додаток – основний проміжний клас для трьох типових моделей, що надає та передає можливість масштабування та відображення як КА для моделей «Життя», «Мурахи» та «Хижак і здобич»;
4) Клітинний Автомат – основний клас для задання відображення розміру та кольору на екрані клітин для трьох типових моделей;
5) Конфігурація – основний проміжний клас для збереження кольору фонів , розміру екрану та клітин;
6) Створіння – основний клас для моделі «хижак і здобич», що оголошує дані для руху, оновлення та зберігання основних даних архетипів;
7) Менеджер суб’єкта – основний клас пов’язаний із усіма діями у грі «Арена», маніпуляції із станом персонажів, їх окремими групами, тощо.;
8) Лічильник кадрів – клас-лічильник кількості кадрів в секунду для трьох типових моделей;
9) Гра Арена – головний файл для гри «Арена», що виконує основну дію запуску відображення та оновлення гри;
10) Гра Життя – задає основні дані, що необхідні моделюванню клітин у грі «Життя»;
11) Хижак і здобич – головний клас для гри «Хижак і Здобич», що зберігає та оновлює стан архетипів персонажів;
12) Випадковість – клас для формування випадкових значень: позицій, кольору, руху клітин у моделях КА;
13) Власник ресурсів – основний проміжний файл для збереження зовнішнього формату оформлення тексту та текстур;

14) Менеджер ресурсів – клас-помічник до класу «Власник ресурсів» що зберігає приклад подання шляху до форматів, перевіряє їх справжність/коректність та у випадку помилки задає формат тексту за замовчуванням;
15) Воїн – основний клас для гри «арена», що зберігає всі показники персонажів/клітин;
16) Всесвіт – містить основні параметри, дані, тощо., що необхідні для формування оточення гри «Арена»;

Атрибути та операції, необхідні для виконання обов'язків кожної сутності-класу наведемо у табл. 2.1.

Таблиця 2.1. – Сутності, атрибути та методи

Сутність	Атрибути	Методи
1	2	3
Нова Мураха	m_ants – масив, що зберігає мурах; m_cells – масив, що зберігає клітини;	input – при натисканні кнопки клавіатури – вводить нову мураху на екран; update – оновлює стан для кожної мурахи; addAnt – задає для мурах нові координати відносно поля; updateAnt – оновлює мураху та відносно поля;
Мураха	m_position – зберігає положення мурахи; m_direction – зберігає напрям руху мурахи; m_colour – змінна, що зберігає колір мурахи;	turn – встановлює напрям руху мурахи; translate – встановлення напряму відносно осі ординат; getPosition – викликає

		координати позиції для мурахи; setX – вибір позиції координати x;

Продовження таблиці 2.1

1	2	3
		setY – вибір позиції координати y; getColour – викликає колір мурахи;
Додаток	m_automaton – слугує вказівником на інший клас; m_window – змінна, що відповідає за дії із вікном; m_fpsCounter – слугує вказівником на інший клас; m_guiText – відповідає за збереження тексту, що виводиться на екран; m_view – відповідає за збереження даних для масштабування; m_pause – змінна зберігає одне значення, що відповідає за стан моделювання;	init – ініціалізує дані для типових моделей; run – запускає моделювання типових видів моделювання; getWindow – викликає вікно; pollEvents – надає можливість взаємодії із вікном моделювання; input – надає можливість рухати екран у процесі моделювання за траєкторією осі ординат; render – оновлює відображення усього інтерфейсу; resetView – повертає параметри масштабування до стандартних показників;

Клітинний Автомат	m_cellVertexPoints – зберігає масив даних, що зберігає дані положення клітини;	input – ініціює метод pollEvents; update – є відновником усіх інших методів update;

Продовження таблиці 2.1

1	2	3
		onRenderGUI – оновлює графічний інтерфейс; render – викликає оновлення клітин та графічного інтерфейсу; getCellIndex – отримує індекс клітини; setCellColour – встановлює колір клітини; cellForEach – виконує «дії» для кожної клітини у типових моделей; addQuad – додає квадрати/пікселі за основою розмірності клітин;
Конфігурація	windowSize – змінна, що зберігає значення двох параметрів, що відповідають за ширину та висоту вікна відображення; simSize – змінна для збереження двох значень, що відповідають за	init – розраховує окремо розмірності ширини та висоти екрану та клітини; GetInstans – виконує переадресацію на конструктор для ініціалізації

	ширину та висоту клітини; cellSize – зберігає значення одного параметру, що відповідає за розмір клітини; fps – зберігає значення, що слугує обмеженням кількості кадрів в секунду;	отриманих параметрів вікна та клітини;

Продовження таблиці 2.1

1	2	3
	WarriorsAtStart – зберігає одне значення, що відповідає за кількість персонажів на початок моделювання гри Арена; bgColour – зберігає три параметри, що відповідають за насиченість кольору заднього фону; fgColour – зберігає три параметри, що відповідають за насиченість кольору переднього фону;	
Створіння	MAX_HEALTH – зберігає один параметр, що відповідає за максимальне значення здоров'я; m_type – змінна, що зберігає тип клітини; m_health – зберігає один параметр, що відповідає за здоров'я клітини;	getColour – викликає колір за основою типу клітини та його здоров'я; getType – викликає тип клітини; setType – задає тип клітини; heal – встановлює зміну здоров'я;

		getHealth – викликає параметр здоров’я клітини; setHealth – задає параметр здоров’я клітині; move – оновлює здоров’я та тип клітини з кожною генерацією;

Продовження таблиці 2.1

1	2	3
		update – оновлює параметри здоров’я клітин; reproduce – за основою здоров’я клітини типу здобич створює їй подібну;
Менеджер суб’єкта	m_WarriorPositions – зберігає положення персонажів/клітин відносно поля; m_WarriorVertices – зберігає значення розмірів персонажів/клітин; m_livingWarriors – зберігає масив кількості живих персонажів m_warrior_buffer – слугує буфером для даних про розмірність та положення персонажів на полі; colonies – зберігає кількість груп персонажів/клітин на полі	Draw – відображає поле; attachtoWorld – заповнює поле відносно клітин; spawnWarriors – створює персонажів; Update – оновлює персонажів на полі; getNbrColonies – викликає кількість груп персонажів на полі; getNbrWarriors – викликає кількість персонажів на полі; updateColonies – оновлює

	p_world – допоміжна змінна, що слугує вказівником на інший клас; m_rnd – змінна що зв’язує клас із іншим класом.	групи персонажів; move – робить рух персонажів; isPositionValid – перевіряє позицію на доступність; populatePosition – заповнює позицію; clearPosition – очищує позицію;

Продовження таблиці 2.1

1	2	3
		clearAllDeadWarriors – очищає поле від мертвих персонажів; checkFriendly – перевірка сусіднього персонажа на відносність до групи; checkCollision – перевірка на зіткнення; killWarrior – вбиває персонажа; reproduce – репродукція персонажа-нащадка поряд із персонажем-предком; spawnChild – введення персонажа-нащадка на поле; isHeWinningFight – перевірка на результат поєдинку між персонажами;

		convertVectorToInt – перетворює вектор у ціле число; updateWarriors – оновлює стан персонажів; updateVertexArray – оновлює масив із розмірністю персонажів;

Продовження таблиці 2.1

1	2	3
		killSpecies – знищує групу персонажів; getPointerAtPosition – викликає вказівник на позицію;
Лічильник кадрів	m_text – змінна для зберігання параметрів тексту; m_delayTimer – лічильник затримки; m_fpsTimer – лічильник секунд; m_fps – зберігає одне значення, що слугує значенням кількості кадрів в секунду; m_frameCount – зберігає одне значення, що є лічильником кількості зображень;	update – оновлює лічильник кадрів в секунду; draw – відображає лічильник кадрів в секунду на екрані;
Гра Арена	world – допоміжна змінна, що слугує вказівником на інший	run – запускає процес моделювання;

	клас; entMgr – допоміжна змінна, що слугує вказівником на інший клас; view – змінна для зберігання параметрів для масштабування; window – відповідає за відображення та інших дій із вікном;	ViewReset – повертає стандартні показники масштабування екрану; render – оновлює інтерфейс із процесом моделювання; processEvents – надає можливість масштабувати екран у процесі моделювання;

Продовження таблиці 2.1

1	2	3
	Counter – зберігає параметрів тексту; fpsCounter – зберігає параметрів тексту; time_passed – змінна зберігає один параметр, що відповідає за пройдений час event_timer – змінна зберігає один параметр, що відповідає за лічильник часу події; pause – змінна зберігає одне значення, що відповідає за стан моделювання fps – змінна зберігає один параметр, що відповідає за лічильник кадрів в секунду; frameCount – змінна зберігає один параметр, що відповідає за	update – оновлює тест на екрані; fpsTextCounter – зберігає параметри тексту для відображення лічильників персонажів та кадрів в секунду;

	лічильник кількості кадрів; delayTimer – лічильник затримки; fpsTimer – лічильник секунд; rnd – змінна що зв'язує клас із іншим класом;	
Гра Життя	m_cells – відповідає за збереження параметрів та положення клітин	update – оновлює всі клітини;

Продовження таблиці 2.1

1	2	3
Хижак і здобич	m_creatures – зберігає масив створінь; m_preyCount – зберігає значення одного параметру, що відповідає за кількість здобич; m_predatorCount – зберігає значення одного параметру, що відповідає за кількість хижаків;	update – оновлює всі клітини на полі; updatePredator – оновлює клітини типу хижак; updatePrey – оновлює клітини типу здобич;
Випадковість	m_rng – зберігає випадкові значення;	get – отримує випадкове значення; Random – перенаправляє випадкове значення; intInRange – визначає випадкове значення типу int у певному діапазоні; floatInRange – визначає випадкове значення типу

		float у певному діапазоні; getMutation – перетворення персонажу за пошуком випадкового значення за типом int; getRandomColor – отримує випадковий колір; generateMovePos – генерує випадкове значення руху на полі;

Продовження таблиці 2.1

1	2	3
		getRandomPosition – отримує випадкову позицію на поле; reseed – перезапуск точки відправлення в генерації послідовності чисел;
Власник ресурсів	fonts – змінна, що слугує вказівником на інший клас; textures – змінна, що слугує вказівником на інший клас	get – отримує строку, що визначає шлях до файлу із файлом шрифту чи текстури;
Менеджер ресурсів	m_folder – зберігає назву шляху до файлу; m_extention – зберігає розширену назву шляху до файлу; m_resources – зберігає саму назву шуканого файлу; alive – змінна стану персонажу;	get – перевіряє на наявність отриманого шляху; exists – пошук файлу по заданому шляху; add – додання файлу із шрифтом чи текстурою за повним ім’ям шляху;

	m_positionInGrid – зберігає положення персонажа на полі; m_color – зберігає колір персонажу; m_strength – показник сили персонажа; m_age – показник віку персонажа; m_reproductionBonus – показник додаткової кількості відтворення персонажа;	getFullname – виклик повного шляху; update – оновлює показники персонажа; getStrength – отримує показник сили; canReproduce – дає змогу створювати нащадків; getMaxAge – отримує максимальне значення здоров'я персонажа;

Закінчення таблиці 2.1

1	2	3
Воїн	m_reproductionValue – лічильник відтворення персонажа; m_maxAge – показник віку персонажа;	kill – вбиває персонажа; getReprductionBonus – отримує додаткову змогу створення нащадків; isAlive – показник живого персонажа; getColor – отримує колір персонажа; setPosition – встановлює позицію; ExpermentalSetColor – експериментальне встановлення кольору персонажу заздалегідь; getPosition – отримує позицію персонажа;

Всесвіт	m_map – зберігає масив розмірності поля; m_tileSize – зберігає значення розмірності клітини поля; m_worldSizeInTiles – значення розмірності поля у клітинах; m_overlayGrid – змінна накладання сітки на поле; m_borderGrid – показник межі поля; m_drawGrid – змінна сітки поля;	toggleGrid – перемикач сітки; setdrawGrid – встановлення стану сітки; Draw – малює сітку поля; createMap – створення поля; getMap – виклик поля; m_BuildGrid – створює сітку поля;

2.2.1.2. Моделювання розподілу обов’язків у системі

Множини класів, які працюють спільно для досягнення деякої поведінки:

- Мураха, Нова Мураха– контроль створення та руху мурах;
- Додаток, Нова Мураха – ініціація та виведення мурах на поле;
- Додаток, Гра Життя – ініціація та виведення клітин на поле;
- Додаток, Хижак і здобич – ініціація та виведення клітин архетипів на поле;
- Додаток, Лічильник кадрів – передає лічильник кожній типовій моделі КА;
- Додаток, Клітинний Автомат – підстановка розмірів екрану і клітин для виконання дій до кожної клітини;
- Клітинний Автомат, Конфігурація – ініціює параметри вікна та клітини для кожної типової моделі КА;
- Клітинний Автомат, Нова Мураха – виконує дії для кожної мурахи;
- Клітинний Автомат, Гра Життя – виконує дії для кожної клітини;
- Клітинний Автомат, Хижак і здобич – виконує дії для кожної клітини архетипу;

- Створіння, Хижак і здобич – отримує та змінює показники архетипів клітин;
- Менеджер суб’єкта, Гра Арена – ініціює та оновлює інтерфейс та події у Гра Арена;
- Випадковість, Нова Мураха – отримання випадкового значення для координат місцеположення нової мурахи;
- Випадковість, Мураха – отримання випадкового кольору для мурахи;
- Випадковість, Створіння – отримання випадкового значення, що вплине на тип нової клітини;
- Випадковість, Хижак і здобич – отримання випадкового значення для координат місцеположення нової клітини;
- Випадковість, Менеджер суб’єкта – отримання випадкових значень: кольору, позиції руху на полі для клітин;
- Власник ресурсів, Додаток – задання шрифту лічильника номеру генерації у інтерфейс для типових моделей КА;
- Власник ресурсів, Лічильник кадрів – задання шрифту лічильника кадрів в секунду у інтерфейс для типових моделей КА;
- Власник ресурсів, Гра Арена – задання шрифту для лічильника кадрів в секунду та кількості персонажів у інтерфейс для Гри Арена;
- Менеджер ресурсів, Власник ресурсів – перевірка на існування та наявність шуканого файлу із шрифтом чи текстурою;
- Гра Арена, Конфігурація – ініціює параметри вікна та клітини для моделі Гра Арена;
- Воїн, Менеджер суб’єкта – отримує, виконує та змінює дії із персонажами у Гра Арена;
- Всесвіт, Менеджер суб’єкта – використовує дані про поле та його межі для взаємодії із персонажами у Гра Арена;

Визначення примітивних типів виконаємо на етапі побудови діаграми класів. Результат моделювання різних видів зв’язків подано у табл. 2.

Таблиця 2.2. – Моделювання залежностей

Клас, котрий зв’язується	Клас із котрим зв’язується	Тип зв’язку
1	2	3
Нова Мураха	Клітинний Автомат	Реалізація
	Мураха	Агрегація
	Конфігурація	Асоціація
	Випадковість	Асоціація
Мураха	Випадковість	Асоціація
Додаток	Лічильник кадрів	Композиція
	Клітинний Автомат	Агрегація
	Конфігурація	Асоціація
	Власник ресурсів	Асоціація

Закінчення таблиці 2.2

1	2	3
	Випадковість	Асоціація
Клітинний Автомат	Конфігурація	Асоціація
Конфігурація	-	-
Створіння	Випадковість	Асоціація
Менеджер суб’єкта	Всесвіт Воїн Випадковість	Композиція; Агрегація; Композиція
Лічильник кадрів	Власник ресурсів	Асоціація
Гра Арена	Всесвіт	Композиція
	Менеджер суб’єкта	Композиція
	Конфігурація	Асоціація
	Власник ресурсів	Асоціація
	Випадковість	Композиція
Гра Життя	Клітинний Автомат	Реалізація

	Конфігурація	Асоціація
Хижак і здобич	Клітинний Автомат	Реалізація
	Створіння	Агрегація
	Конфігурація	Асоціація
	Випадковість	Асоціація
Випадковість	-	-
Власник ресурсів	Менеджер ресурсів	Композиція
Менеджер ресурсів	-	-
Воїн	-	-
Всесвіт	-	-

2.2.1.3. Визначення призначень об’єктів за допомогою CRC карток

Класи можуть бути специфіковані у вигляді CRC-карток (Class - Responsibility- Collaboration).

У таблицях 2.3. – 2.18. наведено CRC-картки на класи проекту.

