МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Мобільний ігровий додаток для розвитку логічного мислення школярів»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ.м-12 Майфет Юрій Володимирович

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «___» грудня 2022 р.

Науковий керівник _____ к.т.н., доц., Баранова І.В.
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Суми-2022

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
В.о. зав. кафедри ІТ

_____ С. М. Ващенко
«___» _____ 2022 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Майфет Юрій Володимирович
(прізвище, ім'я, по батькові)

1 Тема проекту *Мобільний ігровий додаток для розвитку логічного мислення школярів*

затверджена наказом по університету від «_04_» _11__2022 р. № 01013-VI

2 Термін здачі студентом закінченого проекту «___» ___грудня___ 2022 р.

3 Вхідні дані до проекту технічне завдання на розробку мобільного ігрового додатку для розвитку логічного мислення школярів

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз програмних продуктів – аналогів, визначення технічних вимог додатку та засобів реалізації, створення комп'ютерної графіки, реалізація алгоритму розпізнавання зображень, розробка NoSql бази даних, створення інтерфейсу додатку

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність роботи, постановка задачі, аналіз програмних продуктів – аналогів, IDEF0-діаграма та її декомпозиція, діаграма варіантів використання, практична реалізація, демонстрація додатку

. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання 27.08.2022.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Аналіз предметної області	14.09.2022 – 15.09.2022	
2	Планування	22.09.2022 – 13.10.2022	
3	Створення прототипу	29.08.2022 – 05.09.2022	
4	Створення комп’ютерної графіки	14.10.2022 – 20.10.2022	
5	Створення скриптів	23.10.2022 – 28.10.2022	
6	Створення механізму розпізнавання зображень	29.10.2022 – 01.11.2022	
7	Створення інтерфейсу	02.11.2022 – 06.11.2022	
8	Тестування додатку	28.11.2022 – 03.12.2022	
9	Здача роботи	02.12.2022 – 07.12.2022	

Магістрант _____ Майфет Ю.В.

Керівник роботи _____ к.т.н., доц. Баранова І.В.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Мобільний ігровий додаток для розвитку логічного мислення школярів».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 40 найменувань та додатків.

Загальний обсяг роботи − 113 сторінок, у тому числі 50 сторінки основного тексту, 4 сторінки списку використаних джерел, 45 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці мобільного ігрового додатку для розвитку логічного мислення школярів під операційну систему Android,

В першому розділі роботи проведено аналіз предметної області з дослідженням схожих аналогів ігрових додатків та використання сучасних механізмів розпізнавання образів за допомогою штучного інтелекту.

Визначення мети проекту, задач та засобів реалізації проведено у другому розділі.

Третій розділ присвячено структурно-функціональному моделюванню.

У останньому розділі продемонстровано процес реалізації механізмів розпізнавання образів, створення головного меню та ігрового інтерфейса. Після реалізації проекту проведено тестування на працездатність додатка.

Результатом роботи є мобільний додаток, представлений у вигляді .apk файлу.

Практичне значення роботи полягає створенні мобільного ігрового додатку для телефонів на базі Android.

Ключові слова: мобільний ігровий додаток, логіка, штучний інтелект, Unity, Android.

ЗМІСТ

ВСТУП

В наш час можливості комп’ютерів ростуть, а з ними з’являються і нові технології, які значно полегшують життя людині. Однією з таких технологій є штучний інтелект, який може виконувати поставлені задачі швидше та ефективніше за звичайну людину .

Ідея застосування технологій штучного інтелекту для навчання не є новою. Та важливо не просто розробити технологію, а ще й потрібно подати її в такому вигляді, щоб нею зацікавився користувач. І саме формат мобільної ігри може допомогти розробнику привернути увагу до своєї розробки, бо на сьогодні смартфони є невід’ємною частиною людського життя. При достатній компактності вони є досить потужними кишеньковими комп’ютерами, відповідно можуть виконувати ті самі функції, що й звичайні комп’ютери. Тож неминуча і поява нових технологій, які б використовували цю компактність та вбудовані функції по типу сенсорного екрану.

