МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

на здобуття освітньо-кваліфікаційного рівня «бакалавр»
(освітньо-кваліфікаційний рівень)

на тему Розробка програми для тренування логічного мислення. Реалізація логічних задач
Program development for training logical thinking. Implementation of logical tasks

Виконав: студент денної форми навчання
спеціальності 123 – Комп'ютерна інженерія
(шифр і назва напряму підготовки, спеціальності)
Шатілов Валерій Олександрович
(прізвище, ім'я, по-батькові)

Керівник канд фіз.-мат. наук. Антоненко О.С.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент д-р техн. наук, проф. Малахов Є.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№___ від «___»_____ 2020 р.
Завідувач кафедри
_____ Є.В. Малахов
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___
протокол № __ від «___» ____ 2020 р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)
Голова ЕК
_____ Н.Ф. Казакова
(підпис) (прізвище, ініціали)

Одеса-2021

АНОТАЦІЯ

Робота присвячена проектуванню і розробці програми для тренування когнітивних здібностей людини.

Мета даної роботи полягає в розробці підсистеми кімнат-задач системи тренування логічного мислення і інтегрування з підсистемою процедурно-генерованого оточення.

За допомогою Unity на мові C# виконані наступні задачі:

- виконати аналіз існуючих методів тренування когнітивних здібностей людини та програмно реалізованих тренажерів для ума;
- спроектувати підсистему кімнат-задач;
- зробити програмну реалізацію кімнат-задач;
- інтегрувати кімнат-задач з підсистемою процедурно-генеруємого оточення;
- протестувати систему, яка складається з двох підсистем: процедурної генерації оточення та кімнат-задач.

ABSTRACT

The graduate work is devoted to the design and development of an application for training human cognitive abilities.

The purpose of this work is to develop a subsystem of task-rooms of the system for training logical thinking and integration with the subsystem of the procedurally generated environment.

The following tasks are performed by using Unity with C#:

- analyze the existing methods of training human cognitive abilities and software implementation of mind simulators;
- design a subsystem of task-rooms;
- make a software implementation of task-rooms;
- integration of task-rooms with a procedurally generated environment subsystem;
- test a system that consists of two subsystems: procedural environment generation and task-rooms.

АННОТАЦИЯ

Работа посвящена проектированию и разработке приложения для тренировки когнитивных способностей человека.

Цель данной работы заключается в разработке подсистемы комнат-задач системы тренировки логического мышления и интегрирования с подсистемой процедурно-генерируемого окружения.

С помощью Unity на языке C # выполнены следующие задачи:

- выполнить анализ существующих методов тренировки когнитивных способностей человека и программной реализации тренажеров для ума;
- спроектировать подсистему комнат-задач;
- сделать программную реализацию комнат-задач;
- интегрирование комнат-задач с подсистемой процедурно-генерируемого окружения;
- протестировать систему, которая состоит из двух подсистем: процедурной генерации окружения и комнат-задач.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Тренувальник логічного мислення - ТЛМ

Кваліфікаційна робота - КР

Кімната з логічними та математичними задачами - КЛМЗ

UI - інтерфейс користувача

ВСТУП

Тренувальник логічного мислення - це деякий тренажер, який сприяє підвищенню активності головного мозку. В результаті тренування скорочується час, необхідний мозку для вирішення різноманітних задач. Та тренування дозволяє підвищити цілісність сприйняття навколишнього середовища. Проекти в сфері ТЛМ або маленькі, або працюють некоректно та малоефективні. В цілому такі тренажери на достатньо високому рівні застосовуються для іспитів людей у спеціальному середовищі у лабораторіях.

Актуальність кімнат-задач полягає у доступності розвитку своїх розумових здібностей. Маючи доступний ресурс у вигляді деякого виду гри, підвищую інтерес людини у саморозвитку. Тренування сприяє вибудовуванню моделі окремих процесів та явищ. Володіння такого роду якостями мислення як критичність, доказовість, абстрактність та лаконізм потрібні людині у будь яких сферах діяльності.

Систематичне використання ТЛМ також дозволяє більш тривалий час не втомлюючись підтримувати мозок у активному стані.

Користування ТЛМ має велике значення для людини на всіх етапах його життя. Тому це корисно для школярів, студентів та працевлаштованих людей, особливо для людей, діяльність яких пов’язана з розумовою діяльністю.

Мета даної кваліфікаційної роботи полягає у розробці підсистеми кімнат-задач системи тренувальника логічного мислення та інтегрування з підсистемою процедурно-генеруємого оточення.

Концепція ТЛМ складається з процедурно-генерємого оточення, в якому генеруються кімнати-задачі у випадковому порядку. Де ці кімнати задачі - невідомо. Користувач повинен їх знайти. Кожна кімната-задача містить якусь головоломку, яку повинен вирішити користувач. Таким чином покращувати свої когнітивні здібності.

ТЛМ повинен дотримуватись наступних вимог:

- різноманітність середовища;
- різноманітність завдань, які базуються на різних логічних методах тренування мозку;
- цікавість;
- градація складності;
- результати тренувань (Статистика).

Тому задачами даної роботи є:

- виконати аналіз існуючих методів тренування когнітивних здібностей людини та програмно реалізованих тренажерів для ума;
- спроектувати підсистему кімнат-задач;
- зробити програмну реалізацію кімнат-задач;
- інтегрувати кімнат-задач з підсистемою процедурно-генеруємого оточення;
- протестувати систему, яка складається з двох підсистем: процедурної генерації оточення та кімнат-задач.

1 ОГЛЯД ОБЛАСТІ ДОСЛІДЖЕННЯ

Треба почати з дослідження когнітивного розвитку людини та в вчасності образно-логічного розуміння. Після чого проектування системи, яке базується на цьому дослідженні, буде мати сенс.

Когнітивний розвиток (від англ. Cognitive development) - розвиток всіх видів розумових процесів, таких як сприйняття, пам'ять, формування понять, рішення задач, уяву та логіку. Теорія когнітивного розвитку була розроблена швейцарським філософом і психологом Жаном Піаже. [1]

Образне-логічне розуміння - різні розумові процеси так званого “образного” вирішення задач, яке передбачає візуальне уявлення ситуації та оперування образами складаючи її предметів. Даний вид розумової діяльності людини формується починаючи з 1,5 року. [2]

Для розвитку цих якостей людини можна розробити процедурно-генеруєме оточення та кімнати з логічними та математичними задачами. У рамках цієї КР дослідимо побудову кімнат з логічними та математичними задачами.

Існує чи мало задач на логіку, таких як: задачі на пам’ять, задачі з матрицями, задачі на знаходження зайвого, різноманітні задачі Шульте, сортувальники тощо(див. Рисунок 1.1 - 1.5). Більшість з них реалізуються класичними алгоритмами на пошук та сортування.