Таблиця 2.3. – CRC-картка для класу «Нова Мураха»

Базовий клас	Похідні класи (нащадки)
Клітинний Автомат	відсутні
Обов’язки	Зв’язки
Додає та оновлює мурах	Клітинний Автомат, Мураха, Конфігурація, Випадковість

Таблиця 2.4. – CRC-картка для класу «Мураха»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Відповідає за рух та колір мурах	Випадковість

Таблиця 2.5. – CRC-картка для класу «Додаток»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні

Обов’язки	Зв’язки
Ініціює, моделює та надає контроль над моделюванням типових КА.	Лічильник кадрів, Клітинний Автомат, Конфігурація, Власник ресурсів, Випадковість

Таблиця 2.6. – CRC-картка для класу «Клітинний автомат»

Базовий клас	Похідні класи (нащадки)
відсутні	Нова Мураха, Гра Життя, Хижак і здобич
Обов’язки	Зв’язки
Резервує, додає та оновлює усі клітини типових КА	Конфігурація

Таблиця 2.7. – CRC-картка для класу «Конфігурація»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Зберігає параметри вхідних даних користувача, та параметри за замовчуванням.	відсутні

Таблиця 2.8. – CRC-картка для класу «Створіння»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Отримує, встановлює та оновляє типи, колір та стан клітин архетипів	Конфігурація

Таблиця 2.9. – CRC-картка для класу «Менеджер суб’єкта»

Базовий клас	Похідні класи (нащадки)

відсутні	відсутні
Обов’язки	Зв’язки
Виконує всі основні дії із персонажами, оновлює та виводить їх на поле	Всесвіт, Воїн, Випадковість

Таблиця 2.10. – CRC-картка для класу «Лічильник кадрів»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Задає, оновлює та виводить лічильник у типових моделях КА	Власник ресурсів

Таблиця 2.11. – CRC-картка для класу «Гра Арена»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Задає, моделює, оновлює та контролює моделювання гри Арена.	Всесвіт, Менеджер суб’єкта, Конфігурація, Власник ресурсів, Випадковість

Таблиця 2.12. – CRC-картка для класу «Гра Життя»

Базовий клас	Похідні класи (нащадки)
Клітинний автомат	відсутні
Обов’язки	Зв’язки
Задає та оновлює клітини	Клітинний Автомат, Конфігурація

Таблиця 2.13. – CRC-картка для класу «Хижак і здобич»

Базовий клас	Похідні класи (нащадки)

Клітинний автомат	відсутні
Обов’язки	Зв’язки
Створює та оновлює стан обох архетипів клітин	Клітинний Автомат, Створіння, Конфігурація, Випадковість

Таблиця 2.14. – CRC-картка для класу «Випадковість»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Генерує випадкові показники у натуральних та цілих значеннях	відсутні

Таблиця 2.15. – CRC-картка для класу «Власник ресурсів»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Отримує шлях до шрифту та текстур перевіряючи формат шляху	Менеджер ресурсів

Таблиця 2.16. – CRC-картка для класу «Менеджер ресурсів»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Перевіряє шлях та впроваджує шрифт та текстуру	відсутні

Таблиця 2.17. – CRC-картка для класу «Воїн»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Отримує всі показники персонажів та оновлює їх стан	відсутні

Таблиця 2.18. – CRC-картка для класу «Всесвіт»

Базовий клас	Похідні класи (нащадки)
відсутні	відсутні
Обов’язки	Зв’язки
Зберігає дані створення та задання поля і його сітки	відсутні

2.2.1.4. Побудова об’єктної моделі (діаграми класів)

Заключним результатом моделювання представимо у вигляді діаграми класів (рис. 2.2).

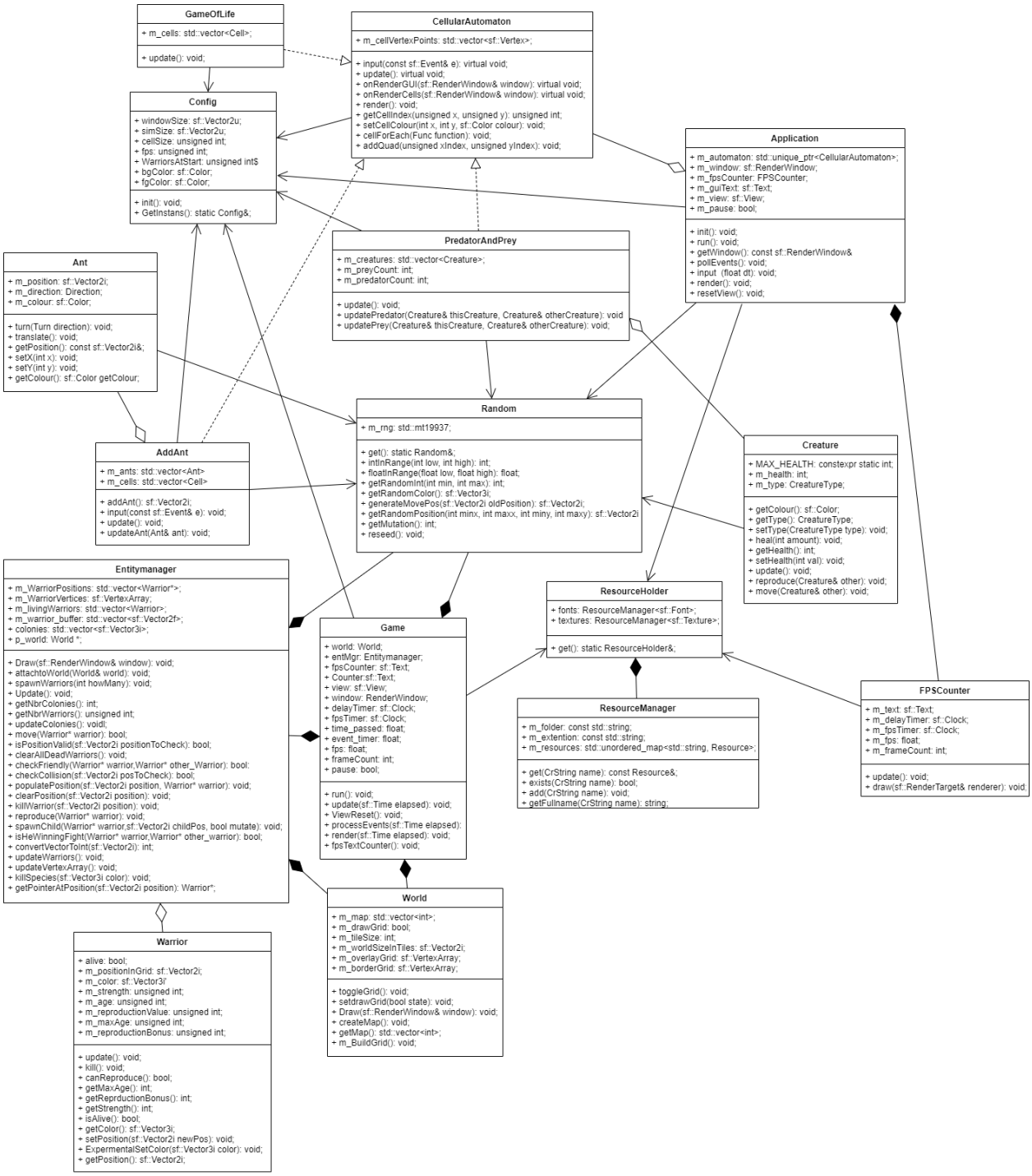


Рисунок 2.2 – Діаграма класів

Класи пов’язані агрегацією: AddAnt та Ant, Application та CellularAutomaton, PredatroAndPrey та Creature, EntityManager та Warrior, тому що другий клас є частиною/складовою першого класу.

Класи пов’язані композицією: EntityManager та Random, EntityManager та World, Game та EntityManager, Game та Random, Game та World, крім зв’язку як частина класу, їх час існування співпадає.

Класи пов’язані реалізацією: GameOfLife та CellularAutomaton, AddAnt та CellularAutomaton, PredatorAndPrey та CellularAutomaton, тому що реалізують для кожного різновиду відповідні методи.

Класи пов’язані асоціація: GameOfLife та Config, CellularAutomaton та Config, Application та Config, PredatroAndPrey та Config, AddAnt та Config, Game та Config,Ant та Random,AddAnt та Random, PredatorAndPrey та Random, Application та Random, Creature та Random, Game та ResourceHolder, Application та ResourceHolder, FPSCounter та ResourceHolder, використовуються для виконання певних дій.

2.2.2. Проектування інтерфейсу користувача

2.2.2.1. Створення ескізів форм

Посилаючись на діаграму прецедентів [пункт 2.1.5] можна виділити окремі форми для надання користувачу можливості виконувати наступні дії:

– спостерігати процес моделювання;
– попередньо задавати параметри відображення;
– обирати нову модель КА.

Користувач може мати доступ швидкого виконання дій. Тому головна форма повинна мати наступне меню:

– «Життя»;
– «Мурахи»;
– «Хижак і здобич»;
– «Арена»;
– «Нотатки»;
– Закрити програму.

Таким чином програмний продукт складається з наступних вікон:

– головне вікно (рис. 3.1);
– вікно візуалізації (рис. 3.2);
– вікно з довідкою (рис. 3.3).

Головне вікно програми (рис. 3.1) має шість пунктів меню, чотири з яких надають користувачу можливість швидко доступу до моделей КА. П’ятий

пункт відповідає за відкриття довідки. Шостий пункт відповідає за завершення роботи програми.

Кожна форма, крім головного вікна та довідки, тобто моделювання КА мають декілька кнопок для керування візуалізацією моделювання, а саме: масштабуванням, рух зони візуалізації відносно вікна, зупинка/відновлення моделювання та закриття форми. Взаємодія реалізується за допомогою миші та клавіатури, призначення клавіш керування наступне:

- рух зони візуалізації відносно вікна відповідають кнопки: «W» (вгору) «A»(ліворуч) «S»(вниз) «D»(праворуч);
- масштабування: «стрілка вгору» (приближення), «стрілка вниз» (віддалення);
- зупинка та відновлення моделювання: «Space»(зупинка) та «G»(відновлення);
- завершення моделювання: кнопка «ESC» чи хрест у правому верхньому кутку екрану форми.

Окрім стандартної взаємодії миші із меню, є окрема взаємодія комбінуванням клавіш клавіатури:

при натисканні на «1-4»(номер бажаної моделі) – відображення опису відповідної моделі;

- «SHIFT» + «1-4»(номер бажаної моделі) – запуск відповідної моделі;
- «SHIFT» + «TAB» – відкриття довідки;
- «SHIFT» + «Q» – завершення роботи програми.

При наведенні курсору миші на відповідну назву моделі КА, чи при натисканні відповідної комбінації клавіш, у правому верхньому боці екрану з’явиться короткий опис відповідної моделі (рис. 3.2).

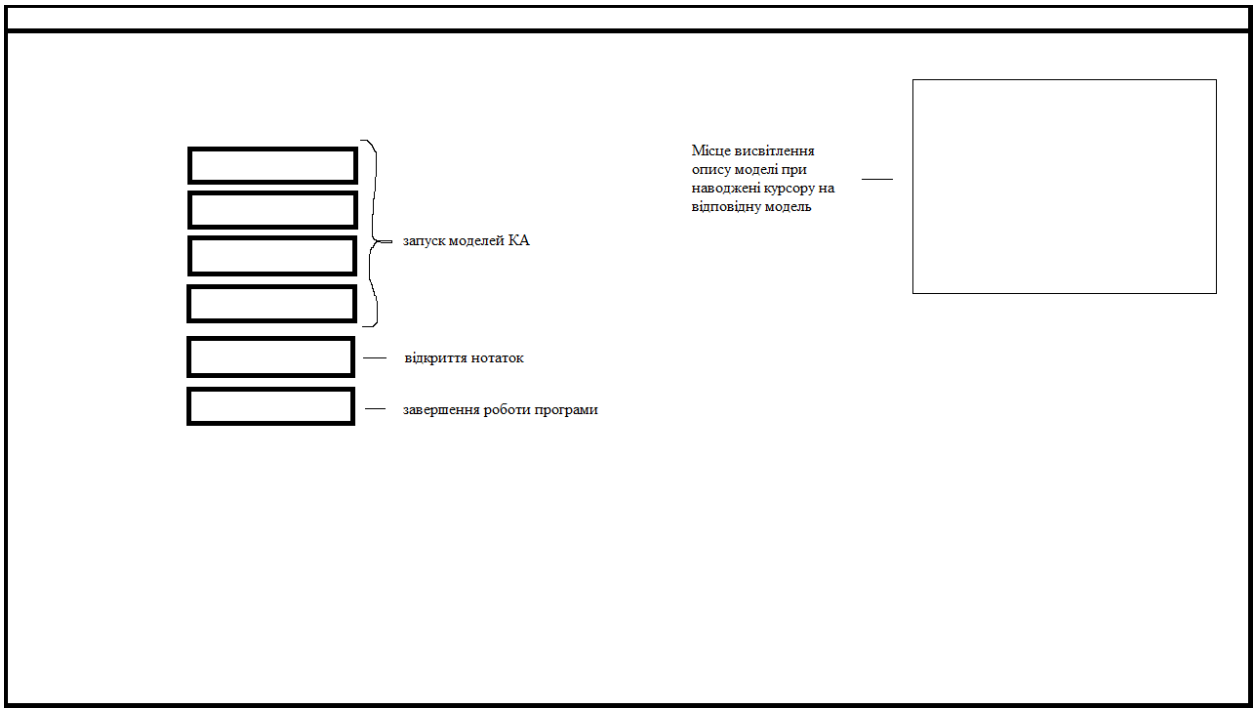


Рисунок 3.1 – Ескіз головного вікна програми

На формі буде зображена напис із короткою довідкою про можливості взаємодії у процесі моделювання із використанням клавіатури, та можливості у певних моделях КА (рис. 3.2).

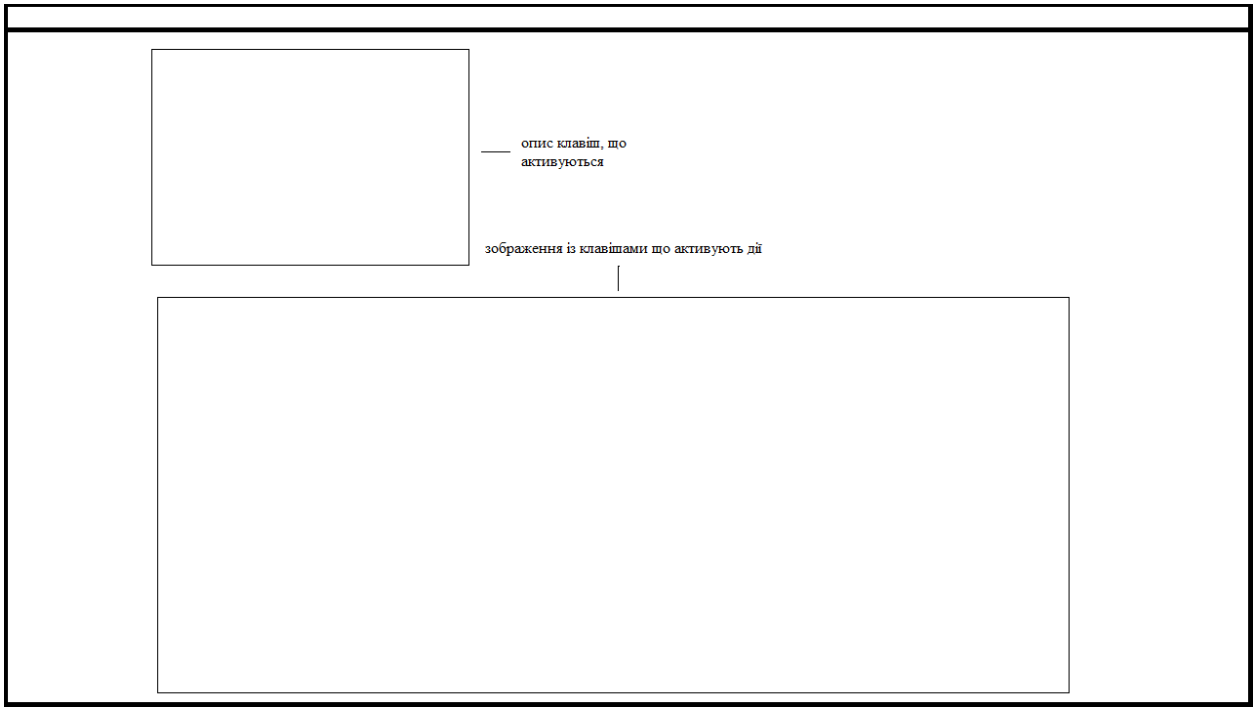


Рисунок 3.2 – Ескіз вікна довідка

Вікно візуального відображення моделювання зображено на (рис. 3.4), котре суцільно містить графічне поле КА. Може надаватися можливість масштабування графічного поля чи запуск/зупинка моделювання.