Актуальність дипломної роботи зумовлена популярністю мобільних відеоігор серед школярів та необхідністю спрямувати зацікавлення відеоіграми на користь. Адже на відміну від класичної форми навчання, за підручниками, навчання в ігровій формі допоможе довше втримати увагу та досягти позитивних результатів.

Об’єктом дослідження є мобільна відеогра «Magic Logic» у жанрі логіки.

Предметом дослідження є застосування механізмів штучного інтелекту з розпізнаванням образів у мобільній грі та можливістю інтеграції таких механізмів у процес навчання.

Мета дипломної роботи – розробити мобільний ігровий додаток в жанрі логіка, який буде мати механізм розпізнавання намальованих образів за допомогою технологій штучного інтелекту.

Методи дослідження:

– теоретичні та емпіричні;

– об’єктно-орієнтоване програмування.

– створення структурно-функціональної моделі додатку.

Практичне значення полягає у тому, що розроблений мобільний додаток сприятиме розвитку логічного мислення у школярів шляхом використання цікавих ігрових механік та технологій штучного інтелекту у навчанні.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд проблеми та актуальність розробки ігрового додатку в жанрі логіка

На сьогодні, через швидкий ритм життя популярність мобільних ігор стрімко зростає, бо такі розваги не потребують багато часу та можуть зацікавити своїм різноманіттям [1]. Особливо мобільними іграми зацікавлені діти, бо для них такі розваги є простими в розумінні і не потребують потужних комп'ютерів чи ігрових консолей [2,3].

Актуальність даної теми зумовлена схильністю дітей приділяти більшість свого часу розвагам, а не навчанню. Розробка ігрового додатку у форматі гри допоможе утримати увагу дитини, а невеликі завдання на логіку – провести ігровий час з користю [4].

Наразі ринок мобільних ігор має досить великий асортимент навчальних вправ, але значна більшість з них має низьку якість реалізації [5,6]. Якщо взяти декілька середньостатистичних ігор з логічними вправами [7], то можна побачити, що здебільшого вони мають одну ігрову механіку, а саме обирання правильної відповіді з готових варіантів. При цьому відрізняються одна від одної вони лише візуальним стилем.

Такий підхід скоріше сприяє візуальному запам'ятовуванню правильної відповіді, а не спонукає думати та створювати свої алгоритми для розв'язання задач, що насамперед є головним заданням логічних ігор.

Слід звернути увагу на технології штучного інтелекту [8,9], який буде основою власного проекту. Головною метою інтеграції штучного інтелекту у додаток буде розпізнавання та обробка намальованих зображень користувачем.

На сьогодні розпізнавання зображень широко використовується в таких галузях як: медицина, військова справа, наукові дослідження тощо [10,11]. Здебільшого дані технології функціонують на базі нейронних мереж [12,13], бо

такі мережі значно швидше та точніше можуть виконувати складний пошук закономірностей. Ефективність нейронних мереж зумовлена великою базою прикладних даних, з яких вони складають свої алгоритми пошуку. Прикладом може слугувати нейронна мережа Deeplab v3+ [14,15], яка широко використовується в медицині і може з точністю до 92% виявляти серйозні хвороби на знімках [14].

Не зважаючи на всі переваги нейронних мереж, написання найпростішої мережі з не дуже великою точністю результатів потребує багато часу та цілої команди досвідчених спеціалістів [16]. Тому згідно тематики роботи, а саме розробка мобільного ігрового додатку, було вирішено звернути увагу на можливі ідеї з реалізації розпізнавання простих фігур за допомогою алгоритмів, які будуть зчитувати дії гравця [17, p. 256]. Такий підхід значно зменшить час на розробку та повністю задовільнить всі цілі проекту.

На сьогодні існує декілька вдалих рішень для простого розпізнавання простих фігур:

- зчитування рухів гравця за координатами;
- зчитування задіяних в процесі малювання активних точок на екрані;
- створення фігур з використанням математичних координат і порівняння цих координат з намальованих гравцем;
- зчитування за допомогою контекстного методу .