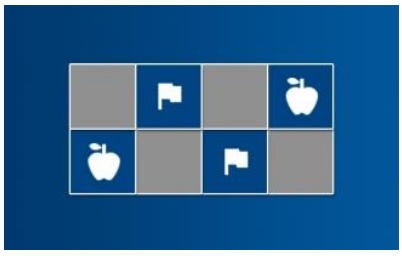

Рисунок 1.1 - Задача на пам’ять

Рисунок 1.2 - Пошук зайвого

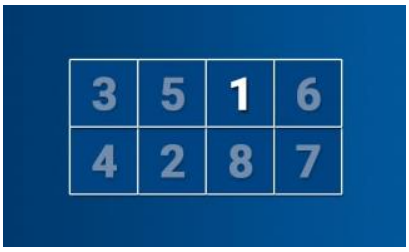

Рисунок 1.3 - Таблиця Шульте

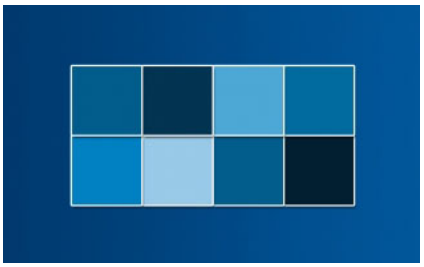

Рисунок 1.4 - Шульте – Колір

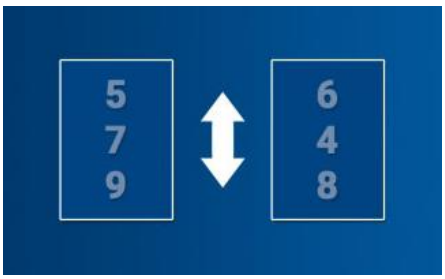

Рисунок 1.5 - Сортувальник чисел

Кожна з цих задач розвиває когнітивні якості людини, такі як: короткочасна пам’ять, спостережливість, образне мислення, швидкість мислення тощо. [8]

Для покращення процесу вирішення цих задач, їх треба програмно реалізувати та інтегрувати у генеруєме оточення.

1.1 Етапи формування випробування

Побудова КЛМЗ починається з формування та деталізації етапів випробування.

Тобто:

- ідея;
- кінцевий результат, до якого повинен дістатися випробуваний;
- шляхи, якими випробуваний може дістатися результату;
- проектування передбачуваної поведінки випробуваного;
- оцінка складності;
- завдяки яким методам випробовуваний повинен вирішити певний етап задачі.

Всі ці етапи повинні бути формалізованими та деталізованими.

На рисунку 1.6 можна побачити послідовність складання випробування. Кожен крок випробуваного повинен бути передбачуваним для подальшого аналізу та формування результатів випробування. Також випробування повинне мати кінцеву ціль та мету, яка націлена на підвищення якості когнітивного мислення.

Рисунок 1.6 – Етапи формування та деталізації задачі

1.2 Ідея

Ідея випробування повинна базуватись на уявленні того, як влаштовані процеси у головному мозку людини та її психології.

Варіацій, класифікацій логічних задач багато і вони націлені на випробування різних частин мозку. Але всі вони також направлені на розвиток когнітивних якостей. За когнітивні якості відповідає кора великого мозку (див. рисунок 1.7).

1.3 Кінцевий результат

Після випробування потрібно скласти статистичні данні для користувача, ґрунтуючись на використаному часі, складності та чи була вирішена задача.

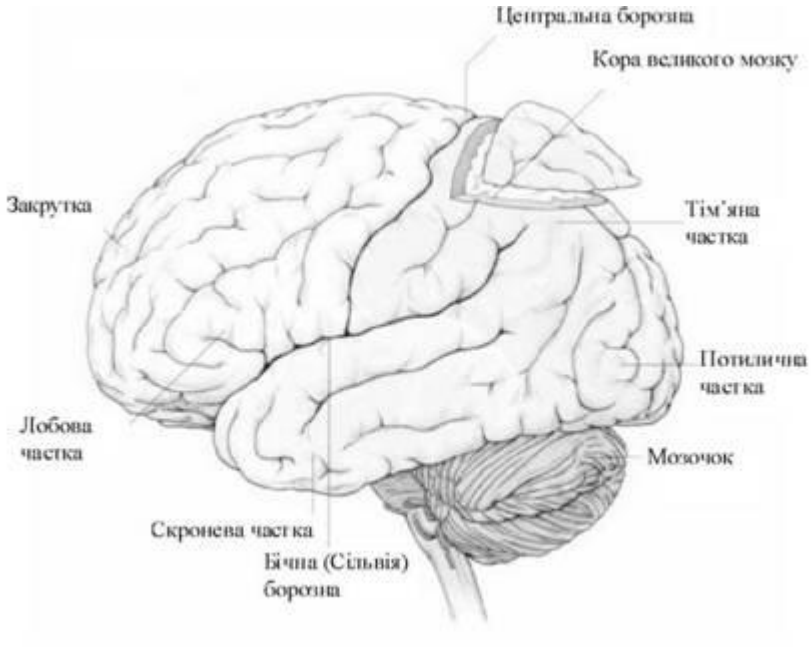

Рисунок 1.7 – Анатомія головного мозку

Для успішного вирішення задачі та встигнути вчасно потрібно використовувати прийоми логічного мислення. Чим більше прийомів використовується, тим менший час потрібен на вирішення задачі.

На основі статистичних даних користувач зможе здійснити аналіз.

Після декількох сесій користувач зможе синтезувати та порівняти свої результати.

Завдяки різноманітності завдань можна класифікувати результати та виявити проблеми. Або зрозуміти, якої з якостей когнітивного мислення потрібно дати більшу увагу.

Наприкінці потрібно узагальнити весь аналіз (див. Рисунок 1.8).

Вміння робити цей особистий аналіз також необхідно користувачу для підвищення когнітивних здібностей. З таким аналізом легше обрати напрям для особистого розвитку.

Рисунок 1.8 - Етапи аналізу випробування

1.4 Шляхи вирішення задачі

Мета у всіх випробувань своя та єдина, але шляхи, якими можна дістатись цієї мети неоднозначні. Випробування потребуюсь вміння узагальнювати, класифікувати та оптимізувати. Одне й те ж саме випробування можна завершити швидко, повільно або не завершити взагалі.

Узагальнення - це знаходження загального в предметах і явищах. Знаходження загального включає в себе зіставлення предметів, вичленення спільних ознак в кожному з даних предметів і об'єднання останніх за цими ознаками. [3]

У процесі навчання завдання набувають більш складний характер: в результаті виділення відмітних і спільних ознак вже кількох предметів, люди намагаються розбити їх на групи. Тут необхідна така операція мислення як класифікація. [3]

Оптимізація — це сукупність процесів, спрямованих на модернізацію та поліпшення існуючих методів досягнення бажаного результату. Оптимізацію можна застосовувати практично в будь-якій сфері діяльності. [4]

1.5 Проектування поведінки користувача

Проектування поведінки користувача дуже складний процес. Треба спрогнозувати усі деталі, які можуть підштовхнути користувача до вирішення задачі та уникнути ймовірність того, що кімната може бути непрохідною.

Одним з методів такого проектування це тест кімнати, пошук ймовірних багів. Пошук очевидних слабостей кімнати, які можна експлуатувати для швидкого проходження. Моделювання поведінки користувача.

Після кожного тесту треба робити прогноз, який буде свідчити або не свідчити про те, що кімната готова до релізу, працює справно з технічної точки зору та виконує свої функції для досягання мети.

1.6 Логічні методи

Потрібно довести кімнату-задачу до стану, коли простір достатньо інформативний, що б застосування логічних методів як морфологічний аналіз та АРВЗ було мінімально можливим.