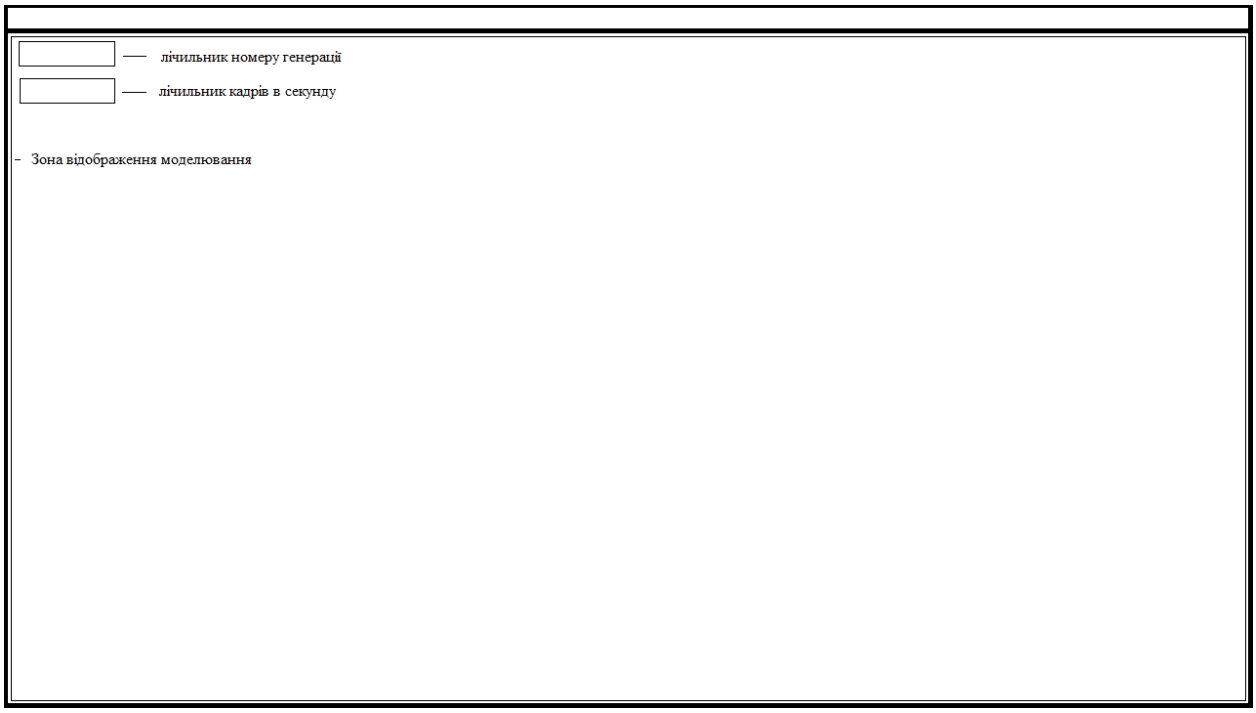


Рисунок 3.3 – Ескіз вікна візуального відображення моделювання

2.2.3. Проектування динаміки системи

В якості об'єктів виступають користувач, котрий ініціює взаємодію, та класи, що володіють поведінкою в системі.

Посилаючись на діаграму прецедентів [пункт 2.1.5] у зв’язку із зображенням екземплярів об'єктів та повідомлень якими на діаграмі послідовності вони обмінюються у рамках одного прецеденту, побудуємо дві діаграми послідовності для варіантів використання:

- задання параметрів відображення;
- користувач контролює моделювання.

Сценарій «задання параметрів відображення»: спочатку користувач за бажанням у файлі «config.txt» змінює значення під відповідною назвою параметру. Всі параметри КА зберігаються у окремому об’єкті (config), який також зберігає дані за замовченням якщо користувач вирішить не вносити

зміни чи файл не вдасться відкрити. Діаграма послідовності представлена на рис. 3.5.

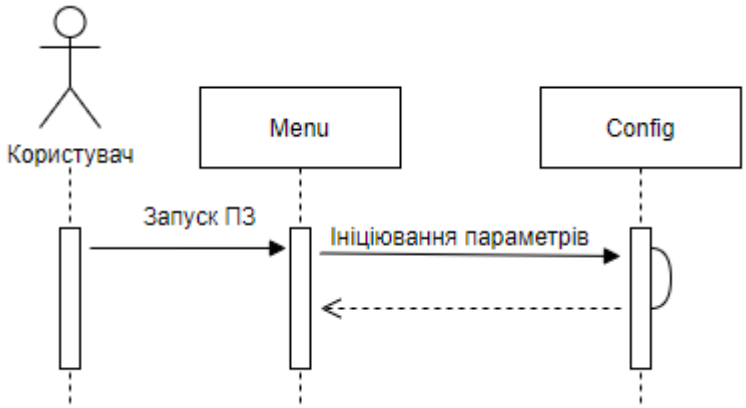


Рисунок 3.5. – Діаграма послідовності для варіанту використання «задання параметрів відображення»

Сценарій «Користувач контролює моделювання»: користувач запускає процес моделювання і контролює його:

– рух зони візуалізації відносно вікна відповідають кнопки: «W» (рух вгору) «A»(рух ліворуч) «S»(рух вниз) «D»(рух праворуч);
– масштабування: «стрілка вгору» (приближення), «стрілка вниз» (віддалення);
– зупинка та відновлення моделювання: «Space»(зупинка) та «G»(відновлення);
– завершення моделювання: кнопка «ESC» чи хрест у правому верхньому кутку екрану форми.

У відповідь на ці дії користувач отримує візуальне відображення КА і протікання процесу. Діаграма послідовності представлена на рис. 3.6.

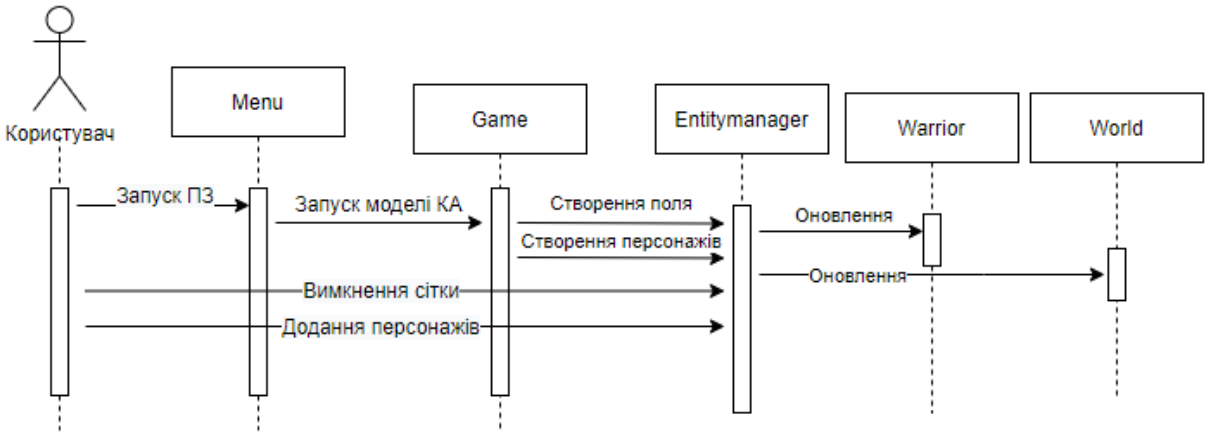


Рисунок 3.6. – Діаграма послідовності для варіанту використання «користувач контролює моделювання»

2.2.4. Вибір мови програмування

Для створення ПЗ була використана мова програмування C++. Мова C++ багато в чому є надмножиною C. Вона додає до C об'єктно-орієнтовані можливості. Він вводить класи, які забезпечують три найважливіші властивості ООП[11]: інкапсуляцію, успадкування і поліморфізм.

При створенні C++ прагнули зберегти сумісність з мовою C, а більшість програм на C справно працюватимуть і з компілятором C++. C++ має синтаксис, заснований на синтаксисі C[11]. Тобто розробники, що можуть зацікавитися ПЗ розробленій на C++, але котрі використовують C, зможуть продовжити чи переробити ПЗ під себе, тобто за своїм бажанням.

Для відображення інтерфейсу була використана додаткова бібліотека «SFML». Бібліотека SFML надає простий інтерфейс для різних компонентів нашого комп'ютера, щоб полегшити розробку ігор та мультимедійних програм. Вона складається з п'яти модулів: system, window, graphics, audio та network.

Використовуючи SFML, наш додаток може бути скомпільований та запущений на найбільш поширених платформах: Windows, Linux, Mac OS X (а ще у розробці підтримка Android та IOS), також попередньо скомпільовані SDK для нашого ОС вже доступні на сторінці завантаження.

SFML офіційно підтримує C та .NET. Також, завдяки своїй активній спільноті, вона доступна багатьма іншими мовами, такими як Java, Ruby, Python, Go та ін.

Висновки до пункту 2.

На стадії проектування системи були розроблені ескізи інтерфейсу користувача, спроектовані основні діаграми для майбутнього програмного засобу та обрана мова проектування. На основі розроблених у цьому розділі деталей проектування програмного засобу, був написаний основний код програми, який має пройти тестування.

3. ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

3.1. Тестування методом «білої скриньки»

Для тестування програми методом білої скриньки вхідними даними є текст програми та специфікації функції. Таким чином далі надаватимуться:

– текст методу (який тестується);
– специфікація цього методу;
– тестування методом покриття рішень;
– тестування методом покриття умов.

Метод 1: Move.

Специфікація: дана функція є обробником події, що виникає при кожній ітерації моделювання. Функція виконує зміну позиції положення для усіх персонажів на полі.

Вхідні дані:

у явному виді – warrior (вказівник на об’єкт класу персонажа).

у не явному – m_rnd (об’єкт класу, що повертає випадкові значення).

Вихідні дані:

у явному виді – повертає результат виконаної операції руху персонажа.

у не явному виді:

– warrior (змінені стан та положення персонажа);
– bool (показник здатності руху персонажа).

Текст методу:

```
bool Entitymanager::move(Warrior* warrior) {
   sf::Vector2i oldpos= warrior->getPosition();
   sf::Vector2i newPos = m_rnd.generateMovePos(oldpos);
   if(isPositionValid(newPos))// 1
   {
      if(checkCollision(newPos))// 2
      {
         Warrior* warriorAtPosition =getPointerAtPosition(newPos);
```

```
            if(!checkFriendly(warrior,warriorAtPosition))// 3
            {
               clearPosition(oldpos);
               if(!isHeWinningFight(warrior,warriorAtPosition))// 4
               {
                  warrior->kill();
                  return false;
               }
               else
               {
                  warrior->setPosition(newPos);
                  killWarrior(newPos);
                  populatePosition(newPos, warrior);
                  return true;
               }
            }
            return false;
         }
         else
         {
            warrior->setPosition(newPos);
            clearPosition(oldpos);
            populatePosition(newPos, warrior);
            return true;
         }
      }
      return false;
   }
```

Результати тестування методом покриття рішень надані у табл. 4.1. Метод містить 4 рішення: isPositionValid(newPos), checkCollision(newPos),

!checkFriendly(warrior,warriorAtPosition),
!isHeWinningFight(warrior,warriorAtPosition).

Для покриття рішення isPositionValid(newPos) необхідно зробити тестування методу isPositionValid, який є методом класу Entitymanager. isPositionValid визначає, чи дозволена передана позиція для переходу, тобто чи не є вона межею.

Вхідні дані:

у явному виді – positionToCheck (значення випадкової позиції для переміщення персонажа).

у не явному виді – p_world (вказівник на об’єкт класу персонажа).

Вихідні дані:

у не явному виді – bool (перевірене значення відсутності зіткнення із межею).

Розробка тестів:

1. Персонаж того самого кольору

Вхідні дані:

– positionToCheck (значення випадкової позиції для переміщення персонажа).
– p_world (поточна позиція персонажа).

Вихідні дані – true (зіткнення із межею немає).

2. Персонаж іншого кольору

Вхідні дані:

– positionToCheck (значення випадкової позиції для переміщення персонажа).
– p_world (розмір поля).

Вихідні дані – false (зіткнення із межею).

Для покриття рішення checkCollision (newPos) необхідно зробити тестування методу checkCollision, який є методом класу Entitymanager. checkCollision визначає, чи є зіткнення на переданій позиції для переходу із іншим персонажем.

Вхідні дані:

у явному виді – posToCheck (значення випадкової позиції для переміщення персонажа).

у не явному виді – m_WarriorPositions (вказівник на об’єкт класу персонажа).

Вихідні дані:

у не явному виді – bool (перевірене значення зіткнення персонажа).

Розробка тестів:

1. Позиція вільна для руху

Вхідні дані:

– posToCheck (значення випадкової позиції для переміщення персонажа).
– m_WarriorPositions (поточна позиція персонажа).

Вихідні дані – true (позиція вільна).

2. Позиція зайнята іншим персонажем

Вхідні дані:

– posToCheck (значення випадкової позиції для переміщення персонажа).
– m_WarriorPositions (поточна позиція персонажа).

Вихідні дані – false (позиція зайнята).

Для покриття рішення !checkFriendly(warrior,warriorAtPosition) необхідно зробити тестування методу checkFriendly, який є методом класу Entitymanager. checkFriendly визначає, чи є персонаж із яким вийшло зіткнення другом.

Вхідні дані:

у не явному виді:

– warrior (вказівник на об’єкт класу персонажа);
– other_warrior (вказівник на об’єкт класу персонажа).

Вихідні дані:

у не явному виді – bool (перевірене значення кольору персонажа).

Розробка тестів:

1. Персонаж того самого кольору

Вхідні дані:

– warrior (показник кольору персонажа);

– other_warrior (показник кольору іншого персонажа).

Вихідні дані – true (персонаж друг).

2. Персонаж іншого кольору

Вхідні дані:

– warrior (показник кольору персонажа);

– other_warrior (показник кольору іншого персонажа).

Вихідні дані – false (персонаж ворог).

Для покриття рішення !isHeWinningFight(warrior,warriorAtPosition) необхідно зробити тестування методу isHeWinningFight, який є методом класу Entitymanager. isHeWinningFight визначає, чи виграв персонаж бій при зіткненні із ворожим персонажем.

Вхідні дані:

у не явному виді:

– warrior (вказівник на об’єкт класу персонажа).

– other_warrior (вказівник на об’єкт класу персонажа).

Вихідні дані:

у не явному виді – bool (перевірене значення параметрів персонажа).

Розробка тестів:

1. Персонаж виграв бій

Вхідні дані:

– warrior (значення сили персонажа);

– other_warrior (показник сили іншого персонажа).

Вихідні дані – true (персонаж виграв).

2. Персонаж програв бій

Вхідні дані:

– warrior (значення сили персонажа);

– other_warrior (показник сили іншого персонажа).

Вихідні дані – false (персонаж програв).

Розробка тестів методу move:

1. Позиція зайнята межею

Вхідні дані – m_rnd (координати позиції межі).

Вихідні дані:

– warrior (персонаж не змінив позицію);

– bool – false (позиція зайнята межею).

2. Позиція не зайнята іншим персонажем

Вхідні дані – m_rnd (координати не зайнятої позиції).

Вихідні дані:

– warrior (персонаж займає нову позицію);

– bool – true (позиція вільна).

3. Позиція зайнята іншим дружелюбним персонажем

Вхідні дані – m_rnd (координати позиції зайняті дружелюбним персонажем).

Вихідні дані:

– warrior (персонаж не змінив позицію);

– bool – false (позиція зайнята дружелюбним персонажем).

4. Позиція зайнята іншим ворожим персонажем, результат бою – перемога

Вхідні дані – m_rnd (координати позиції зайняті ворожим персонажем).

Вихідні дані:

– warrior (персонаж виграв бій та зайняв позицію виграного персонажа).

– bool – true (позиція звільнена після бою).

5. Позиція зайнята іншим ворожим персонажем, результат бою – програш

Вхідні дані – m_rnd (координати позиції зайняті ворожим персонажем).

Вихідні дані:

– warrior (персонаж програв бій та знищився).

– bool – false (позиція зайнята, персонаж вмер).

Таблиця 4.1. – Покриття умов та рішень

Номер тесту	Номер рішення			
	1	2	3	4
1	-			
2	+	-		
3	+	+	-	
4	+	+	+	-
5	+	+	+	+

Таким чином, для покриття усіх можливих умов та рішень методу «Move» потрібно усього 5 тестів.

Метод 2: updateWarriors.

Специфікація: дана функція є обробником події, що також виникає при кожній ітерації моделювання. Функція виконує оновлення стану для усіх персонажів на полі.

Вхідні дані:

у не явному виді – warrior (об’єкт класу, що викликає параметри персонажу).