Перший метод працює таким чином, що рухи гравця зчитуються прямо в процесі малювання і програма на ходу може вгадувати введений символ. Але такий метод потребує більших апаратних потужностей і більше підходить для реалізації разом з нейронними мережами [18–20].

Другий метод полягає в тому, що створюються початкові набори точок з яких складається фігура і при зчитування програма вираховує скільки точок перетнув гравець при малювання фігури, а далі порівнює результат перетнутих точок з шаблонами і надає результат [19–21].

Третій метод найпростіший, бо для розпізнавання зображень він лише порівнює математичні координати заздалегідь підготованого шаблону фігури та

координати, які створив гравець в процесі малювання. Такий підхід є досить неточним бо розпізнавання можливе лише тоді коли гравець намалював фігуру яка майже ідеально повторює підготований варіант [19,20].

Останній метод сам розбиває фігуру на точки та вираховує відносний розподіл точок на площині відносно кожної точки на фігурі, тому для такого обчислення потрібні високі апаратні потужності [22,23].

На даний момент для розробки власного проекту буде використано поєднання другого та третього методів.

1.2 Огляд існуючих аналогів

Перед початком розробки продукту необхідно провести детальний огляд та аналіз максимально схожих за функціоналом вже готових рішень. Зокрема необхідно розглянути існуючі технології з розпізнаванням зображень.

Для пошуку подібних існуючих ігор використано електронний магазин Play Market.

В додатку Japanese Writing Wizard гравцю запропоновано спробувати вивчити та намалювати базові літери одного японського алфавіту. Також цей додаток має механізм малювання літер за допомогою сенсорного екрану (рис. 1.1).

Але провівши деякий час і вивчаючи механізм малювання, можна побачити, що в додатку не реалізовані функції розпізнавання зображення. Також немає ніякого інтерактиву та інших механік, окрім малювання, тому для кожного завдання необхідно робити самоперевірку. Такий підхід не є дуже вдалим, бо без перевірки в самій грі та виводу результатів проходження рівнів неможливо довго утримувати увагу гравця [24–26]. Ще слід зазначити, що графічний інтерфейс додатку занадто простий, щоб привернути до себе увагу.