Морфологічний аналіз (морфологічний ящик, аналіз кола проблем) - метод систематизації перебору варіантів усіх теоретично можливих рішень, заснований на аналізі структури об'єкта і наступному систематизованому отриманні їх поєднань (комбінуванні). [5]

Алгоритм рішення винахідницьких задач (АРВЗ) - покрокова послідовність дій щодо виявлення і вирішення протиріч, тобто рішенням винахідницьких завдань.

АРВЗ включає, власне програму, інформаційне забезпечення, що базується з інформаційного фонду, методи управління психологічними факторами, які входять складовою частиною в методи розвитку творчої уяви. [6]

1.7 Оцінка складності

Оцінка складності кімнати-задачі повинно базуватися на наступних параметрах:

- мінімальний та максимальний час необхідний на завершення задачі;
- тип кімнати, який базується на обраній складності на початку сесії;
- варіант кімнати;
- коефіцієнт психологічної напруги (це коефіцієнт заданий розробником по відношенню до інших кімнат).

Якщо кімната надто складна, вона повертається до етапу шляхів вирішення задач.

1.8 Порівняння існуючих тренажерів

Порівняти тренажери для розвитку когнітивних здібностей людини можна тільки за складністю.

Кожний тренажер унікальний та спрямований на дослідження окремих якостей людини. Тому не існує явних алгоритмів для побудови тренажера. Можна реалізувати такий самий тренажер або застосувати алгоритміку цього тренажера у більш комплексному тренажері, який складається з декількох примітивів.

1.9 Узагальнення огляду області дослідження

Побудова КЛМЗ дуже важка задача, яка потребує широкого спектра знань та багатий досвід у складанні завдань, які повинні виконувати та вирішувати люди. Треба розуміти чи можливо завершити це випробування, або навпаки чи не занадто воно просте. Який коефіцієнт корисності у цієї кімнати. Треба постійно аналізувати та тестувати тренажери.

2 ПРОЕКТУВАННЯ ТЛМ ТА КІМНАТ-ЗАДАЧ

Робота присвячена побудові програмної реалізації тренувальника логічного мислення за допомогою Unity на мові C#.

Unity - міжплатформене середовище розробки комп'ютерних ігор, розроблена американською компанією Unity Technologies.

Основними перевагами Unity є наявність візуальної середовища розробки, міжплатформеній підтримки і модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами і труднощі при підключенні зовнішніх бібліотек.

Можливості Unity:

- простий Drag&Drop інтерфейс;
- налаштування плагінів;
- налагодження додатка у редакторі;
- підтримка мови програмування C#;
- розрахунки фізики виконуються за допомогою PhysX від NVIDIA;
- підтримка нових версій DirectX(на цей час підтримується DX 12);
- проект ділиться на окремі сцени. Кожна сцена має свій набір об’єктів;
- велика кількість асетів для розробників.

2.1 Загальні вимоги до додатка та кімнат-задач в часності

Користувач повинен мати можливість створювати сессію. Обирати складність цієї сесії щоб змінювати складність задач.

Після старту користувачу почне генеруватися унікальне середовище. В цьому середовищі генеруються кімнати з випробуваннями.

Користувач повинен досліджувати навколишнє середовище, вирішувати задачі та продовжувати свій шлях. Це зроблено для того, що б користувач міг мати час на відпочинок, дивуючись навколишнім середовищем.

У будь якій кімнаті з задачею користувач має можливість взаємодіяти с оточенням. Ця взаємодія друкується правилами кімнати з задачею.

Користувач повинен мати можливість у будь який час завершити сесію та закрити додаток.

Можливість кооперативної сесії опціонально. Будь який додаток з можливістю робити щось з іншою людиною покращує досвід від витраченого часу. Тому наявність такої функції є непоганим доповненням.

Таке загальне проектування демонструє Рисунок 2.1

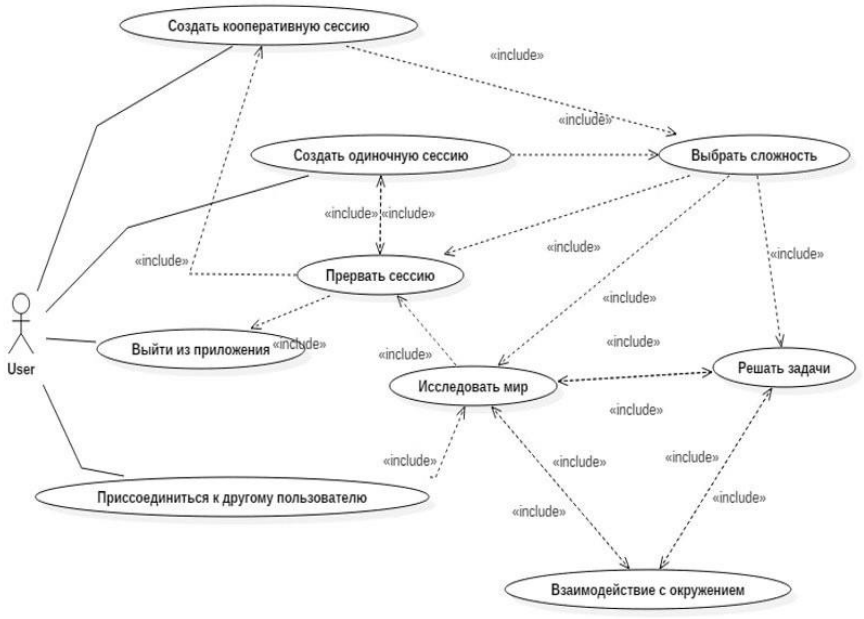

Рисунок 2.1 - Загальна USE Case діаграма ТЛМ

Через те, що для кожної кімнати-задачі необхідна своя діаграма, у разі прикладу можна розглянути діаграму задачі судоку (Рисунок 2.2).

Користувач має можливість взаємодіяти з дошкою для судоку. Після взаємодії відкривається характерний UI у вигляді дошки для судоку. Користувач може закрити цей UI, таким чином завершити взаємодію з дошкою для судоку. Поки взаємодія триває можна міняти значення у клітинках. Таким чином вирішити судоку або у разі помилки - повернути судоку у початковий стан.