Вихідні дані:

у не явному виді – warrior (оновлені персонажі).

Результати тестування методом покриття рішень надані у табл. 4.2. Метод містить 2 умови: warrior.isAlive(), move(&warrior) && warrior.canReproduce().

Текст методу:

```
void Entitymanager::updateWarriors() {
   for(auto& warrior:m_livingWarriors)
```

```
    {
      warrior.update();
      if(warrior.isAlive()) // 1
      {
         if(move(&warrior)&& warrior.canReproduce()) // 2 (2, 3)
         {
            reproduce(&warrior);
         }
      }
    }
   clearAllDeadWarriors();
}
```

Для покриття умови warrior.isAlive() необхідно зробити тестування методу isAlive, який є методом класу Warrior. isAlive() визначає стан персонажа.

Вхідні дані:

у не явному – alive (значення стану персонажа).

Вихідні дані:

у не явному виді – alive (перевірене значення стану персонажа).

Розробка тестів:

1. Персонаж живий

Вхідні дані – alive (значення стану персонажа).

Вихідні дані – true (персонаж живий).

2. Персонаж мертвий

Вхідні дані – alive (значення стану персонажа).

Вихідні дані – false (персонаж мертвий).

Для покриття умови move(&warrior) необхідно зробити тестування методу move, який є методом класу Entitymanager. move визначає, чи змінить персонаж позицію.

Вхідні дані:

у явному виді – warrior (вказівник на об’єкт класу персонажа).

у не явному – m_rnd (об’єкт класу, що повертає випадкові значення).

Вихідні дані:

у явному виді – повертає результат виконаної операції руху персонажа.

у не явному виді – warrior (змінені стан та положення персонажа).

Розробка тестів:

1. Зміна позиції можлива

Вхідні дані:

– warrior (вказівник на об’єкт класу персонажа);
– m_rnd (об’єкт класу, що повертає випадкові значення).

Вихідні дані – true (рух можливий).

2. Зміна позиції не можлива

Вхідні дані:

– warrior (вказівник на об’єкт класу персонажа);
– m_rnd (об’єкт класу, що повертає випадкові значення).

Вихідні дані – false (рух не можливий).

Для покриття умови warrior.canReproduce() необхідно зробити тестування методу canReproduce, який є методом класу Warrior. canReproduce визначає, чи має персонаж змогу репродукувати.

Вхідні дані:

у не явному виді – m_reproductionValue.

Вихідні дані:

у явному виді – bool значення true чи false.

Розробка тестів:

1. Репродукція можлива

Вхідні дані – m_reproductionValue (значення здатності до репродукції).

Вихідні дані – true (здатність присутня).

2. Репродукція не можлива

Вхідні дані – m_reproductionValue (значення здатності до репродукції).

Вихідні дані – false (здатність відсутня).

Розробка тестів методу updateWarriors:

1. Немає живих персонажів

Вхідні дані – warrior (стан персонажу).

Вихідні дані – warrior (перевірений стан персонажа).

2. Персонаж живий, але не може рухатися чи не має змоги до репродукції

Вхідні дані – warrior (стан персонажу та його спроможність рухатися та змоги до репродукції нащадка).

Вихідні дані – warrior (перевірений стан, змога рухатися та змогу до репродукції нащадка персонажа).

3. Персонаж живий, може рухатися та репродукції нащадків

Вхідні дані – warrior (стан персонажу та його спроможність рухатися та змогу до репродукції нащадка).

Вихідні дані – warrior (оновлений стан та створений нащадок персонажа).

Таблиця 4.2. – Покриття рішень

Номер тесту	Номер рішення	
	1	2
1	-	
2	+	-
3	+	+

Таким чином, для покриття усіх можливих рішень методу «updateWarriors» потрібно усього три тести.

Результатом тестування методу «updateWarriors» методом покриття умов надані у табл. 4.3.

Метод містить 3 умови: warrior.isAlive(), move(&warrior), warrior.canReproduce().

Розробка тестів методу updateWarriors:

1. Немає живих персонажів

Вхідні дані – warrior (стан персонажу).

Вихідні дані – warrior (перевірений стан персонажа).

2. Персонаж живий, може рухатися, але не має змоги до репродукції

Вхідні наді – warrior (стан персонажу та його спроможність рухатися та змогу до репродукції нащадка).

Вихідні дані – warrior (оновлений стан та позиція персонажа).

3. Персонаж живий, не може рухатися, але має змогу до репродукції

Вхідні наді – warrior (стан персонажу та його спроможність рухатися та змогу до репродукції нащадка).

Вихідні дані – warrior (оновлений стан та створений нащадок персонажа).

4. Персонаж живий, може рухатися та має змогу до репродукції нащадків

Вхідні дані – warrior (стан персонажу та його спроможність рухатися та має змогу до репродукції нащадка).

Вихідні дані – warrior (оновлений стан, позиція та створений нащадок персонажа).

Таблиця 4.3. – Покриття умов

Номер тесту	Номер умов		
	1	2	3
1	-		
2	+	+	-
3	+	-	+
4	+	+	+

Таким чином, для покриття усіх можливих умов методу «updateWarriors» потрібно вже чотири тести.

3.2. Тестування методом «чорної скриньки»

Метод 1: Move.

Специфікація: дана функція є обробником події, що виникає при кожній ітерації моделювання. Функція виконує зміну позиції положення для усіх персонажів на полі.

Вхідні дані:

у явному виді – warrior (вказівник на об’єкт класу персонажа).

у не явному – m_rnd (об’єкт класу, що повертає випадкові значення).

Вихідні дані:

у явному виді – повертає результат виконаної операції руху персонажа.

у не явному виді:

– warrior (оновлені параметри та положення персонажа);
– bool (показник здатності руху персонажа).

Класи метода еквівалентного розбиття представлені у табл. 4.4.

Таблиця 4.4. – Класи еквівалентності

Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності
При обробці руху, умова повертає	Відповідне bool значення (1)	Не відповідне bool значення (2,3)
Вказівник на об’єкт класу персонажа повертає	Оновлені параметри та позиція персонажа (2)	Незмінені параметри та позиція персонажа (3)

Розробка тестів методу еквівалентного розбиття:

1. Персонаж займає іншу позицію

Вхідні дані – warrior (параметри та позиція персонажа).

Вихідні дані:

– warrior (оновлені параметри та позиція персонажа);
– bool – true (персонаж займає іншу позицію).

2. Персонаж не може рухатись

Вхідні дані – warrior (параметри та позиція персонажа).

Вихідні дані:

– warrior (оновлені параметри та позиція персонажа);

– bool – true (персонаж займає іншу позицію).

3. Персонаж програв бій

Вхідні дані – warrior (параметри та позиція персонажа).

Вихідні дані:

– warrior (незмінені параметри та позиція персонажа);

– bool – true (персонаж виграв бій).

Список усіх припущень про помилку:

– не повертає очікуване значення bool при перевірці на наявність межі на позиції переходу;

– у повернених параметрах персонажа не відбулися належні зміни.

Припущення 1: не повертає очікуване значення bool при true перевірці на наявність межі на позиції руху.

Вхідні дані – warrior (параметри та позиція персонажа).

Вихідні дані:

– warrior (оновлені параметри та позиція персонажа);

– bool – false (персонаж змінює позицію).

Аналіз результатів: програма де буде працювати, оскільки некоректно повернене значення перевірки.

Припущення 2: у повернених параметрах персонажа не відбулися належні зміни.

Вхідні дані – warrior (параметри та позиція персонажа).

Вихідні дані:

– warrior (ті самі параметри та позиція персонажа);

– bool – true (персонаж виграв бій та змінює позицію).

Аналіз результатів: програма де буде працювати, оскільки не відбулися очікувані зміни параметри та позиції персонажа.

Метод 2: updateWarriors.

Специфікація: дана функція є обробником події, що також виникає при кожній ітерації моделювання. Функція виконує оновлення стану для усіх персонажів на полі.

Вхідні дані:

у не явному виді – warrior (об’єкт класу, що викликає параметри персонажу).

Вихідні дані:

у не явному виді – warrior (оновлені персонажі).

Класи метода еквівалентного розбиття представлені у табл. 4.5.

Таблиця 4.5. – Класи еквівалентності

Вхідні умови	Правильні класи еквівалентності	Неправильні класи еквівалентності
При поверненні об’єктом класу значень персонажа	Правильне повернення: стан, спроможність руху та змогу репродукції (1)	Хоча б одне із значень повернено невірно (2, 3)
Персонаж при перевірці на спроможність робити репродукцію, виконується	Відповідно із обома умовами (2)	Хоча б одна з умов не виконана (1, 3)

Розробка тестів методу еквівалентного розбиття:

1. Персонаж живий, може рухатися але не має змоги до репродукції нащадків.

Вхідні дані – warrior (стан персонажу та його спроможність рухатися та змогу до репродукції нащадка).

Вихідні дані – warrior (перевірений стан, змога рухатися).

2. Персонаж живий, може рухатися та має змогу до репродукції нащадків.

Вхідні дані – warrior – (стан персонажу та його спроможність рухатися та змогу до репродукції нащадка).

Вихідні дані – warrior (перевірений стан, змога до репродукції нащадка персонажа).

3. Персонаж мертвий.

Вхідні дані – warrior (стан персонажу та його спроможність рухатися та змогу до репродукції нащадка).

Вихідні дані – warrior (перевірений стан та змога рухатися).

Список усіх припущень про помилку:

– невірно повертає значення при оновлені;
– репродукція виконується при відсутності одної з умов.

Припущення 1: невірно повертає значення при оновлені.

Вхідні дані – warrior (стан персонажу та його спроможність рухатися та змогу до репродукції нащадка).

Вихідні дані – warrior (спроможність рухатися та змогу до репродукції нащадка).

Аналіз результатів: оновлення не буде виконано. Треба прослідкувати, щоб значення співвідносилися.

Припущення 2: репродукція виконується при відсутності одної з умов.

Вхідні дані – warrior (стан персонажу та його спроможність рухатися).

Вихідні дані – warrior (стан персонажу та його спроможність рухатися та змогу до репродукції нащадка).

Аналіз результатів: треба прослідкувати, щоб умови були правильно пов’язані між собою.

Висновки до пункту 3.

При тестуванні, не було виявлено помилок, але основна функція інтерфейсу «menu», що викликала моделі – на третину перевищувала стандартний показник 16 Mb для аналізу кода. Для вирішення цієї проблеми, достатньо було виділити відповідну кількість байт на стек у налаштуваннях

проекту. Також, однією із виявлених проблем була спроба додати параметри насиченості переднього/заднього фонів, що призводило до неправильної передачі формату даних. Під час спроби виправлення, після довгих спроб зрозуміти, як саме передаються значення, пошук вирішення проблеми закінчився невдачею, що призвело до рішення, прибрати цю функцію із наданих користувачеві можливостей впливу на моделювання.

ВИСНОВОК

На даний час існує дуже багато найрізноманітніших програмних засобів, що надаюсь можливість використовувати клітинні автомати для моделювання певних процесів. В розпорядженні користувача є універсальні інструменти, що дозволяють будувати клітині автомати за потребами користувача, але такий інструментарій потребує часу для вивчення його можливостей та вміння формування правил функціонування КА. В розробці увага сконцентрована на візуалізації роботи певних моделей, які відповідають обраній тематиці. Такий спосіб моделювання має свої недоліки і переваги. З одного боку, досить прості правила формування поведінки окремого персонажа, або взаємодії різних типів персонажів, з іншого – витратна складова з точки зору обсягів пам’яті на конкретну модель, оскільки на неї впливає розмір матриці моделювання і завантаженість атрибутами конкретної клітини.

Результатом проектування є проект програмного засобу, для моделювання певних процесів клітинними автоматами. Розроблений програмний продукт надає можливість моделювати такі типові системи як «хижак-здобич», гра «життя», «мурахи» та оригінальну модель: гру «Арена». Все що потрібно від користувача – це вибрати певну модель що описує процес який зацікавив користувача.

Під час моделювання, кожна ітерація вираховується за особливими подіями, тобто випадає випадковий показник події (як наприклад руху) і відповідно із ним вираховується новий етап еволюції клітинного автомату. У програмі використовується рівномірний закон розподілу як для підрахунку модельного часу, так і для моделювання ймовірнісного клітинного автомату.

В процесі роботи отримуємо візуальне відображення еволюції клітинного автомата, статистичні данні. Під статистикою розуміється: кількість кадрів в секунду, що дає поняття завантаженості процесу, кількісний вміст клітин на полі клітинного автомату чи/та етап генерації у певний момент часу.

Розглянемо в якості прикладу результати використання четвертої моделі. Існує декілька стандартних конфігурацій, але дозволяється впливати на розвиток моделі шляхом додавання персонажів, які випадковим чином розміщуються на полі. Місце їх появи може вплинути на поведінку інших в трьох напрямках. Перший – поряд, декілька персонажів, що з’являться будуть одного типу та в них буде шанс на утворення свого власного об’єднання, другий – окремо, персонажі будуть розкидані по одинці на всьому полі, що призведе до досить швидкої смерті, третій – серед ворогів, персонажі з’являються вже в чомусь об’єднанні то вони стають лише сировиною для зміцнення відповідної групи.

Таким чином кінцевий користувач ПЗ отримує програмний продукт, який надає можливість моделювати різні за напрямами ігрового дослідження еволюції клітинних автоматів.

ЛІТЕРАТУРА

1. Карпов Ю. Г. Имитационное моделирование систем / Ю. Г. Карпов. – СПб.: БХВ-Петербург , 2006. – 389 с.
2. Simulink: Инструмент моделирования динамических систем [Електронный ресурс] // Режим доступа до сайту: http://matlab.exponenta.ru/simulink/book1/1.php
3. TerraME: Simulation and Modelling of Terrestrial Systems [Електронный ресурс] // Режим доступа до сайта: http://www.terrame.org/doku.php
4. Бандман О. Л. Клеточно – автоматные модели пространственной динамики // Системная информатика. Новосибирск: СО РАН, 2006. Вып. 10. С. 59–113.
5. von Neumann J. Theory of self reproducing automata / J. von Neumann – University of Illinois, Urbana, USA. 1966.
6. Wolfram S. A new kind of science / S. Wolfram – Wolfram Media Inc., Champaign, Ill., USA. 2002.
7. Аноприенко А. Я., Коноплева А. П. Клеточные автоматы в историческом контексте и их классификация // Збірка матеріалів п’ятої міжнародної науково-технічної конференції студентів, аспірантів та молодих науковців 2009 р.. Серія «Інформатика та комп’ютерні технології» (ІКТ-2009). – Донецьк: ДонНТУ. – 2009.
8. Советов Б. Я. Моделирование систем / Б. Я. Советов, С. А. Яковлев. – М.: Высшая школа, 1998. – 343 с. 9. Шеннон Р. Имитационное моделирование систем – искусство и наука / Р. Шеннон – М.: Мир, 1978. – 418 с.
9. Шеннон Р. Имитационное моделирование систем – искусство и наука / Р. Шеннон – М.: Мир, 1978. – 418 с.
10. Советов Б. Я. Моделирование систем / Б. Я. Советов, С. А. Яковлев. – М.: Высшая школа, 1998. – 343 с.
11. C++. [Електронный ресурс] // Режим доступа до сайту: https://uk.wikipedia.org/wiki/C%2B%2B

ДОДАТКИ

Текст програми

Tr2.cpp:

```
#include "Util/menu.h"
#include "Util/Win32.h"
#ifdef _WIN32
#include <windows.h>
#endif
void setConsolePosition()
{
#ifdef _WIN32
   HWND consoleWindow = GetConsoleWindow();
   SetWindowPos(consoleWindow, 0, 0, 0, 500, 500,
SWP_NOSIZE | SWP_NOZORDER);
#endif
}
int main()
{
   setlocale(LC_ALL, "rus");
   setConsolePosition(); //задання позиції конскольного вікна
   menu();
}
```