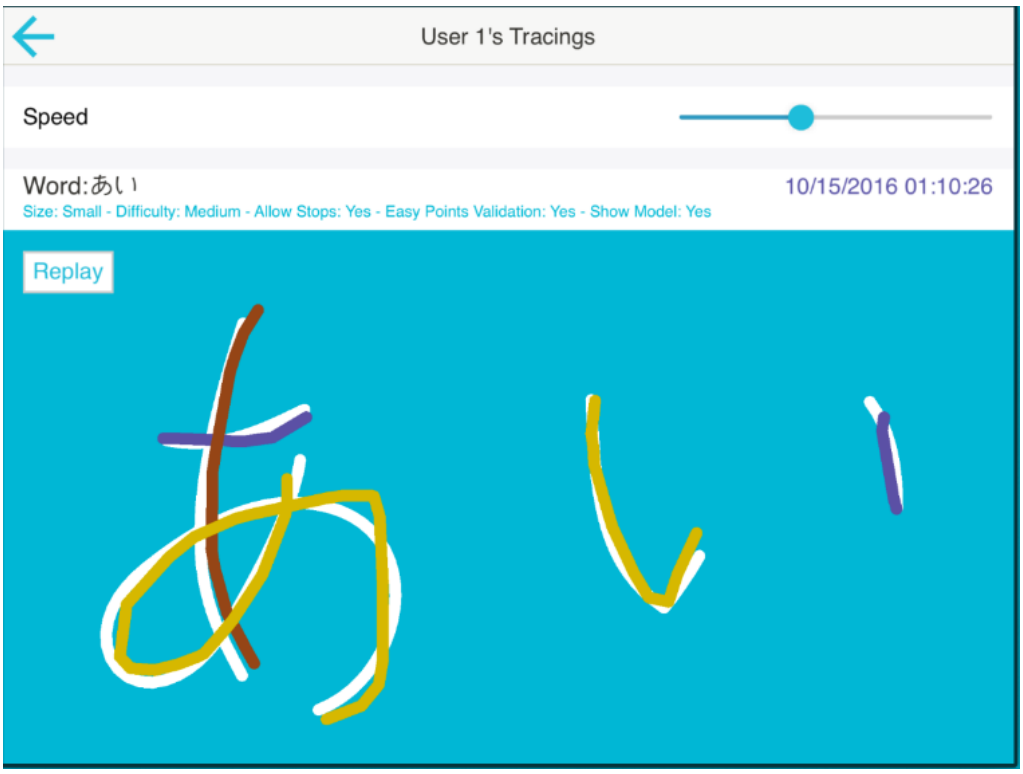


Рисунок 1.1 – Механізм малювання літер в додатку Japanese Writing Wizard

В грі Math games ми бачимо класичну для майже всіх додатків в жанрі логіки реалізацію механіки з питанням та декількома відповідями (рис. 1.2).

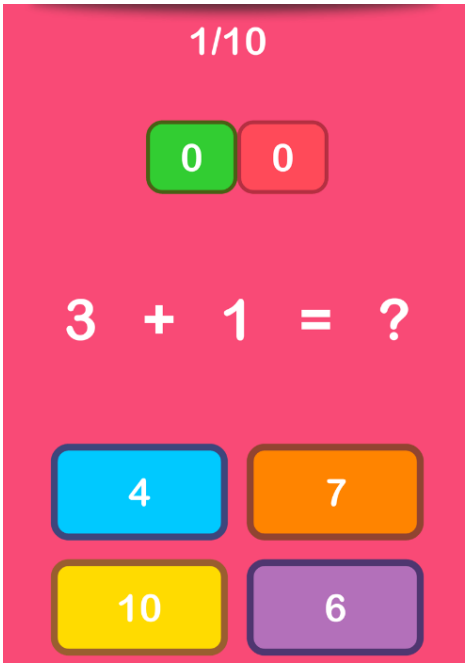


Рисунок 1.2 – Ігровий процес в додатку Math games

Ігровий додаток Math games має приємний візуальний стиль та нескладні завдання на базову арифметику. Очевидно, що ця гра розрахована на дитячу аудиторію, і завдяки різноманіттю завдань вона добре утримує увагу гравця. Але головний недолік підходу з питанням та готовими відповідями – це сприяння візуальному запам’ятовуванню відповідей, а не розвитку логічного мислення.

Рішення Shapecatcher не є грою чи мобільним додатком, але представляє собою найбільш наближену до наших цілей технологію розпізнавання зображень. Даний ресурс працює на базі високошвидкісних графічних прискорювачів з наявністю CUDA ядер [27,28] що дозволяє в декілька порядків швидше проганяти намальований користувачем рисунок через алгоритми штучного інтелекту. Приклад роботи ресурсу наведено на рисунку 1.3.