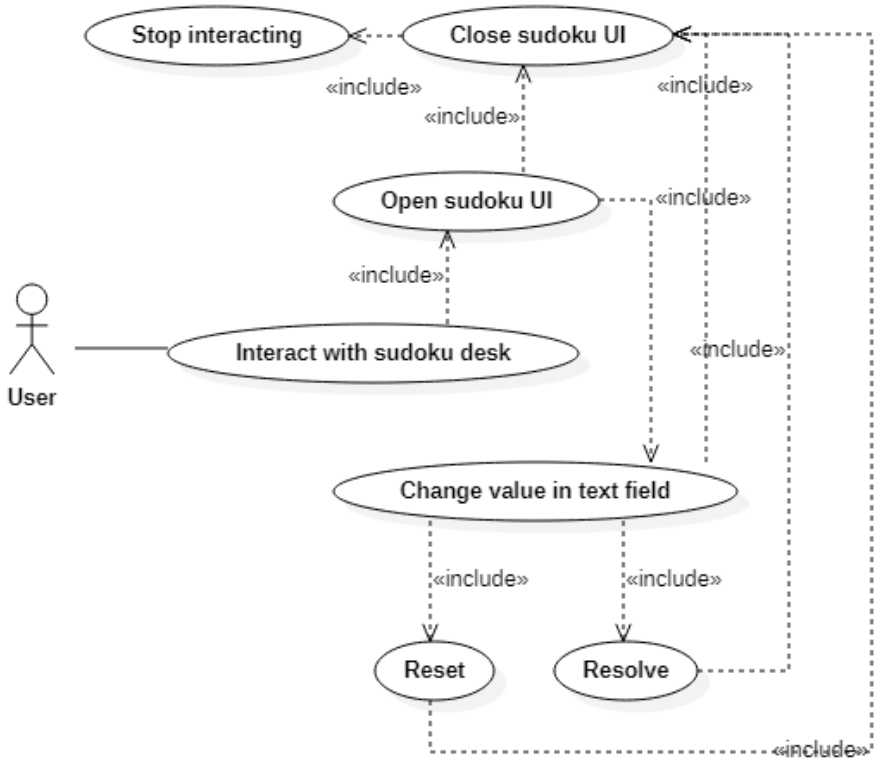

Рисунок 2.2 – USE case діаграма задачі судоку

2.2 Проектування загальних процесів додатку та кімнат-задач вчасності

Планування процесів додатка це остання і складна задача перед реалізацією. Необхідно з’ясувати які об’єкти повинні створюватись, коли

видалятись, за допомогою яких методові викликається конструктор наступного об’єкта тощо.

На рисунку 2.3 зображено повний цикл життя додатку. Основними об’єктами додатку є:

- головне меню MainMenu;
- оточення World;
- персонаж Character;
- кімнати-задачі PuzzleRoom;
- таймер Timer.

Після запуску додатка першим чином створюється головне меню. Викликаються відповідні методи для налаштування параметрів для створення оточення.

Починається сесія. Об’єкт головного меню видаляється та створюється оточення, створюється персонаж, кімнати-задачі та починає йти час. Персонаж може викликати відповідні методи для вирішення задач.

Як і з USE case діаграмою, так і Sequence діаграма для кожної кімнати задачі унікальна.

Для прикладу розглянемо діаграму процесів для кімнати з задачею судоку (Рисунок 2.4). Основними класами кімнати є керуючий клас SudokuManager, клас інтерфейсу SudokuUI, клас SudokuValue для парсінгу та збереження числових значень.

Користувач повинен знайти цю головоломку та з’ясувати за якими принципами її можливо знайти швидше.

Персонаж повинен бути близько до задачі и дивитись на неї. Через праву кнопку миші користувач може взаємодіяти з дошкою для судоку. Викликається метод Interact(), який вже викликає методи для ініціалізації самої дошки та ініціалізації UI компонентів. UI підписується на подію у SudokuManager для того,

що б передавати значення у сам обробник для подальшої роботи з цими значенням. Після кожного введеного значення дошка повинна перевірятися на правильність. Якщо у дошку введено неправильне значення, це просто змінює стан результату, але користувачу це не повинно бути відомо. Тільки наприкінці заповнення дошки користувач дізнається упорався він із випробуванням, чи ні.