Entitymanager.h:

```
#pragma once

#include <SFML/Graphics.hpp>
#include "World.h"
#include "../Util/Random.h"
#include <vector>
#include "Warrior.h"
#include <iostream>
#include <map>
#include <algorithm>

class Entitymanager {
public:
   Entitymanager();
   void Draw(sf::RenderWindow& window);
   void attachtoWorld(World& world);
   void spawnWarriors(int howMany);
   void Update();
   int getNbrColonies();
   unsigned int getNbrWarriors(){ return
m_livingWarriors.size();};
   ~Entitymanager();

private:
   void updateColonies();
   bool move (Warrior* warrior);
   bool isPositionValid(sf::Vector2i positionToCheck);
   void clearAllDeadWarriors();
   bool checkFriendly(Warrior* warrior,Warrior* other_Warrior);
   bool checkCollision(sf::Vector2i posToCheck);
   void populatePosition(sf::Vector2i position, Warrior* warrior);
   void clearPosition(sf::Vector2i position);
   void killWarrior(sf::Vector2i position);
   void reproduce(Warrior* warrior);
   void spawnChild(Warrior* warrior,sf::Vector2i childPos);
   bool isHeWinningFight(Warrior* warrior,Warrior*
other_warrior);
   int convertVectorToInt(sf::Vector2i);
   void updateWarriors();
   void updateVertexArray();
   void killSpecies(sf::Vector3i color);

private:
   std::vector<Warrior*>m_WarriorPositions;
   sf::VertexArray m_WarriorVertices;
   std::vector<Warrior>m_livingWarriors;
   std::vector<sf::Vector2f>m_warrior_buffer;
   std::vector<sf::Vector3i>colonies;
   World * p_world;
   Random m_rnd;
   Warrior* getPointerAtPosition(sf::Vector2i position);
};
```

Game.h:

```
#pragma once

#include <iostream>
#include <SFML/Graphics.hpp>
#include "../Util/Config.h"
#include "../Util/ResourceHolder.h"
#include "../Util/Random.h"
#include "World.h"
#include "Entitymanager.h"

class Game {
private:
   World world;
   Entitymanager entMgr;
   sf::Text fpsCounter;
   sf::Text Counter;
   sf::View view;
   sf::RenderWindow window;
   sf::Clock delayTimer;
   sf::Clock fpsTimer;
   Random rnd;
public:
   Game();
   ~Game();
   void run();
private:
   float time_passed;
   float event_timer;
   float fps;
   int frameCount = 0;
   bool pause = false;
   void update(sf::Time elapsed);
   void ViewReset();
   void processEvents(sf::Time elapsed);
   void render(sf::Time elapsed);
   void fpsTextCounter();
};
```

World.h:

```
#pragma once

#include <iostream>
#include <SFML/Graphics.hpp>
#include <vector>
#include <string>

class World {
public:
   friend class Entitymanager;
   World(sf::Vector2u worldSizeTiles,int tileSize);
   void toggleGrid();
   enum Tilestate{BORDER,GROUND/*,WATER,DESERT*/};
   void setdrawGrid(bool state);
   void Draw(sf::RenderWindow& window);
   void createMap();
   ~World();
   std::vector<int> getMap();
private:
   void m_BuildGrid();
   std::vector<int>m_map;
   bool m_drawGrid;
   int m_tileSize;
   sf::Vector2u m_worldSizeInTiles;
   sf::VertexArray m_overlayGrid;
   sf::VertexArray m_borderGrid;
};
```

Warrior.h:

```
#pragma once

#include "SFML/Graphics.hpp"
#include <vector>
```

```
#include <chrono>
#include <cstdlib>
#include <ctime>

class Warrior {
public:
   Warrior(sf::Vector2i spawnPos, unsigned int strength, unsigned
int maxAge, unsigned int reproductionBonus);
   void update();
   void kill();
   bool canReproduce();
   int getMaxAge(){ return m_maxAge;};
   int getReprductionBonus(){ return m_reproductionBonus;};
   int getStrength(){return m_strength;};
   bool isAlive(){ return alive;};
   sf::Vector3i getColor();
   void setPosition(sf::Vector2i newPos);
   void ExpermentalSetColor(sf::Vector3i color){m_color =
color;};
   sf::Vector2i getPosition();
   ~Warrior();
private:
   bool alive;
   sf::Vector2i m_positionInGrid;
   sf::Vector3i m_color;
   unsigned int m_strength;
   unsigned int m_age;
   unsigned int m_reproductionValue;
   unsigned int m_maxAge;
   unsigned int m_reproductionBonus;
};
```

AddAnt.h:

```
#pragma once

#include <vector>
#include "CellularAutomaton.h"
#include "Ant.h"

class AddAnt : public CellularAutomaton
{
   enum Cell
   {
      On,
      Off
   };
   public:
      AddAnt();
      void input(const sf::Event& e) override;
      void update() override;
   private:
      sf::Vector2i addAnt();
      void updateAnt(Ant& ant);
      std::vector<Ant> m_ants;
      std::vector<Cell> m_cells;
};
```

Ant.h:

```
#ifndef ANT_H_INCLUDED
#define ANT_H_INCLUDED

#include <SFML/Graphics.hpp>
#include <iostream>
//порядок проти годинникової стрілки
enum class Direction
{
   Up,
   Left,
   Down,
   Right,
};

enum class Turn
{
   Left, Right
};

class Ant //стрелка сюда как ассоциация от AddAnt
{
   public:
      Ant            (int xPosition, int yPosition);
      void turn      (Turn direction);
      void translate  ();
      const sf::Vector2i& getPosition() const { return m_position; }
      void setX(int x) { m_position.x = x;    }
      void setY(int y) { m_position.y = y;    }
      sf::Color getColour () const { return m_colour; }
   private:
      sf::Vector2i    m_position;
      Direction       m_direction;
      sf::Color       m_colour;
};
#endif // ANT_H_INCLUDED
```

Application.h:

```
#ifndef APPLICATION_H_INCLUDED
#define APPLICATION_H_INCLUDED

#include <memory>
#include <SFML/Graphics.hpp>
#include "../Util/FPSCounter.h"
#include "CellularAutomaton.h"

class Application
{
   public:
      Application();
      template<typename T>
      void init()
      {
         m_automaton = std::make_unique<T>();
      }
      void run();
   private:
      void pollEvents();
      void input  (float dt);
      void render ();
      void resetView();
      FPSCounter m_fpsCounter;
      std::unique_ptr<CellularAutomaton> m_automaton;
      sf::RenderWindow m_window;
      sf::Text m_guiText;
      sf::View  m_view;
      bool m_pause = false;
};
#endif // APPLICATION_H_INCLUDED
```

CellularAutomaton.h:

```
#pragma once

#include <SFML/Graphics.hpp>
#include <vector>
#include "../Util/Config.h"

class CellularAutomaton
{
   public:
      CellularAutomaton();
      virtual ~CellularAutomaton() = default;
      virtual void input(const sf::Event& e) {}
      virtual void update() = 0;
      virtual void onRenderGUI(sf::RenderWindow& window) {};
      virtual void onRenderCells(sf::RenderWindow& window) {};
      void render(sf::RenderWindow& window);
   protected:
      unsigned getCellIndex(unsigned x, unsigned y);
      void setCellColour(int x, int y, sf::Color colour);
      template<typename Func>
      void cellForEach(Func function);
      void addQuad(unsigned xIndex, unsigned yIndex);
      std::vector<sf::Vertex> m_cellVertexPoints;
```

```
};

template<typename Func>
void CellularAutomaton::cellForEach(Func function)
{
   for (unsigned y = 0; y < Config::GetInstans().simSize.y; y++) {
      for (unsigned x = 0; x < Config::GetInstans().simSize.x; x++)
{
         function(x, y);
      }
   }
}
```

Creature.h:

```
#pragma once

#include <SFML/Graphics.hpp>

enum class CreatureType
{
   Predator = 0,
   Prey = 1,
   Nothing = 2
};
class Creature
{
public:
   constexpr static int MAX_HEALTH = 100;
   Creature();
   sf::Color getColour() noexcept;
   CreatureType getType() const noexcept;
   void setType(CreatureType type) noexcept;
   void heal(int amount);
   int  getHealth() const noexcept;
   void setHealth(int val) noexcept;
   void update();
   void reproduce(Creature& other);
   void move(Creature& other);
private:
   CreatureType m_type;
   int m_health = MAX_HEALTH / 5;
};
```

GameOfLife.h:

```
#pragma once

#include <vector>
#include "CellularAutomaton.h"

class GameOfLife : public CellularAutomaton
{
   enum Cell
   {
      On,
      Off
   };
   public:
      GameOfLife();
      void update() override;
   private:
      std::vector<Cell> m_cells;
};
```

PredatorAndPrey.h:

```
#pragma once

#include <vector>
#include "../Util/Random.h"
#include "CellularAutomaton.h"
#include "Creature.h"

class PredatorAndPrey : public CellularAutomaton
{
   enum Cell
   {
      On,
      Off
   };
   public:
      PredatorAndPrey();
      void update() override;
   private:
      void updatePredator(Creature& thisCreature, Creature&
otherCreature);
      void updatePrey(Creature& thisCreature, Creature&
otherCreature);
      std::vector<Creature> m_creatures;
      int m_preyCount = 0;
      int m_predatorCount = 0;
};
```

Config.h:

```
#ifndef CONFIG_H_INCLUDED
#define CONFIG_H_INCLUDED

#include <SFML/Graphics.hpp>
#include <iostream>

class Config
{
   Config() = default;
   Config(const sf::Vector2u& winSize, unsigned cellSize)
      : windowSize    (winSize)
      ,   cellSize    (cellSize)
   {
      init();
   }
public:
   void init()
   {
      windowSize.x -= windowSize.x % cellSize;
      windowSize.y -= windowSize.y % cellSize;

      simSize.x = windowSize.x / cellSize;
      simSize.y = windowSize.y / cellSize;
   }
   sf::Vector2u windowSize = { 1280, 720 };
   sf::Vector2u simSize;
   sf::Color bgColour = {125, 125, 125};
   sf::Color fgColour = {25, 25, 25};
   unsigned cellSize = 2;
   unsigned fps = 30;
   unsigned WarriorsAtStart = 50;
   static Config& GetInstans()
   {
      static Config tmp;
      return tmp;
   }
};
#endif // CONFIG_H_INCLUDED
```

FPSCounter.h:

```
#ifndef FPSCOUNTER_H_INCLUDED
#define FPSCOUNTER_H_INCLUDED

#include <SFML/Graphics.hpp>

class FPSCounter
{
   public:
      FPSCounter();
      void update();
      void draw(sf::RenderTarget& renderer);
   private:
      sf::Text m_text;
      sf::Clock m_delayTimer;
      sf::Clock m_fpsTimer;
      float m_fps = 0;
      int m_frameCount = 0;
```

```
};
#endif // FPSCOUNTER_H_INCLUDED
```

menu.h:

```
#pragma once

#include "Config.h"
#include "ResourceHolder.h"
#include <sstream>
#include "../TypicalModel/Application.h"
#include "../TypicalModel/GameOfLife.h"
#include "../TypicalModel/AddAnt.h"
#include "../TypicalModel/PredatorAndPrey.h"
#include "../OriginalModel/Game.h"
#include "Win32.h"
#ifdef _WIN32
#include <windows.h>
#endif

using namespace sf;
template<typename T>
void run()
{
        Application app;
        app.init<T>();
        app.run();
}
void loadConfig();
void menu();
```

Random.h:

```
#ifndef RANDOM_H_INCLUDED
#define RANDOM_H_INCLUDED

#include <random>
#include <time.h>
#include "SFML/Graphics.hpp"

class Random
{
   public:
      Random();

      static Random& get();
      Random            (const Random& other)    = delete;
      Random& operator=   (const Random& other)   = delete;

      int intInRange(int low, int high);
      float floatInRange(float low, float high);

      int getRandomInt(int min, int max);
      sf::Vector3i getRandomColor();
      sf::Vector2i generateMovePos(sf::Vector2i oldPosition);
      sf::Vector2i getRandomPosition(int minx, int maxx, int miny,
int maxy);
      int getMutation();
      void reseed();

   private:
      std::mt19937 m_rng;
};

#endif // RANDOM_H_INCLUDED
```

ResourceHolder.h:

```
#ifndef RESOURCEHOLDER_H_INCLUDED
#define RESOURCEHOLDER_H_INCLUDED

#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

#include "ResourceManager.h"

class ResourceHolder
{
   public:
      static ResourceHolder& get();

      ResourceManager<sf::Font>          fonts;
      ResourceManager<sf::Texture>       textures;

   private:
      ResourceHolder();
};

#endif // RESOURCEHOLDER_H_INCLUDED
```

ResourceManager.h:

```
#ifndef RESOURCEMANAGER_H_INCLUDED
#define RESOURCEMANAGER_H_INCLUDED

#include <unordered_map>
#include <string>

template<typename Resource>
class ResourceManager
{
   using CrString = const std::string&;

   public:
      ResourceManager (CrString folder, CrString extention)
      :   m_folder    ("Res/" + folder + "/")
      ,   m_extention ("." + extention)
      { }

      const Resource& get(CrString name)
      {
         if (!exists(name))  {
            add(name);
         }

         return m_resources.at(name);
      }

      bool exists(CrString name) const
      {
         return m_resources.find(name) != m_resources.end();
      }

      void add(CrString name)
      {
         Resource r;

         //якщо ресурс не завантажується, він додає ресурс "fail"
за замовчуванням
         if(!r.loadFromFile(getFullname(name))) {
            Resource fail;
            fail.loadFromFile(m_folder + "помилка!" +
m_extention);
            m_resources.insert(std::make_pair(name, fail));
         }
         else {
            m_resources.insert(std::make_pair(name, r));
         }
      }

   private:
      std::string getFullname(CrString name)
      {
         return m_folder + name + m_extention;
      }

      const std::string m_folder;
      const std::string m_extention;

      std::unordered_map<std::string, Resource> m_resources;
```

```
};

#endif // RESOURCEMANAGER_H_INCLUDED
```

Win32.h:

```
#ifndef WIN32_H_INCLUDED
#define WIN32_H_INCLUDED

#include <ostream>

#ifdef _WIN32
   #include "windows.h"

   enum class TextColour
   {
      Default      = -1,
      Black        = 0,
      DarkBlue     = FOREGROUND_BLUE,
      DarkGreen    = FOREGROUND_GREEN,
      DarkCyan     = FOREGROUND_GREEN    |
FOREGROUND_BLUE,
      DarkRed      = FOREGROUND_RED,
      DarkMagenta  = FOREGROUND_RED      |
FOREGROUND_BLUE,
      DarkYellow   = FOREGROUND_RED      |
FOREGROUND_GREEN,
      DarkGrey     = FOREGROUND_RED      |
FOREGROUND_GREEN | FOREGROUND_BLUE,
      Grey         = FOREGROUND_INTENSITY,
      Blue         = FOREGROUND_INTENSITY |
FOREGROUND_BLUE,
      Green        = FOREGROUND_INTENSITY |
FOREGROUND_GREEN,
      Cyan         = FOREGROUND_INTENSITY |
FOREGROUND_GREEN     | FOREGROUND_BLUE,
      Red          = FOREGROUND_INTENSITY |
FOREGROUND_RED,
      Magenta      = FOREGROUND_INTENSITY |
FOREGROUND_RED       | FOREGROUND_BLUE,
      Yellow       = FOREGROUND_INTENSITY |
FOREGROUND_RED       | FOREGROUND_GREEN,
      White        = FOREGROUND_INTENSITY |
FOREGROUND_RED       | FOREGROUND_GREEN |
FOREGROUND_BLUE,
   };

   std::ostream& operator<< (std::ostream& stream, TextColour t);

#endif // __WIN32

#endif // WIN32_H_INCLUDED
```

Entitymanager.cpp:

```
#include "Entitymanager.h"

Entitymanager::Entitymanager() {
}

Entitymanager::~Entitymanager() {

}

void Entitymanager::Draw(sf::RenderWindow &window) {
   window.draw(m_WarriorVertices);
}

void Entitymanager::spawnWarriors(int howMany)
{
   for (int warrior = 0; warrior < howMany; warrior++)
   {
      sf::Vector2i rand_pos = m_rnd.getRandomPosition(1,
p_world->m_worldSizeInTiles.x - 1, 1,
                                                      p_world-
>m_worldSizeInTiles.y - 1);
      if (p_world->m_map[convertVectorToInt(rand_pos)] !=
World::BORDER && isPositionValid(rand_pos) &&
!checkCollision(rand_pos))
      {
         Warrior buffer = Warrior(rand_pos,
m_rnd.getRandomInt(10, 100),
            m_rnd.getRandomInt(40, 100),
            m_rnd.getRandomInt(0, 10));
         sf::Vector3i random_color = m_rnd.getRandomColor();
         colonies.push_back(random_color);
         buffer.ExpermentalSetColor(random_color);
         m_livingWarriors.push_back(buffer);
         m_WarriorPositions[convertVectorToInt(rand_pos)] =
&m_livingWarriors.back();
      }
   }
}

void Entitymanager::attachtoWorld(World& world) {
   this->p_world = &world;
   m_WarriorPositions.resize(p_world->m_map.size());
   m_WarriorPositions.assign(m_WarriorPositions.size(), nullptr);
//заповнити вектор нульовими покажчиками;
   long max_amount_vertices = 4*(p_world->m_map.size());
   /*p_WarriorVertices = &m_WarriorVertices;*/
   m_WarriorVertices.setPrimitiveType(sf::Quads);
   m_WarriorVertices.resize(max_amount_vertices);
   m_warrior_buffer.reserve((p_world->m_map.size()*4));
   m_livingWarriors.reserve(p_world->m_map.size());
}


int Entitymanager::convertVectorToInt(sf::Vector2i pos) {
   return(p_world->m_worldSizeInTiles.y*pos.x)+pos.y;
}

void Entitymanager::updateVertexArray() {
   m_WarriorVertices.clear();
   m_warrior_buffer.clear();
   int m_tileSize = p_world->m_tileSize;
   int tile_warrior_size_difference = m_tileSize*0.8;
   for(auto& item:m_livingWarriors)
   {
      sf::Vector2i pos = item.getPosition();

m_warrior_buffer.push_back(sf::Vector2f((float)((pos.x*m_tileSiz
e)+tile_warrior_size_difference),(float)((pos.y*m_tileSize)+tile_w
arrior_size_difference)));

m_warrior_buffer.push_back(sf::Vector2f((float)(((pos.x+1)*m_til
eSize)-
tile_warrior_size_difference),(float)((pos.y*m_tileSize)+tile_warri
or_size_difference)));

m_warrior_buffer.push_back(sf::Vector2f((float)(((pos.x+1)*m_til
eSize)-
tile_warrior_size_difference),(float)(((pos.y+1)*m_tileSize)-
tile_warrior_size_difference)));

m_warrior_buffer.push_back(sf::Vector2f((float)((pos.x*m_tileSiz
e)+tile_warrior_size_difference),(float)(((pos.y+1)*m_tileSize)-
tile_warrior_size_difference)));
   }
   m_WarriorVertices.resize(m_warrior_buffer.size());
   for(int x =0;x<m_warrior_buffer.size();x++)
   {
      m_WarriorVertices[x].position = m_warrior_buffer[x];
      sf::Vector3i color = m_livingWarriors[(int)x/4].getColor();
      m_WarriorVertices[x].color =
sf::Color(color.x,color.y,color.z);
   }
}


void Entitymanager::Update() {
   updateWarriors();
```

```
    updateColonies();
    if (getNbrColonies() == 1 || getNbrWarriors() > 2000)
    {
        killSpecies(colonies[0]);
    }
    if (getNbrWarriors() == 0)
    {
        spawnWarriors(50);
    }
    if(getNbrColonies()<10)
    {
        spawnWarriors(10-getNbrColonies());
    }
    updateVertexArray();
}

void Entitymanager::updateWarriors() {
    for(auto& warrior:m_livingWarriors)
    {
        warrior.update();
        if(warrior.isAlive())
        {
            if(move(&warrior)&& warrior.canReproduce())
            {
                reproduce(&warrior);
            }
        }
    }
    clearAllDeadWarriors();
}

bool Entitymanager::move(Warrior* warrior) {
    sf::Vector2i oldpos= warrior->getPosition();
    sf::Vector2i newPos = m_rnd.generateMovePos(oldpos);
    if(isPositionValid(newPos)) //перевірити, чи дозволена ця
позиція          1 (+) Позиція зайнята межею - вихід
    {
        if(checkCollision(newPos)) // 2 (-) Позиція не зайнята
іншим персонажем - зайняття позиції - вихід   3 (+) Позиція
зайнята іншим персонажем
        {
            //щож у нас зіткнення, давайте розберемося, що робити
            Warrior* warriorAtPosition =
getPointerAtPosition(newPos);
            if(!checkFriendly(warrior,warriorAtPosition)) //  3- 4+
            {
                clearPosition(oldpos);
                //це не друг, дозволяємо битися
                if(!isHeWinningFight(warrior,warriorAtPosition)) // 4+
5-
                {
                    //щож він програв
                    warrior->kill();
                    return false;
                }
                else
                {
                    //а якщо він виграв, вбиваємо опонента
                    warrior->setPosition(newPos);
                    killWarrior(newPos);
                    populatePosition(newPos, warrior);
                    return true;
                }
            } //
            //є друг, ми не рухаємося і не розмножуємося (запобігає
перенаселення в цій області)
            return false;
        }
        else  //
        {
            //добре, це безпечно, тож давайте перейдемо туди
            warrior->setPosition(newPos);
            clearPosition(oldpos);
            populatePosition(newPos, warrior);
            return true;
        }
    }
    return false; //це був перехід на заборонену позицію
}

bool Entitymanager::checkCollision(sf::Vector2i posToCheck) {
    return (m_WarriorPositions[convertVectorToInt(posToCheck)]
!= nullptr);
}

bool Entitymanager::isHeWinningFight(Warrior* warrior,
Warrior* other_warrior) {
    return warrior->getStrength()<= other_warrior->getStrength();
}

bool Entitymanager::checkFriendly(Warrior* warrior, Warrior*
other_warrior) {
    return (warrior->getColor() == other_warrior->getColor());
}

void Entitymanager::populatePosition(sf::Vector2i position,
Warrior* warrior) {
    m_WarriorPositions[convertVectorToInt(position)]=warrior;
}

void Entitymanager::clearPosition(sf::Vector2i position) {
    m_WarriorPositions[convertVectorToInt(position)] = nullptr;
}

void Entitymanager::killWarrior(sf::Vector2i position) {
    m_WarriorPositions[convertVectorToInt(position)]->kill();
}

void Entitymanager::reproduce(Warrior* warrior) {
    sf::Vector2i parentPos = warrior->getPosition();
    sf::Vector2i childPos = m_rnd.generateMovePos(parentPos);
    if(!checkCollision(childPos) && isPositionValid(childPos))
    {
        //немає зіткнення, тож давайте зробимо дитину
        spawnChild(warrior,childPos);
    }
}

void Entitymanager::spawnChild(Warrior* parent, sf::Vector2i
childPos) {
    if(m_rnd.getMutation() == 1)
    {
        Warrior buffer = Warrior(childPos,parent-
>getStrength()+m_rnd.getMutation(),parent->getMaxAge()+
m_rnd.getMutation(),parent-
>getReprductionBonus()+m_rnd.getMutation());
        buffer.ExpermentalSetColor(parent->getColor());
        m_livingWarriors.push_back(buffer);
        populatePosition(childPos,&m_livingWarriors.back());
    }
    else
    {
        Warrior buffer = Warrior(childPos,parent-
>getStrength(),parent->getMaxAge(),parent-
>getReprductionBonus());
        buffer.ExpermentalSetColor(parent->getColor());
        m_livingWarriors.push_back(buffer);
        populatePosition(childPos,&m_livingWarriors.back());
    }
}

bool Entitymanager::isPositionValid(sf::Vector2i
positionToCheck) {
    if(convertVectorToInt(positionToCheck) < 0 ||
convertVectorToInt(positionToCheck) > p_world->m_map.size())
    {
        std::cout<<positionToCheck.x<<"|"<<positionToCheck.y<< "
за вектором, "
<<convertVectorToInt(positionToCheck)<<std::endl;
        return false;
    }
    if(p_world->m_map[convertVectorToInt(positionToCheck)] ==
World::BORDER)
    {
        //printf("border");
        return false;
```

```
    }
    return true;
}

Warrior *Entitymanager::getPointerAtPosition(sf::Vector2i
position) {
    return m_WarriorPositions[convertVectorToInt(position)];
}

void Entitymanager::clearAllDeadWarriors() {
    for(int x=0;x<m_livingWarriors.size();x++)
    {
        if(!m_livingWarriors[x].isAlive())
        {
            clearPosition(m_livingWarriors[x].getPosition()); //викинь
старий покажчик
            m_livingWarriors.erase(m_livingWarriors.begin()+x);
//видалити об’єкт
            x=0;
        }
    }
}

void Entitymanager::updateColonies() {
    for(int x=0;x<colonies.size();x++)
    {
        bool found =0;
        for(auto& cr:m_livingWarriors)
        {
            if(cr.getColor()==colonies[x])
            {
                found =1;
                break;
            }
        }
        if(!found)
        {
            colonies.erase(colonies.begin()+x);
        }
    }
}

int Entitymanager::getNbrColonies() {
    return colonies.size();
}

void Entitymanager::killSpecies(sf::Vector3i color) {
    for(int x=0;x<m_livingWarriors.size();x++)
    {
        if(m_livingWarriors[x].getColor() == color)
        {
            m_livingWarriors[x].kill();
        }
    }
}
```

Game. cpp:

```
#include "Game.h"

Game::Game():
    world({ Config::GetInstans().simSize.x / 5,
Config::GetInstans().simSize.y / 5}, Config::GetInstans().cellSize
* 4) ,
    window({ Config::GetInstans().windowSize.x,
Config::GetInstans().windowSize.y },L"Арена")
{
    rnd.reseed();
    world.setdrawGrid(true);
    entMgr.attachtoWorld(world);
    entMgr.spawnWarriors(Config::GetInstans().WarriorsAtStart);
    view.setSize({ (float)Config::GetInstans().windowSize.x ,
(float)Config::GetInstans().windowSize.y });
    view.setCenter({ (float)Config::GetInstans().windowSize.x / 2 -
120, (float)Config::GetInstans().windowSize.y / 2  - 100});
}

Game::~Game() {
}
void Game::fpsTextCounter()
{
    fpsCounter.setString("FPS: " + std::to_string((int)fps));
    fpsCounter.setFont(ResourceHolder::get().fonts.get("arial"));
    fpsCounter.setCharacterSize(24);
    fpsCounter.setStyle(sf::Text::Bold);
    fpsCounter.setFillColor(sf::Color::Black);
    fpsCounter.setPosition(10, -50);

    Counter.setString(std::wstring(L"Персонажів: " +
std::to_wstring(entMgr.getNbrWarriors())));
    Counter.setFont(ResourceHolder::get().fonts.get("arial"));
    Counter.setCharacterSize(24);
    Counter.setStyle(sf::Text::Bold);
    Counter.setFillColor(sf::Color::Black);
    Counter.setPosition(10, -30);
}

void Game::run() {

    sf::Clock clock;
    while (window.isOpen())
    {
        sf::Time elapsed = clock.restart();
        processEvents(elapsed);

        if (pause == false) {
            update(elapsed);
            render(elapsed);
        }
    }
}

void Game::update(sf::Time elapsed) {
    fpsTextCounter();

    time_passed += elapsed.asSeconds();
    if (time_passed > 0.1)
    {
        window.draw(Counter);
        window.draw(fpsCounter);
        window.display();

        time_passed = 0.0;
        entMgr.Update();
    }

    frameCount++;
    if (delayTimer.getElapsedTime().asSeconds() > 0.2)
    {
        fps = frameCount / fpsTimer.restart().asSeconds() / 100;
        frameCount = 0;
        delayTimer.restart();
    }
}

void Game::ViewReset()
{
    view = sf::View();
    view.setSize({ (float)Config::GetInstans().windowSize.x ,
(float)Config::GetInstans().windowSize.y });
    view.setCenter({ (float)Config::GetInstans().windowSize.x / 2 -
120, (float)Config::GetInstans().windowSize.y / 2 - 100 });
}

void Game::processEvents(sf::Time elapsed) {
    event_timer+=elapsed.asSeconds();
    if(event_timer>0.05)
    {
        event_timer=0.0;
        sf::Event event;
        while (window.pollEvent(event)) {
            switch (event.type) {
                case sf::Event::KeyPressed:
                    if(sf::Keyboard::isKeyPressed(sf::Keyboard::Escape))
                    {
```

```
                window.close(); //закриття вікна демонстрації
                break;
            }
            if(sf::Keyboard::isKeyPressed(sf::Keyboard::Enter)) {
                entMgr.spawnWarriors(100); //додання
персонажей
                break;
            }

if(sf::Keyboard::isKeyPressed(sf::Keyboard::BackSpace)) {
                entMgr.spawnWarriors(25); //додання персонажей

                break;
            }
            if(sf::Keyboard::isKeyPressed(sf::Keyboard::H)) {
                world.toggleGrid(); //вимкнути сітку
                break;
            }
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)){
                view.zoom(0.95f);
                break;
            }
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
{
                view.zoom(1.05f);
                break;
            }
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {
                ViewReset();
                break;
            }else
            { break;}
        case sf::Event::Closed:
            window.close();
            break;
      }
    }

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
    {
        view.move(-25.0f,0.0f);
    }
    else if(sf::Keyboard::isKeyPressed(sf::Keyboard::D))
    {
        view.move(25.0f,0.0f);
    }

    else if(sf::Keyboard::isKeyPressed(sf::Keyboard::W))
    {
        view.move(0.0f,-25.0f);
    }
    else if(sf::Keyboard::isKeyPressed(sf::Keyboard::S))
    {
        view.move(0.0f,25.0f);
    }
    else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
        pause = true;
    }
    else if ((pause == true) &&
sf::Keyboard::isKeyPressed(sf::Keyboard::G))
    {
        pause = false;
    }
  }
}


void Game::render(sf::Time elapsed) {
  time_passed+=elapsed.asSeconds();
  if(time_passed>0.05)
  {
    window.clear(sf::Color(50,50,50));

    window.setView(view);

    world.Draw(window);
    entMgr.Draw(window);
    window.draw(Counter);
    window.draw(fpsCounter);

    window.display();
  }
}
```

Warrior.cpp:

```
#include "Warrior.h"

Warrior::~Warrior() {

}

sf::Vector3i Warrior::getColor() {
  return m_color;
}

sf::Vector2i Warrior::getPosition() {
  return m_positionInGrid;
}

Warrior::Warrior(sf::Vector2i spawnPos, unsigned int strength,
unsigned int maxAge, unsigned int reproductionBonus) {

  this->m_positionInGrid = spawnPos;
  this->m_strength = strength;
  this->m_maxAge = maxAge;
  this->m_reproductionBonus = reproductionBonus;
  alive = true;
  m_age=0;
  m_reproductionValue=reproductionBonus;
}

void Warrior::update() {
  m_age++;
  m_reproductionValue++;
  if(m_age>m_maxAge)
  {
    alive = false;
  }
}

void Warrior::kill() {

  alive= false;
}

bool Warrior::canReproduce() {
  if(m_reproductionValue>15)
//"REPRODUCTION_THRESHOLD"
  {
    m_reproductionValue=m_reproductionBonus;
    return true;
  }
  return false;
}

void Warrior::setPosition(sf::Vector2i newPos) {
  m_positionInGrid=newPos;
}
```