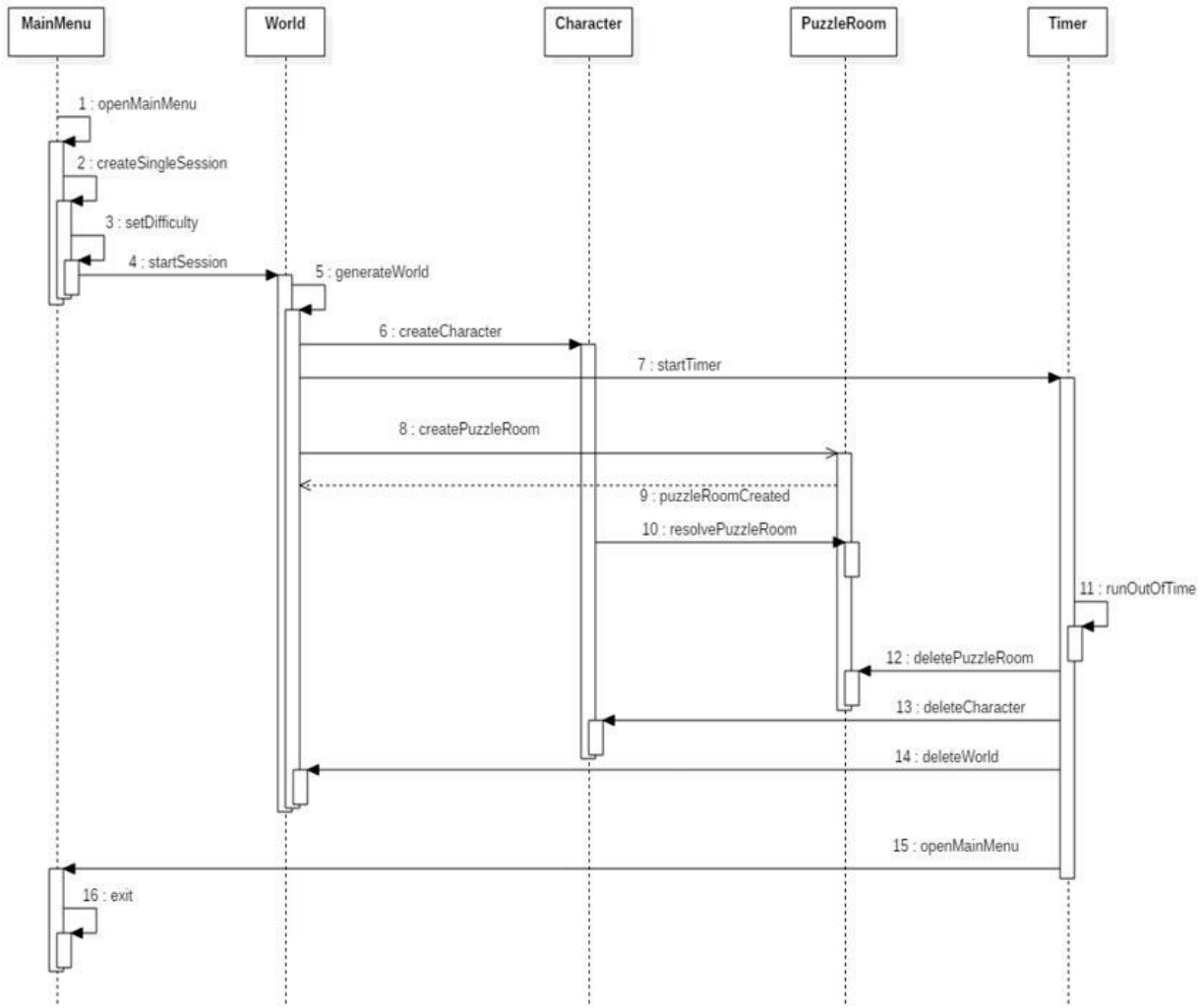

Рисунок 2.3 – Загальна Sequence діаграма процесів LogicalProcedure

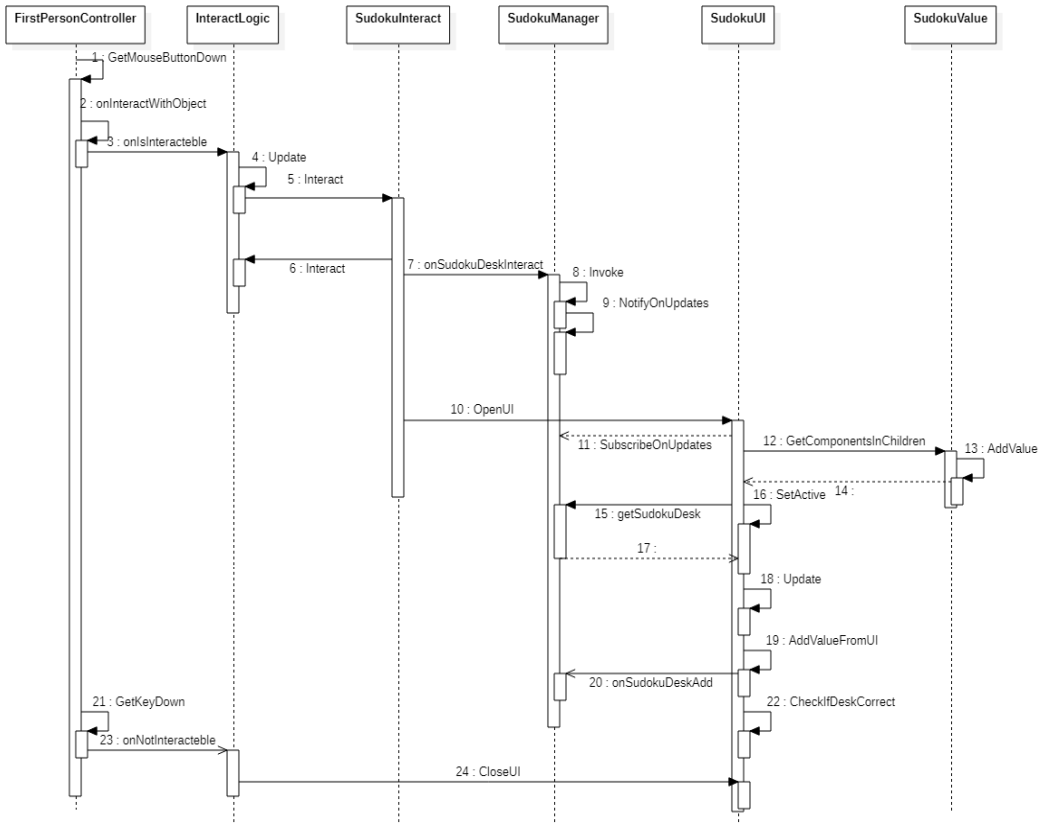

Рисунок 2.4 - Sequence діаграма судоку

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КІМНАТ-ЗАДАЧ

Після детального дослідження та проектування кімнта-задач, потрібно реалізувати ці самі кімнати-задачі.

Поділимо розробку на етапи:

1) Реалізація класу InteractibleLogic;
2) Розширити клас керування персонажем;
3) Реалізація адаптивного інтерфейсу користувача для задачі судоку;
4) Реалізація логіки для задачі судоку;
5) Інтегрування задачі судоку у процедурно-генеруєме оточення, розшири клас блочної генерації;
6) Загальне тестування.

Для реалізації кімнат-задач була розроблена система класів за допомогою мови програмування C#.

3.1 Реалізація класу InteractibleLogic

Клас InteractibleLogic це загальний клас для усіх об’єктів, з якими користувач моє взаємодіяти.

Для цього створюється C# скрипт, який унаслідується від класу MonoBihaviour (MonoBihaviour це клас Unity, який реалізує поведінку об’єкта). В цьому скрипті реалізовано, саме за якими правилами користувач може взаємодіяти з об’єктом. Розраховується дистанція до об’єкту, змінюється стан можливості взаємодії. Тобто почалась взаємодія чи взаємодія завершилась.

Щоб зрозуміти з яким об’єктом ми взаємодіємо треба повернути цей самий об’єкт. У кожного об’єкта розташованого на сцені Unity є колайдер, а колайдер це об’єкт фізики. Тому коли повернеться Raycast (див. Пункт 3.2), треба

застосувати метод GetComponent<InteractibleLogic>. Таким чином повертається об’єкт, поведінка якого описана класом InteractibleLogic.

Щоб об’єкт перейняв поведінку класу IntercatibleLogic, треба додати скрипт у компоненти об’єкту (див. Рисунок 3.1).

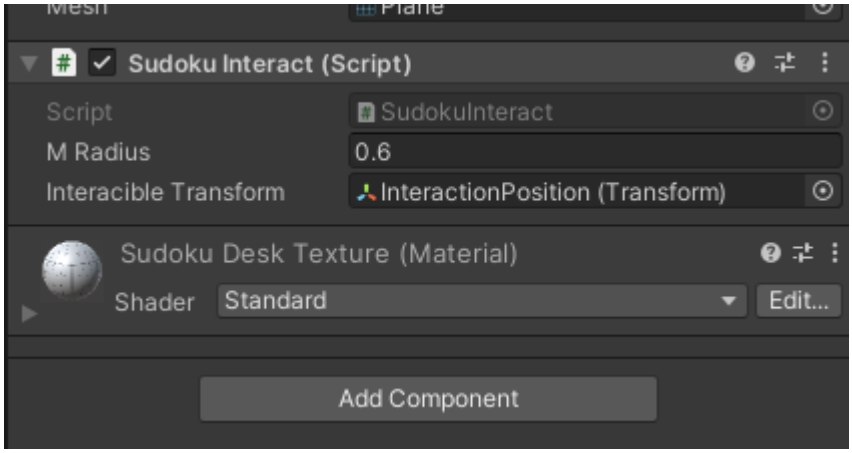

Рисунок 3.1 - Додання скрипту до компонентів об’єкту

На рисунку зображено скрипт SudokuInteract. Він унаслідується від InteractibleLogic. Це зроблено для того, що б додавати та перевизначити логіку класу взаємодії. Тому що логіка взаємодії може змінюватись в залежності від кімнати-задачі.

3.2 Розширення класу керування персонажем

У класі керування персонажем додаються нові клавіши керування та логіка, яка слідкує після натискання клавіши. Також додається логіка при взаємодії, а саме зміна режиму управління.

Додається дві події для клавіши RightMouse та клавіши E. RightMouse відповідає за початок взаємодії. E відповідає за завершення взаємодії.

Для розрахунку та передачі значень у клас InteractibleLogic використовується Physics.Raycast. Physics це клас, який відповідає за розрахунок фізики у Unity. Raycast це метод, який малює луч від центру камери та до тих пір,

поки не зіткнеться з об’єктом, та повертає значення типу float, що використовується для вимірювання відстані між камерою та об’єктом.

Якщо взаємодія успішна, то режим контролю змінюється. Пересування персонажа стає неможливим, з’являється курсор. На цьому етапі користувач може користуватися тільки мишкою.

3.3 Реалізація адаптивного інтерфейсу користувача для задачі судоку

До сцени додається Canvas. Це об’єкт, який являє собою абстрактний простір, в якій відбувається налаштування і відображення UI. Реалізація інтерфейса для судоку починається з додання двох панелей (UI елемент, який має вигляд прямокутника). Одна панель буде відповідати за відображення дошки для судоку, а інша панель застосовується для відображення повідомлення, що задача вирішена.

Як виглядає цей інтерфейс, можна подивитись на рисунку 3.2 та 3.3.