World.cpp:

```
#include "World.h"
World::World(sf::Vector2u worldSizeTiles,int tileSize) {
  this->m_tileSize = tileSize;
  this->m_worldSizeInTiles = worldSizeTiles;
  this->m_drawGrid = true;
  createMap();
  if(m_drawGrid)
    m_BuildGrid();
}

World::~World() {
```

```
}

void World::Draw(sf::RenderWindow &window) {
   if(m_drawGrid)
      window.draw(m_overlayGrid);
   window.draw(m_borderGrid);
}

void World::m_BuildGrid() {
   std::vector<sf::Vector2f> vertices;
   for(int x_position
=0;x_position<m_worldSizeInTiles.x;x_position++)
   {

vertices.push_back(sf::Vector2f((float)x_position*m_tileSize,0.0f)
);

vertices.push_back(sf::Vector2f((float)x_position*m_tileSize,(floa
t)m_worldSizeInTiles.y*m_tileSize));
   }
   for(int y_position
=0;y_position<m_worldSizeInTiles.y;y_position++)
   {

vertices.push_back(sf::Vector2f(0.0f,(float)y_position*m_tileSize)
);

vertices.push_back(sf::Vector2f((float)m_worldSizeInTiles.x*m_ti
leSize,(float)y_position*m_tileSize));
   }

   int counter =0;
   m_overlayGrid = sf::VertexArray(sf::Lines,vertices.size());
   for(auto& entries:vertices)
   {
      m_overlayGrid[counter].position = entries;
      m_overlayGrid[counter].color = sf::Color(
            65, 82, 89
      );
      counter++;
   }

   int vertexCountBorders = 16;
   m_borderGrid= sf::VertexArray(sf::Quads,
vertexCountBorders);
   //ліва межа
   m_borderGrid[0].position = sf::Vector2f(0.0f, 0.0f);
   m_borderGrid[1].position = sf::Vector2f(m_tileSize, 0.0f);
   m_borderGrid[2].position = sf::Vector2f(m_tileSize,
m_worldSizeInTiles.y*m_tileSize);
   m_borderGrid[3].position = sf::Vector2f(0.0f,
m_worldSizeInTiles.y*m_tileSize);

   //нижня межа
   m_borderGrid[4].position = sf::Vector2f(0.0f,
(m_worldSizeInTiles.y-1)*m_tileSize);
   m_borderGrid[5].position =
sf::Vector2f(m_tileSize*m_worldSizeInTiles.x,
(m_worldSizeInTiles.y-1)*m_tileSize);
   m_borderGrid[6].position =
sf::Vector2f(m_tileSize*m_worldSizeInTiles.x,
m_worldSizeInTiles.y*m_tileSize);
   m_borderGrid[7].position = sf::Vector2f(0.0f,
m_worldSizeInTiles.y*m_tileSize);

   //права межа
   m_borderGrid[8].position =
sf::Vector2f((m_worldSizeInTiles.x-1)*m_tileSize, 0.0f);
   m_borderGrid[9].position =
sf::Vector2f(m_tileSize*m_worldSizeInTiles.x, 0.0f);
   m_borderGrid[10].position =
sf::Vector2f(m_tileSize*m_worldSizeInTiles.x,
m_worldSizeInTiles.y*m_tileSize);
   m_borderGrid[11].position =
sf::Vector2f((m_worldSizeInTiles.x-1)*m_tileSize,
m_worldSizeInTiles.y*m_tileSize);

   //верхня межа
   m_borderGrid[12].position = sf::Vector2f(0.0f, 0.0f);
   m_borderGrid[13].position =
sf::Vector2f(m_tileSize*m_worldSizeInTiles.x, 0.0f);
   m_borderGrid[14].position =
sf::Vector2f(m_tileSize*m_worldSizeInTiles.x, m_tileSize);
   m_borderGrid[15].position = sf::Vector2f(0.0f, m_tileSize);

   for(int vertex =0;vertex < vertexCountBorders;vertex++)
   {
      m_borderGrid[vertex].color = sf::Color(253, 217, 181);
//встановити колір межі
   }


}

void World::setdrawGrid(bool state) {
   m_drawGrid = state;
}

std::vector<int> World::getMap() {
   return m_map;
}

void World::createMap() {
   for (int
x_position=0;x_position<m_worldSizeInTiles.x;x_position++)
   {
      for (int
y_position=0;y_position<m_worldSizeInTiles.y;y_position++)
      {
         if(x_position==0 || x_position==m_worldSizeInTiles.x-1 ||
y_position==0 || y_position==m_worldSizeInTiles.y-1)
         {
            m_map.push_back(BORDER);
         }
         else
         {
            m_map.push_back(GROUND);
         }
      }
   }
}

void World::toggleGrid() {
   m_drawGrid = !m_drawGrid;
}
```

AddAnt.cpp:

```
#include "AddAnt.h"
#include <iostream>

#include "../Util/Random.h"
#include "../Util/Config.h"

AddAnt::AddAnt()
   : CellularAutomaton()
   , m_cells(Config::GetInstans().windowSize.x * 720)
{
   for (int i = 0; i < 5; i++) {
      addAnt();
   }
   std::cout << "Натисніть Enter для додання мурах!\n";
}

void AddAnt::input(const sf::Event& e)
{
   switch (e.type) {
      case sf::Event::KeyReleased:
         switch (e.key.code) {
            case sf::Keyboard::Enter: {
               auto location = addAnt();
               std::cout << "Мураха додана (за координатами:) X:
" << location.x << " Y: " << location.y << '\n';
            }
            default:
               break;
```

```
            }
        default:
            break;
    }
}

void AddAnt::update()
{
    for (auto& ant : m_ants)
        updateAnt(ant);
}

void AddAnt::updateAnt(Ant& ant)
{
    ant.translate();
    const auto& position = ant.getPosition();

    //handle oob/ wrapping
    if (position.x == 1280)
        ant.setX(0);
    else if (position.x == -1)
        ant.setX(Config::GetInstans().simSize.x - 1);

    if (position.y == (int)Config::GetInstans().simSize.y)
        ant.setY(0);
    else if (position.y == -1)
        ant.setY(Config::GetInstans().simSize.y - 1);

    auto& cell = m_cells[position.y *
Config::GetInstans().simSize.x + position.x];

    sf::Color colour;
    switch (cell) {
    case Cell::Off:
        cell = Cell::On;
        ant.turn(Turn::Right);
        colour = ant.getColour();
        break;

    case Cell::On:
        cell = Cell::Off;
        ant.turn(Turn::Left);
        colour = Config::GetInstans().fgColour;
        break;
    }

    CellularAutomaton::setCellColour(ant.getPosition().x,
        ant.getPosition().y,
        colour);
}

sf::Vector2i AddAnt::addAnt()
{
    int x = Random::get().intInRange(0,
Config::GetInstans().simSize.x - 1);
    int y = Random::get().intInRange(0,
Config::GetInstans().simSize.y - 1);
    m_ants.emplace_back(x, y);
    return {
        x, y
    };
}
```

Ant.cpp:

```
#include "Ant.h"

#include "../Util/Random.h"

namespace
{
    sf::Color getRandomColour()
    {
        uint8_t r = (uint8_t)Random::get().intInRange(0, 255);
        uint8_t g = (uint8_t)Random::get().intInRange(0, 255);
        uint8_t b = (uint8_t)Random::get().intInRange(0, 255);

        return { r, g, b };
    }
}

Ant::Ant(int xPosition, int yPosition)
:   m_position  (xPosition, yPosition)
,   m_direction (Direction(Random::get().intInRange(0, 3)))
,   m_colour    (getRandomColour())
{ }

void Ant::turn(Turn direction)
{
    int right = int(Direction::Right);
    int dir = (int)m_direction;
    switch (direction)
    {
        case Turn::Left:
            dir++;
            break;

        case Turn::Right:
            dir--;
            break;
    }

    if (dir < 0)
        dir = right;
    else if (dir > right)
        dir = 0;

    m_direction = Direction(dir);
}

void Ant::translate()
{
    switch (m_direction)
    {
        case Direction::Up:
            m_position.y--;
            break;

        case Direction::Down:
            m_position.y++;
            break;

        case Direction::Left:
            m_position.x--;
            break;

        case Direction::Right:
            m_position.x++;
            break;
    }
}
```

Application.cpp:

```
#include "Application.h"

#include <iostream>
#include <ctime>
#include <thread>

#include "../Util/Random.h"
#include "../Util/Config.h"
#include "../Util/Win32.h"

#include "../Util/ResourceHolder.h"

Application::Application()
    : m_window({ Config::GetInstans().windowSize.x,
Config::GetInstans().windowSize.y }, L"Клітинний автомат")
{
    resetView();
    m_guiText.setFont(ResourceHolder::get().fonts.get("arial"));
    m_guiText.move(10, 3);
    m_guiText.setCharacterSize(18);
```

```
    m_guiText.setOutlineColor(sf::Color::Black);
    m_guiText.setOutlineThickness(2);

    m_window.setFramerateLimit(Config::GetInstans().fps);
}

void Application::run()
{
    sf::Clock deltaClock;
    unsigned year = 0;
    while (m_window.isOpen()) {
        pollEvents();
        input(deltaClock.restart().asSeconds());
        if (m_pause == false) {
            m_guiText.setString(std::wstring(L"Генерація: " +
std::to_wstring(year++)));
            m_fpsCounter.update();
            m_automaton->update();
            render();
        }
    }
}

void Application::pollEvents()
{
    sf::Event e;
    while (m_window.pollEvent(e))
    {
        m_window.setView(m_view);
        m_automaton->input(e);
        if (e.type == sf::Event::Closed) {
            m_window.close();
        }
        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
            m_window.close(); //закриття вікна демонстрації
        }
        else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
            m_pause = true;
        }
        else if ((m_pause == true) &&
sf::Keyboard::isKeyPressed(sf::Keyboard::G)) {
            m_pause = false;
        }
        else if (e.type == sf::Event::KeyReleased) {
            if (e.key.code == sf::Keyboard::Up) {
                m_view.zoom(0.95f);
            }
            else if (e.key.code == sf::Keyboard::Down) {
                m_view.zoom(1.05f);
            }
            else if (e.key.code == sf::Keyboard::R) {
                resetView();
            }
        }
    }
}

void Application::input(float dt)
{
    float speed = 250;
    sf::Vector2f change;
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
        change.y -= speed;
    }
    else if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
        change.y += speed;
    }
    if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
        change.x -= speed;
    }
    else if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
        change.x += speed;
    }

    m_view.move(change * dt);
}

void Application::render()
{
    m_window.clear(Config::GetInstans().bgColour);

    //Пікселі
    m_window.setView(m_view);
    m_automaton->render(m_window);

    //GUI
    m_window.setView(m_window.getDefaultView());
    m_window.draw(m_guiText);
    m_fpsCounter.draw(m_window);

    m_window.display();
}

void Application::resetView()
{
    m_view = sf::View();
    m_view.setCenter({ (float)Config::GetInstans().windowSize.x /
2,  (float)Config::GetInstans().windowSize.y / 2 });
    m_view.setSize({ (float)Config::GetInstans().windowSize.x,
(float)Config::GetInstans().windowSize.y });
}
```

CellularAutomaton.cpp:

```
#include "CellularAutomaton.h"

#include <iostream>
#include "../Util/Win32.h"

CellularAutomaton::CellularAutomaton()
{
    m_cellVertexPoints.reserve(Config::GetInstans().simSize.x *
Config::GetInstans().simSize.y * 4);
    for (unsigned y = 0; y < Config::GetInstans().simSize.y; y++) {
        for (unsigned x = 0; x < Config::GetInstans().simSize.x; x++)
{
            addQuad(x, y);
        }
    }

}

void CellularAutomaton::render(sf::RenderWindow & window)
{
    window.draw(m_cellVertexPoints.data(),
m_cellVertexPoints.size(), sf::Quads);
    onRenderCells(window);
    onRenderGUI(window);
}

unsigned CellularAutomaton::getCellIndex(unsigned x, unsigned
y)
{
    return x + y * Config::GetInstans().simSize.x;
}

void CellularAutomaton::setCellColour(int x, int y, sf::Color
colour)
{
    auto index = (y * Config::GetInstans().simSize.x + x) * 4;
    for (int i = 0; i < 4; i++)
    {
        m_cellVertexPoints[index + i].color = colour;
    }
}

void CellularAutomaton::addQuad(unsigned xIndex, unsigned
yIndex)
{
    float cellSize = (float)Config::GetInstans().cellSize;
    sf::Vertex topLeft;
    sf::Vertex topRight;
    sf::Vertex bottomLeft;
```

```
    sf::Vertex bottomRight;

    float pixelX = xIndex * cellSize;
    float pixelY = yIndex * cellSize;

    topLeft     .position = { pixelX,             pixelY };
    topRight    .position = { pixelX + cellSize,  pixelY };
    bottomLeft  .position = { pixelX,             pixelY + cellSize };
    bottomRight .position = { pixelX + cellSize,  pixelY + cellSize
};

    topLeft     .color = Config::GetInstans().fgColour;
    topRight    .color = Config::GetInstans().fgColour;
    bottomLeft  .color = Config::GetInstans().fgColour;
    bottomRight .color = Config::GetInstans().fgColour;

    m_cellVertexPoints.push_back(topLeft);
    m_cellVertexPoints.push_back(bottomLeft);
    m_cellVertexPoints.push_back(bottomRight);
    m_cellVertexPoints.push_back(topRight);
}
```

Creature.cpp:

```
#include "Creature.h"

#include "../Util/Random.h"

Creature::Creature()
{
    auto n = Random::get().intInRange(0, 1000);

    if (n > 100)
    {
        m_type = CreatureType::Nothing;
    }
    else if (n > 50)
    {
        m_type = CreatureType::Prey;
    }
    else
    {
        m_type = CreatureType::Predator;
    }
}

sf::Color Creature::getColour() noexcept
{
    if (m_type == CreatureType::Nothing || m_health == 0)
    {
        return sf::Color::Black;
    }
    else
    {
        float normalisedHealth = (float)m_health /
(float)MAX_HEALTH;
        uint8_t col = uint8_t(normalisedHealth * 255);

        switch (m_type)
        {
        case CreatureType::Predator:
            return { col, 0, 0 };

        case CreatureType::Prey:
            return { 0, col, 0 };

        default:
            return sf::Color::Black;
        }
    }
}

CreatureType Creature::getType() const noexcept
{
    return m_type;
}

void Creature::setType(CreatureType type) noexcept
{
    m_type = type;
}

void Creature::heal(int amount)
{
    m_health += amount;
    m_health = std::min(m_health, MAX_HEALTH);
}

int Creature::getHealth() const noexcept
{
    return m_health;
}

void Creature::update()
{
    switch (m_type)
    {
    case CreatureType::Predator:
        heal(-1);
        break;

    case CreatureType::Prey:
        heal(1);
        break;

    default:
        break;
    }
}

void Creature::reproduce(Creature& other)
{
    other.m_health = 10;
    other.m_type = CreatureType::Prey;
}

void Creature::move(Creature& other)
{
    other.m_health = m_health;
    other.m_type = m_type;
    m_type = CreatureType::Nothing;
}

void Creature::setHealth(int val) noexcept
{
    m_health = val;
}
```

GameOfLife.cpp:

```
#include "GameOfLife.h"
#include <iostream>
#include <random>

GameOfLife::GameOfLife()
    : CellularAutomaton()
    , m_cells(Config::GetInstans().simSize.x *
Config::GetInstans().simSize.y)
{
    std::mt19937 rng((unsigned)std::time(nullptr));
    cellForEach([&](unsigned x, unsigned y)
    {
        unsigned index = getCellIndex(x, y);
        std::uniform_int_distribution<int> dist(0, 1);

        auto& cell = m_cells[index];
        cell = (Cell)dist(rng);
        CellularAutomaton::setCellColour(x, y, cell == Cell::On ?
sf::Color::Black : Config::GetInstans().fgColour);
    });
}

void GameOfLife::update()
{
```

```
    std::vector<std::pair<sf::Vector2i, Cell>> updates;
    cellForEach([&](unsigned x, unsigned y)
    {
        sf::Vector2i loc(x, y);
        unsigned count = 0;
        for (int nX = -1; nX <= 1; nX++)    //перевірка сусідів
            for (int nY = -1; nY <= 1; nY++)
            {
                int newX = nX + x;
                int newY = nY + y;

                if (newX == -1 || newX ==
(int)Config::GetInstans().simSize.x ||
                    newY == -1 || newY ==
(int)Config::GetInstans().simSize.y || //за межами
                    (nX == 0 && nY == 0)) //самаж клітина
                {
                    continue;
                }

                auto cell = m_cells[getCellIndex(newX, newY)];
                if (cell == Cell::On)
                    count++;
            }

        int index = getCellIndex(x, y);
        auto cell = m_cells[index];
        switch (cell)
        {
            case Cell::On:
                if (count < 2 || count > 3)
                {
                    updates.emplace_back(loc, Cell::Off);
                }
                break;

            case Cell::Off:
                if (count == 3)
                {
                    updates.emplace_back(loc, Cell::On);
                }
                break;
        }
    });
    for (auto& update : updates) {
        m_cells[getCellIndex(update.first.x, update.first.y)] =
update.second;

        CellularAutomaton::setCellColour(update.first.x,
update.first.y,
            update.second == Cell::On ? sf::Color::Black :
Config::GetInstans().fgColour);
    }
}
```

PredatorAndPrey.cpp:

```
#include "PredatorAndPrey.h"

PredatorAndPrey::PredatorAndPrey()
    : CellularAutomaton()
    , m_creatures(Config::GetInstans().simSize.x *
Config::GetInstans().simSize.y)
{
    cellForEach([&](unsigned x, unsigned y) {
        auto index = getCellIndex(x, y);
        auto type = m_creatures[index].getType();
        switch (type)
        {
        case CreatureType::Prey:
            m_preyCount++;
            break;

        case CreatureType::Predator:
            m_predatorCount++;
            break;

        case CreatureType::Nothing:
            break;
        }
        setCellColour(x, y, m_creatures[index].getColour());
    });

}

void PredatorAndPrey::update()
{
    cellForEach([&](unsigned x, unsigned y) {
        auto index = getCellIndex(x, y);
        auto& thisCreature = m_creatures[index];
        auto thisType = thisCreature.getType();

        if (thisType == CreatureType::Nothing)
            return;

        int xChange = Random::get().intInRange(-1, 1);
        int yChange = Random::get().intInRange(-1, 1);
        int xAdj = x + xChange;
        int yAdj = y + yChange;

        if (xAdj < 0 || xAdj >= (int)Config::GetInstans().simSize.x)
return;
        if (yAdj < 0 || yAdj >= (int)Config::GetInstans().simSize.y)
return;

        auto adjIndex = getCellIndex(xAdj, yAdj);
        auto& otherCreature = m_creatures[adjIndex];

        thisCreature.update();
        switch (thisType) {
            case CreatureType::Predator:
                updatePredator(thisCreature, otherCreature);
                break;

            case CreatureType::Prey:
                updatePrey(thisCreature, otherCreature);
                break;

            default:
                break;
        }

        setCellColour(x, y, m_creatures[index].getColour());
    });
}

void PredatorAndPrey::updatePredator(Creature & thisCreature,
Creature & otherCreature)
{
    if (thisCreature.getHealth() <= 0) {
        m_predatorCount--;
        thisCreature.setType(CreatureType::Nothing);
        return;
    }

    auto otherType = otherCreature.getType();

    switch (otherType) {
        case CreatureType::Prey:
            m_preyCount--;
            m_predatorCount++;
            otherCreature.setType(CreatureType::Predator);
            thisCreature.heal(otherCreature.getHealth());
            break;

        case CreatureType::Predator:
            break;

        case CreatureType::Nothing:
            thisCreature.move(otherCreature);
            break;
        }
}

void PredatorAndPrey::updatePrey(Creature & thisCreature,
Creature & otherCreature)
```

```
{
   auto otherType = otherCreature.getType();

   bool reproduce = false;
   if (thisCreature.getHealth() >= Creature::MAX_HEALTH)  {
      thisCreature.setHealth(10);
      reproduce = true;
   }

   switch (otherType) {
      case CreatureType::Prey:
         break;

      case CreatureType::Predator:
         break;

      case CreatureType::Nothing:
         if (reproduce) {
            m_preyCount++;
            thisCreature.reproduce(otherCreature);
         }
         else {
            thisCreature.move(otherCreature);
         }
         break;

   }
}
```

FPSCounter.cpp:

```
#include "FPSCounter.h"

#include "ResourceHolder.h"

#include <iostream>

FPSCounter::FPSCounter()
{
   m_text.move            (10, 20);
   m_text.setFillColor    (sf::Color::White);
   m_text.setFont         (ResourceHolder::get().fonts.get("arial"));
   m_text.setCharacterSize (18);
   m_text.setOutlineColor  (sf::Color::Black);
   m_text.setOutlineThickness(2);
}


void FPSCounter::update()
{
   m_frameCount++;

   if (m_delayTimer.getElapsedTime().asSeconds() > 0.2)
   {
      m_fps = m_frameCount / m_fpsTimer.restart().asSeconds();
      m_frameCount = 0;
      m_delayTimer.restart();
   }
}

void FPSCounter::draw(sf::RenderTarget& renderer)
{
   m_text.setString("FPS: " + std::to_string((int)m_fps));
   renderer.draw(m_text);
}
```

menu.cpp:

```
#include "menu.h"

#include <fstream>
void loadConfig() // завантаження параметрів з файлу
{
      std::ifstream inFile("Config.txt");
      if (!inFile.is_open()) { // якщо не вдалося відкрити файл
            std::cout << "Невдалося завантажити config,
встановлення налаштувань за замовчуванням\n";
            Config::GetInstans().fps = { 30 };
            Config::GetInstans().windowSize = { 1280, 720 };
            Config::GetInstans().cellSize = { 2 };
            Config::GetInstans().WarriorsAtStart = { 50 };
            return;
      }
      std::string line;
      while (std::getline(inFile, line)) { // явщо вдалося
            if (line == "fps") {
                  inFile >> Config::GetInstans().fps;
                  if (Config::GetInstans().fps < 10 ||
Config::GetInstans().cellSize ==
isalpha(Config::GetInstans().cellSize))
                  Config::GetInstans().fps = { 30 };
            }
            else if (line == "winx") {
                  inFile >> Config::GetInstans().windowSize.x;
                  if (Config::GetInstans().windowSize.x > 1920 ||
Config::GetInstans().windowSize.x < 100 ||
Config::GetInstans().cellSize ==
isalpha(Config::GetInstans().cellSize))
                        Config::GetInstans().windowSize.x = { 1280
};
            }// діагональ вікна
            else if (line == "winy") {
                  inFile >> Config::GetInstans().windowSize.y;
                  if (Config::GetInstans().windowSize.y > 1080 ||
Config::GetInstans().windowSize.y < 100 ||
Config::GetInstans().cellSize ==
isalpha(Config::GetInstans().cellSize))
                        Config::GetInstans().windowSize.y = { 720
};
            }// вертикаль вікна
            else if (line == "cellsize") {// розмір клітини
                  inFile >> Config::GetInstans().cellSize;
                  if (Config::GetInstans().cellSize > 30 ||
Config::GetInstans().cellSize < 2 || Config::GetInstans().cellSize
== isalpha(Config::GetInstans().cellSize))
                        Config::GetInstans().cellSize = { 2 };
            }
            else if (line == "WarriorsAtStart") {
                  inFile >> Config::GetInstans().WarriorsAtStart;
                  if (Config::GetInstans().WarriorsAtStart > 1000 )
Config::GetInstans().WarriorsAtStart = { 50 };
            }
      }
      Config::GetInstans().init(); // ініціяція параметрів
}

void menu() {
      RenderWindow window({ 1280, 720 }, L"Дипломна
робота", sf::Style::Default);
      Texture menuBackground, about;
      Text Question, Menu1, Menu2, Menu3, Menu4, About, Exit,
TEXT;

      std::wstring question = L"Який клітинний автомат ви
хочете запустити?";
      Question.setFont(ResourceHolder::get().fonts.get("arial"));
      Question.setString(question);
      Question.setCharacterSize(26);
      Question.setStyle(Text::Bold);
      Question.setFillColor(Color::Red);
      Question.setPosition(100, 40);

      std::wstring menU1 = L"Життя";
      Menu1.setFont(ResourceHolder::get().fonts.get("arial"));
      Menu1.setString(menU1);
      Menu1.setCharacterSize(24);
      Menu1.setStyle(Text::Bold);
      Menu1.setPosition(100, 100);

      std::wstring menU2 = L"Мурахи";
      Menu2.setFont(ResourceHolder::get().fonts.get("arial"));
      Menu2.setString(menU2);
      Menu2.setCharacterSize(24);
```

```
	Menu2.setStyle(Text::Bold);
	Menu2.setPosition(100, 150);

	std::wstring menU3 = L"Хижак та здобич";
	Menu3.setFont(ResourceHolder::get().fonts.get("arial"));
	Menu3.setString(menU3);
	Menu3.setCharacterSize(24);
	Menu3.setStyle(Text::Bold);
	Menu3.setPosition(100, 200);

	std::wstring menU4 = L"Арена";
	Menu4.setFont(ResourceHolder::get().fonts.get("arial"));
	Menu4.setString(menU4);
	Menu4.setCharacterSize(24);
	Menu4.setStyle(Text::Bold);
	Menu4.setPosition(100, 250);

	std::wstring AbouT = L"Нотатка";
	About.setFont(ResourceHolder::get().fonts.get("arial"));
	About.setString(AbouT);
	About.setCharacterSize(24);
	About.setStyle(Text::Bold);
	About.setPosition(100, 300);

	std::wstring exiT = L"Закрити програми";
	Exit.setFont(ResourceHolder::get().fonts.get("arial"));
	Exit.setString(exiT);
	Exit.setCharacterSize(24);
	Exit.setStyle(Text::Bold);
	Exit.setPosition(100, 350);

	TEXT.setFont(ResourceHolder::get().fonts.get("arial"));
	TEXT.setCharacterSize(14);
	TEXT.setStyle(Text::Bold);
	TEXT.setPosition(730, 45);

	{
		WindowHandle wHandle;
		wHandle = window.getSystemHandle();
		HCURSOR Cursor = LoadCursor(NULL,
IDC_ARROW);
		SetCursor(Cursor);
		SetClassLongPtr(wHandle, GCLP_HCURSOR,
reinterpret_cast<LONG_PTR>(Cursor));
	}

	menuBackground.loadFromFile("images/Bg.jpg");
	about.loadFromFile("images/About.png");

	Sprite  menuBg(menuBackground), abouT(about);
	bool isMenu = 1;
	int menuNum = 0;

	menuBg.setPosition(0, 0);
	abouT.setPosition(0, 0);

	bool c = false;
	//----------МЕНЮ----------
	while (window.isOpen())
	{
		while (isMenu)
		{
			sf::Event EvM;
			while (window.pollEvent(EvM))
			{
				if (EvM.type == sf::Event::Closed)
window.close();
				loadConfig(); //завантаження даних, що
можна змінити
				Menu1.setFillColor(Color::Green);
				Menu2.setFillColor(Color::Green);
				Menu3.setFillColor(Color::Green);
				Menu4.setFillColor(Color::Green);
				About.setFillColor(Color::Cyan);
				Exit.setFillColor(Color::Magenta);
				TEXT.setFillColor(Color::Cyan);
				menuNum = 0;
				window.clear(Color(129, 181, 221));
				if (IntRect(100, 100, 300,
50).contains(Mouse::getPosition(window))) {
Menu1.setFillColor(Color::Blue); menuNum = 1; }
				else if
(Keyboard::isKeyPressed(Keyboard::Num1)) { menuNum = 1; } //
else if використовуються для випадку відсутності, чи не
бажанні використовувати мишу
				if (IntRect(100, 150, 300,
50).contains(Mouse::getPosition(window))) {
Menu2.setFillColor(Color::Blue); menuNum = 2; }
				else if
(Keyboard::isKeyPressed(Keyboard::Num2)) { menuNum = 2; } //
вони викличуть нотатки до первної моделі КА
				if (IntRect(100, 200, 300,
50).contains(Mouse::getPosition(window))) {
Menu3.setFillColor(Color::Blue); menuNum = 3; }
				else if
(Keyboard::isKeyPressed(Keyboard::Num3)) { menuNum = 3; }
				if (IntRect(100, 250, 300,
50).contains(Mouse::getPosition(window))) {
Menu4.setFillColor(Color::Blue); menuNum = 4; }
				else if
(Keyboard::isKeyPressed(Keyboard::Num4)) { menuNum = 4; }
				if (IntRect(100, 300, 300,
50).contains(Mouse::getPosition(window))) {
About.setFillColor(Color::Blue); menuNum = 5; }
				if (IntRect(100, 350, 300,
50).contains(Mouse::getPosition(window))) {
Exit.setFillColor(Color::Blue); menuNum = 6; }

				if (Mouse::isButtonPressed(Mouse::Left)) //
клік мишою по відповидному пункту меню спризведе його
виклик
				{
					if (menuNum == 1) {
run<GameOfLife>(); }
					if (menuNum == 2) { run<AddAnt>();
}
					if (menuNum == 3) {
run<PredatorAndPrey>(); }
					if (menuNum == 4) { Game game;
game.run(); }
					if (menuNum == 5) {
						window.draw(abouT);
window.display(); while
(!Keyboard::isKeyPressed(Keyboard::Escape));
					}
					if (menuNum == 6) { window.close();
isMenu = false; }
				}
				// далі випадок, якщо користувач не має
змоги, чи не хоче використовувати мишу, натиск
конбінування двух кнопок клавіатури спризведе до виклику
певного пунку меню, що й спризведе його виклик
				else if
(Keyboard::isKeyPressed(sf::Keyboard::Num1) &&
Keyboard::isKeyPressed(sf::Keyboard::LShift)) {
run<GameOfLife>(); }
				else if
(Keyboard::isKeyPressed(sf::Keyboard::Num2) &&
Keyboard::isKeyPressed(sf::Keyboard::LShift)) { run<AddAnt>();
}
				else if
(Keyboard::isKeyPressed(sf::Keyboard::Num3) &&
Keyboard::isKeyPressed(sf::Keyboard::LShift)) {
run<PredatorAndPrey>(); }
				else if
(Keyboard::isKeyPressed(sf::Keyboard::Num4) &&
Keyboard::isKeyPressed(sf::Keyboard::LShift)) { Game game;
game.run(); }
				else if
(Keyboard::isKeyPressed(sf::Keyboard::Tab) &&
Keyboard::isKeyPressed(sf::Keyboard::LShift)) {
					window.draw(abouT);
window.display(); while
(!Keyboard::isKeyPressed(Keyboard::Escape)); }
```

```
                    else if
(Keyboard::isKeyPressed(sf::Keyboard::Q) &&
Keyboard::isKeyPressed(sf::Keyboard::LShift)) { window.close();
isMenu = false;  }

                    switch (menuNum)
                    {
                    case 1:

      TEXT.setString(std::wstring(L"Моделювання гри 'Життя' -
це моделювання поведінки величезної\nмножини деяких
персонажів у константному середовищі, тобто без\nзовнішніх
подразників."));
                        break;
                    case 2:

      TEXT.setString(std::wstring(L"Моделювання гри 'Мурахи'
- це моделювання поведінки від кількох\nдо багатьох різних
персонажів, що тільки при перетині впливають\nодин на
одного."));
                        break;
                    case 3:

      TEXT.setString(std::wstring(L"Моделювання гри 'Хижак –
здобич' - це моделювання поведінки\nдвох архітипів, двох
видів множин персонажів, де вони живуть\nмайже як у грі
життя, але для кожного виду є зовнішній подразник у\nвигляді
іншого виду, тобто моделюється поведінко
массових\nперсонажів 'здобич' котрі є ціллю персонажів
'хижак', які всіляко\nзнищують перших, за рахунок котрих й
існують."));
                        break;
                    case 4:

      TEXT.setString(std::wstring(L"Моделювання гри 'Арена' -
це моделювання поведінки, котра\nзонснована на грі 'Мурахи',
де є від кількох до багатьох різних\nперсонажів, та грі 'Хижак
– здобич', де для кожного персонажу інший\nперсонаж, що не
відноситься до тогож типу/виду, є подразником."));
                        break;
                    default:
                        TEXT.setString(std::wstring(L""));
                        break;
                    }
                    window.draw(menuBg);
                    window.draw(Question);
                    window.draw(Menu1);
                    window.draw(Menu2);
                    window.draw(Menu3);
                    window.draw(Menu4);
                    window.draw(About);
                    window.draw(Exit);
                    window.draw(TEXT);
                    window.display();
                }
            }
        }
}
```

Random.cpp:

```
#include "Random.h"

#include <ctime>

Random& Random::get()
{
   static Random r;
   return r;
}

Random::Random()
{
   m_rng.seed ((unsigned)std::time(nullptr));
}

int Random::intInRange(int low, int high)
{
   std::uniform_int_distribution<int> dist(low, high);
   return dist(m_rng);
}

float Random::floatInRange(float low, float high)
{
   std::uniform_real_distribution<float> dist(low, high);
   return dist(m_rng);
}

int Random::getRandomInt(int min, int max) {
   return min + rand() % ((max + 1) - min);
}
void Random::reseed() {
   srand(time(NULL));
}

sf::Vector3i Random::getRandomColor() {
   sf::Vector3i random_color(0, 0, 0);
   while (random_color.x + random_color.y + random_color.z <
250)
   {
      random_color = sf::Vector3i(getRandomInt(0, 255),
getRandomInt(0, 255), getRandomInt(0, 255));
   }
   return random_color;
}

sf::Vector2i Random::getRandomPosition(int minx, int maxx, int
miny, int maxy) {
   return sf::Vector2i(getRandomInt(minx, maxx),
getRandomInt(miny, maxy));
}

sf::Vector2i Random::generateMovePos(sf::Vector2i oldPosition)
{
   int x_val = 2 - getRandomInt(1, 3);
   int y_val = 2 - getRandomInt(1, 3);
   while (x_val == 0 && y_val == 0)
   {
      x_val = 2 - getRandomInt(1, 3);
      y_val = 2 - getRandomInt(1, 3);
   }
   return sf::Vector2i(oldPosition.x + x_val, oldPosition.y +
y_val);
}

int Random::getMutation() {
   int chance = getRandomInt(0, 10);
   if (chance == 4)
   {
      return (2 - getRandomInt(1, 3));
   }
   else
   {
      return 0;
   }
}
```

ResourceHolder.cpp:

```
#include "ResourceHolder.h"

ResourceHolder& ResourceHolder::get()
{
   static ResourceHolder holder;
   return holder;
}

ResourceHolder::ResourceHolder()
:  fonts          ("Fonts", "ttf")
,  textures       ("Textures", "png")
{ }
```

Win32.cpp:

```
#include "Win32.h"

#ifdef _WIN32

   namespace
   {
      WORD _default;
      bool _first = true;
   }

   std::ostream& operator<< (std::ostream& stream, TextColour t)
   {
      if(_first)
      {
         _first = false;
         CONSOLE_SCREEN_BUFFER_INFO Info;

GetConsoleScreenBufferInfo(GetStdHandle(STD_OUTPUT_HA
NDLE), &Info);
         _default = Info.wAttributes;
      }

      if(t == TextColour::Default)

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDL
E), _default);
      else

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDL
E), (WORD)t);
      return stream;
   }

#endif // __WIN32
```