Інтерфейс повинен адаптуватись під зміни розмірів монітору ПК. Тому у Canvas, у розділі Canvas Scaler треба виставити UI Scale Mode у значення Scale With Screen Size (див. Рисунок 3.4)

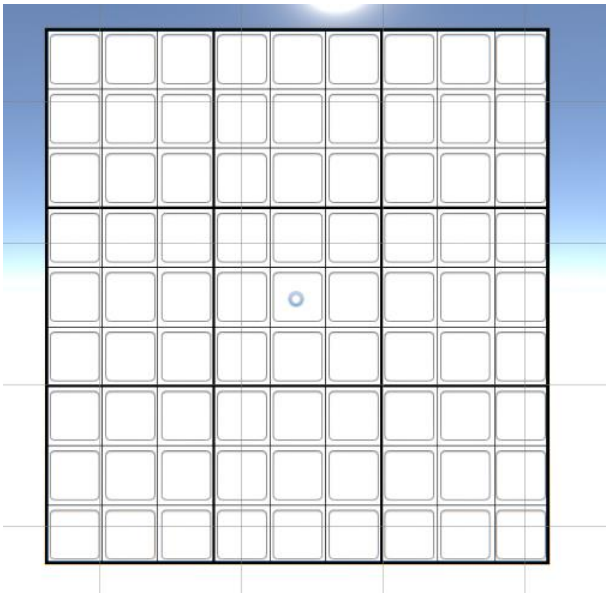

Рисунок 3.2 - UI дошки судоку

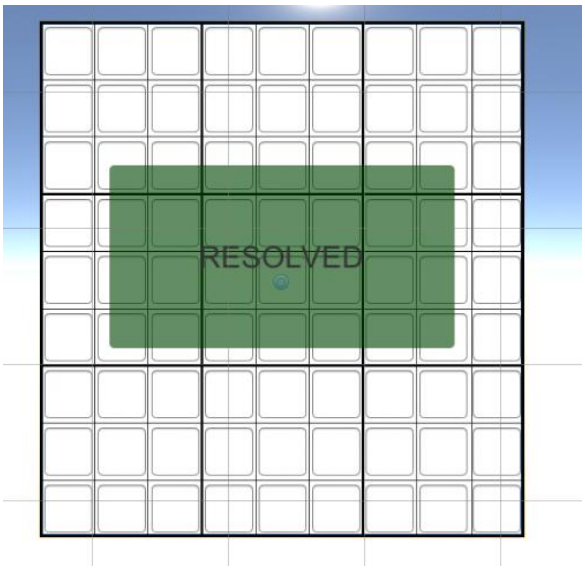

Рисунок 3.3 - UI Завершеної задачі

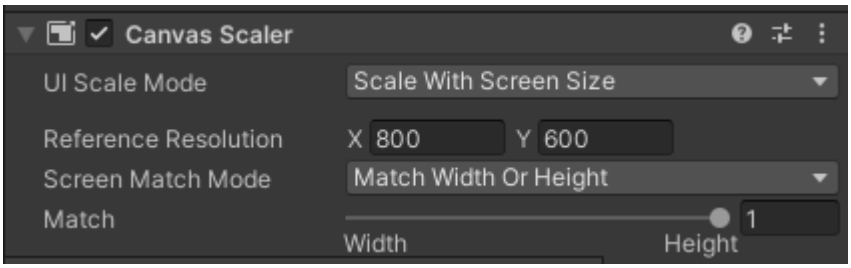

Рисунок 3.4 - Налаштування UI Scale

3.4 Реалізація логіки для задачі судоку

Реалізація поведінки UI дошки судоку починається з C# скрипта, де написано клас SudokuValue. Цей скрипт є компонентом UI елемента InputField. Дошка судоку складається з 81 InputField. SudokuValue відповідає за парсінг текстових типів у тип int32.

Скрипт SudokuManager це головний скрипт задачі судоку.

У ньому реалізовано:

- конструювання масива значень на дошці;
- патерн Singleton для уникнення створення більш ніж одної дошки за момент часу;
- система подій із застосуванням delegate;
- система CallBack;
- додавання та видалення значень з дошки.

Система подій та система CallBack необхідна для того, що б об’єкти дошки мали можливість підписуватись на подію OnSudokuDeskChanged. Таким чином, коли значення на дошці змінюється, викликається CallBack і всі підписані об’єкти виконують свою роботу.

Скрипт SudokuUI є скриптом UI інтерфейсу дошки судоку.

У ньому реалізовано:

- підписка на CallBack класу SudokuManager;
- відображення значень на дошці;
- перевірка введених значень за правилами гри в судоку;
- розширення режиму взаємодії під специфіку задачі судоку для персонажа;
- відображення повідомлення завершення задачі.

Для перевірки введених значень використовується hash. Hash використовується у виді зручного контейнера для зберігання значень у строках, стовпцях та блоках 3x3. Hash обрано завдяки його зручному маніпулюванню збереженими даними. Таким чином перевіряється кожна строка, яка складається с 9 значень. Кожне значення записується у контейнер. І за допомогою hash.Contains(value) перевіряє, чи було таке value вже серед цих 9 значень. Якщо ні, тоді викликаєм hash.Add() та записуємо нове значення. Якщо так, тоді завершуємо перевірку, та нічного не робимо. Після кожної дев’ятої ітерації, якщо збігів у строчці, колонці та блоку немає, викликається hash.Clear() та значення у контейнеру затираються.

На рисунку 3.5 можно побачити класову діаграму для дошки судоку.

Кімната-задача з арифметичними обчислюваннями пишеться аналогічно задачі судоку. Наслідування від класу InteractbleLogic, написання інтерфейсу, та написання логіки самої задачі.

Реалізовані класи:

- MathLauncherUI відповідає за відображення інтерфейсу;
- MathLogicManager відповідає за логіку арифметичних обчислювань;
- MathLogicInteract відповідає за специфіковану взаємодію.

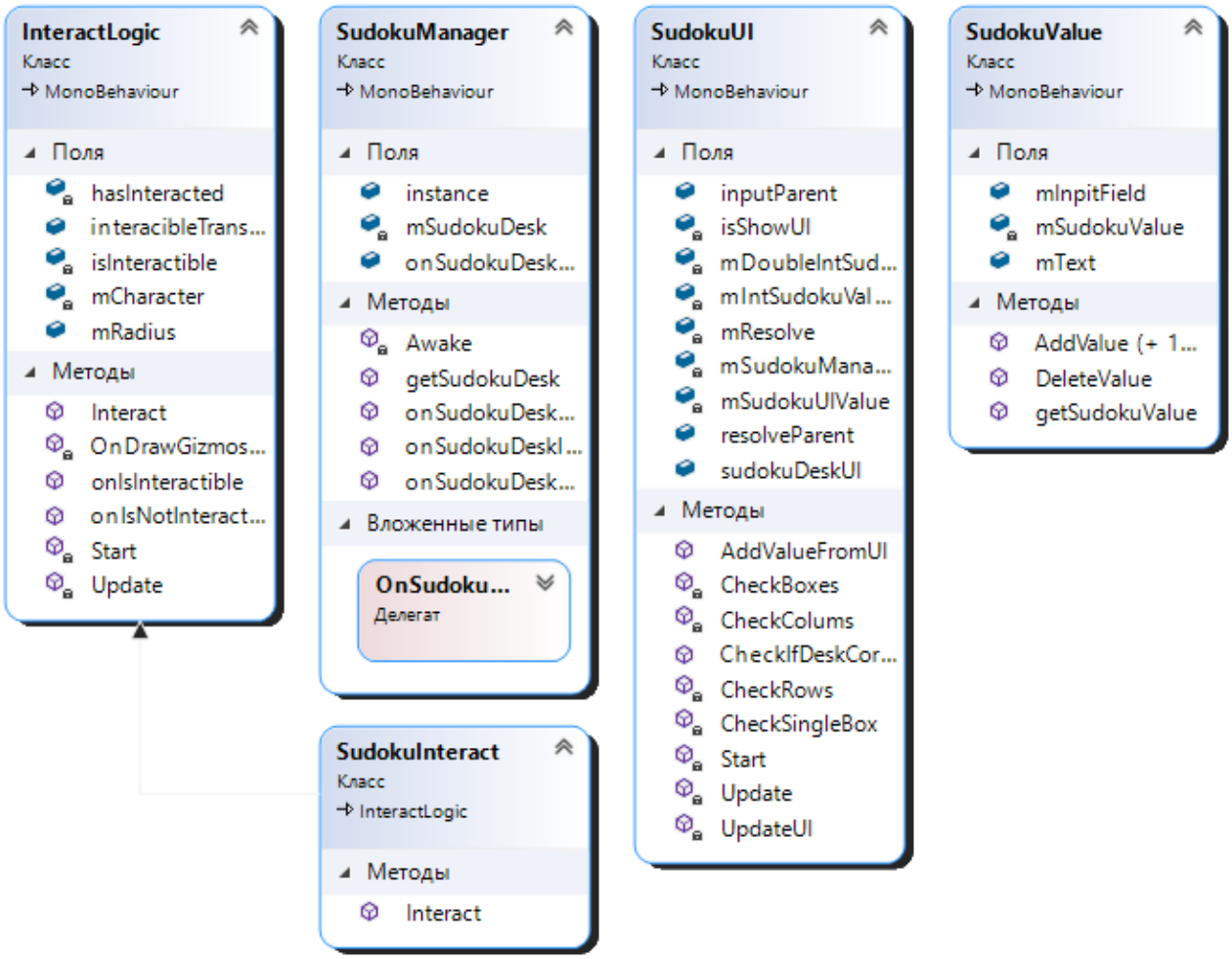

Рисунок 3.5 – діаграма класів дошки судоку

3.5 Інтегрування підсистеми кімнат-задач у процедурного-генеруєме оточення

Кімнати задачі інтегровані у процедурно-генеруєме оточення за допомогою Prefab. У Prefab зберігаються усі блоки, які використовуються для генерації оточення. Тому об’єкт задачі додається до одного з цих блоків.

Таким чином дошка судоку з’являється тільки біля блоку з часами. А задача з арифметичними обчислюваннями знаходиться біля столів.

3.6 Контроль версій проекту ТЛМ

У разі системи керування версій поєднаної роботи використовується Git. Сервіс GitHub використовується у разі віддаленого репозиторія для збереження версії проекту.

Контроль версій знадобився для впевненості, що під кінець реалізації проект буде у працюючому стані. Оскільки роботи перетинаються у деяких класах, за допомогою Git можливо зручно зливати зміни та уникати несправностей через вирішення конфліктів. Також навіть у випадку критичної несправності завжди можна використати git reset --soft до необхідного коміту або --hard якщо необхідно.

3.7 Приклад роботи підсистеми кімнат-задач

Після знаходження користувачем задачі (див. Рисунок 3.6) він може з нею взаємодіяти (див. Рисунок 3.7).

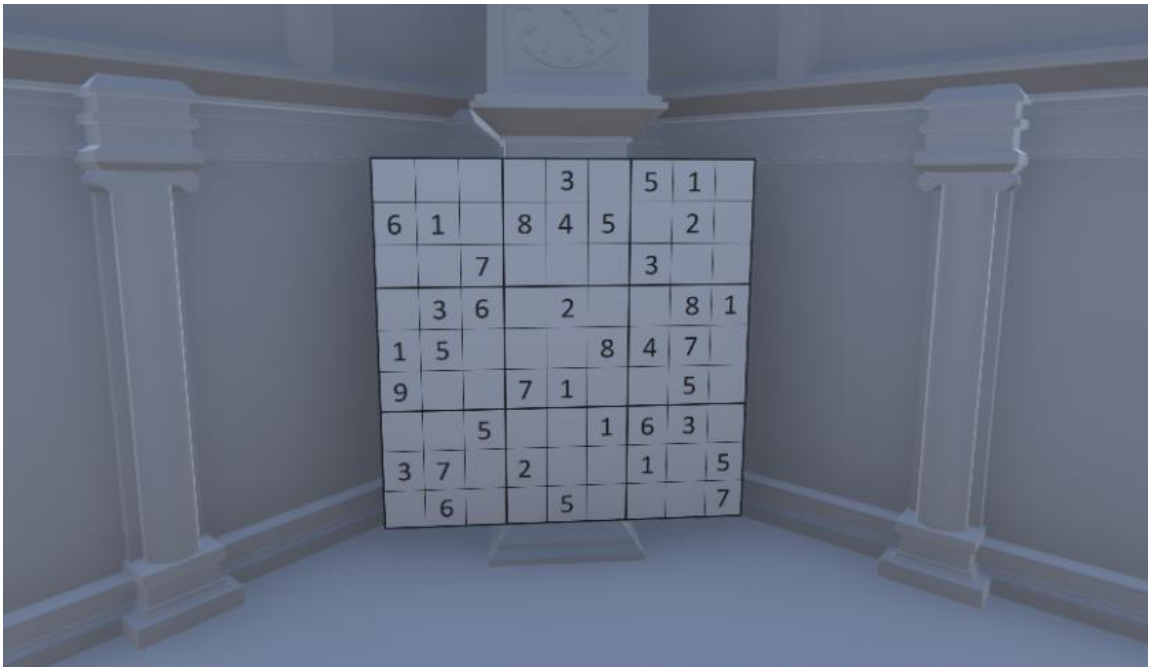

Рисунок 3.6 - Дошка судоку у генеруємому оточенні

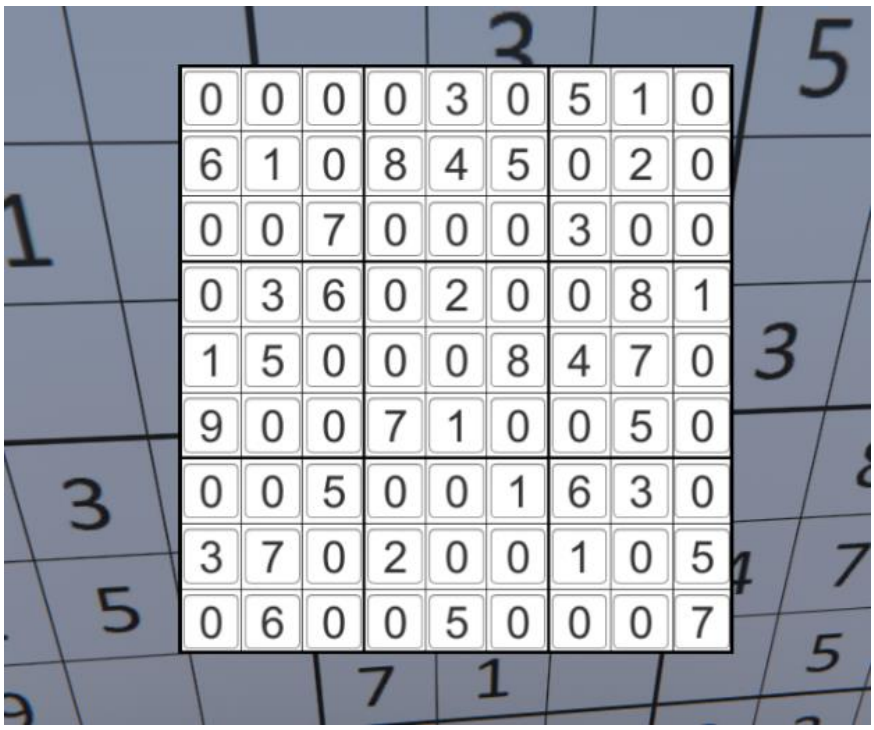

Рисунок 3.7 - Взаємодія з задачею

Якщо користувач не вирішить задачу, він нічого не побачить. Але якщо він вирішить задачу, то побачить повідомлення (див. Рисунок 3.8).

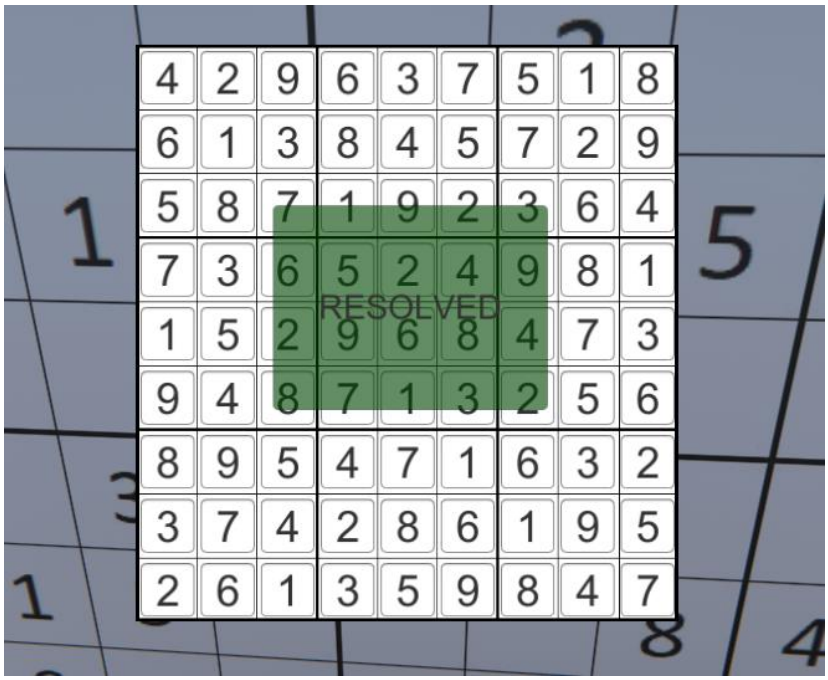

Рисунок 3.8 - Повідомлення про вирішення судоку

На рисунку 3.9 можна побачити інший приклад задачі – а саме арифметичну задачу. Користувач має ввести з клавіатури правильне значення арифметичного виразу. Підраховується кількість вірних та невірних відповідей, а також час відповіді.

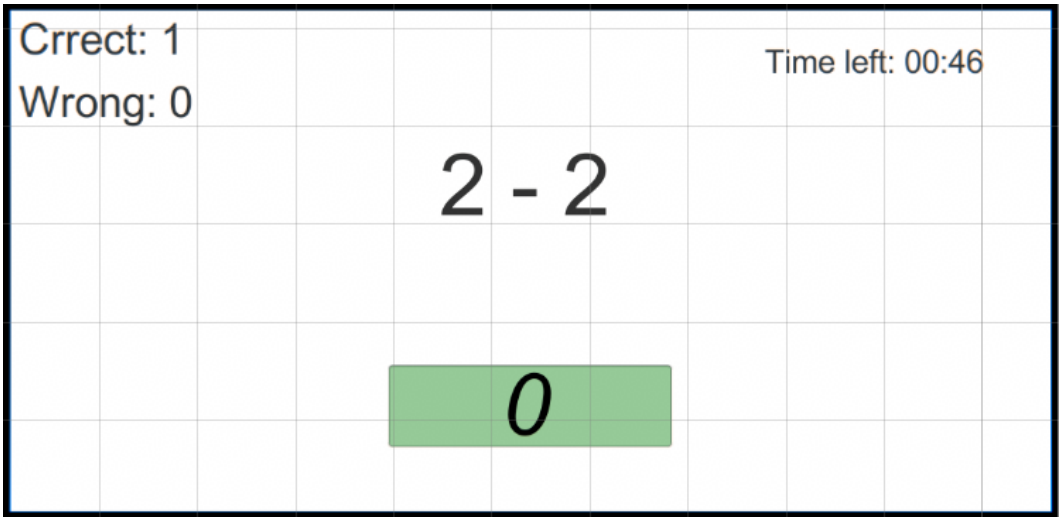

Рисунок 3.9 - Задача на швидкість арифметичних обчислень

ВИСНОВКИ

В ході виконання кваліфікаційної роботи проаналізовані принципи побудови тренажерів логічного мислення, розібрані різні варіанти логічних задач. Досліджена предметна область логічних методів тренування розумової діяльності, вплив вирішення задач на когнітивні здібності людини. Спроектована загальна архітектура програми у вигляді двох підсистем: підсистеми процедурної генерації оточення та підсистеми кімнат-задач та визначена загальна архітектура будь якої кімнати-задачі та архітектура двох конкретних кімнат-задач: судоку та тренажер з арифметичними обчислюваннями. Для реалізації обрано середовище розробки Unity та мова програмування C#. Реалізовані всі необхідні класи для реалізації алгоритмів цих двох кімнат-задач. Підсистема кімнат-задач інтегрована у процедурно-генеруєме оточення. Система, яка складається з підсистем процедурно-генеруємого оточення та кімнат-задач, протестована та працює справно.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Когнитивное развитие [Електронний ресурс] – Режим доступу: https://www.psychologies.ru
2. Логіка [Електронний ресурс] – Режим доступу: https://4brain.ru/logika/
3. Розвиток логічного мислення [Електронний ресурс] – Режим доступу: https://www.creativeschool.com.ua/blog/trenuyemo-logichne-myslennya-lajfhaky-vid-eksperta/
4. Оптимізація [Електронний ресурс] – Режим доступу: https://www.taina.com.ua/shho-take-optymizacija/
5. Морфологічний аналіз [Електронний ресурс] – Режим доступу: http://groupdynamics.kspu.edu/wiki/a/11
6. Теорія розв'язку винахідницьких задач [Електронний ресурс] – Режим доступу: https://www.kursak.com/teoriya-rozv-yazannya-vynakhidnytskykh-zavdan-trvz/
7. Офіційна документація Unity [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/Manual/index.html
8. Тренажеры для ума – MOZGOTREN [Електронний ресурс] – Режим доступу: https://mozgotren.ru/trenagors