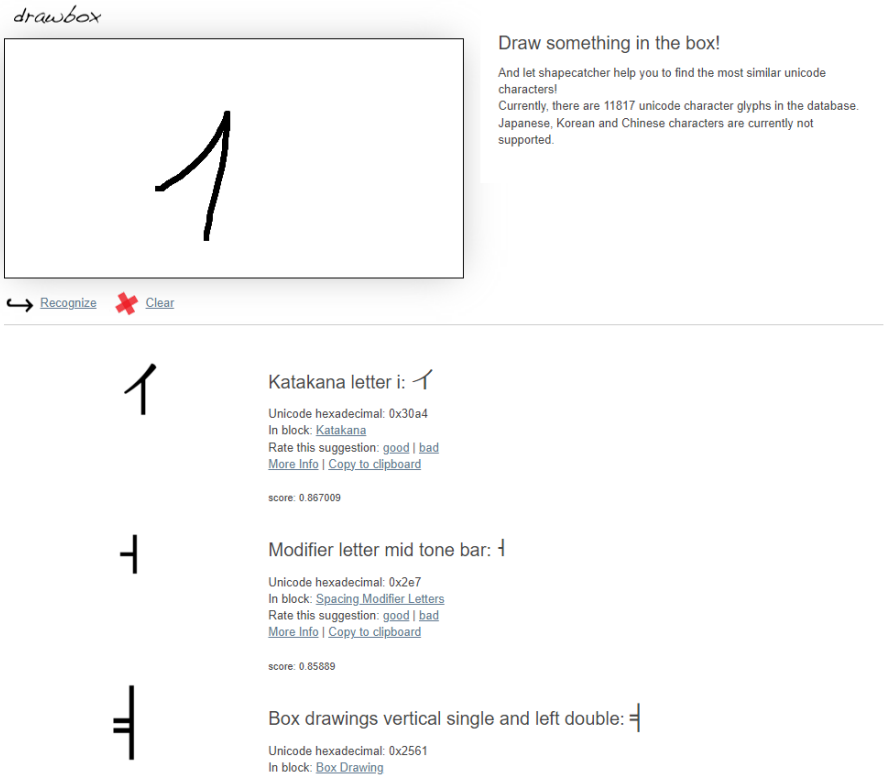


Рисунок 1.3 – Розпізнавання образів за допомогою ресурсу Shapecatcher

Також автор розробки стверджує, що використовував алгоритм, оснований на ідеї контекстного розпізнання зображення [22,23]. Основна ідея контекстного алгоритму – це враховуючи набір точок із зображення, контекст фігури фіксує відносний розподіл точок на площині відносно кожної точки на фігурі. Зокрема,

обчислення відбувається за допомогою логарифмічних полярних координат [29], як показано на рисунку 1.4.

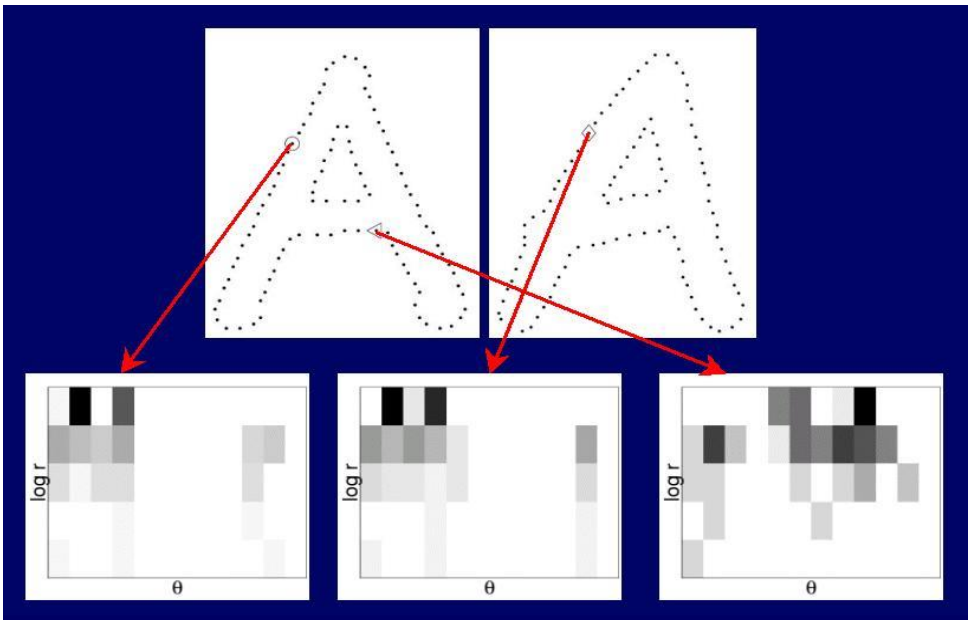


Рисунок 1.4 – Приклад обчислення за допомогою контекстного алгоритму

Дивлячись на результат даного алгоритму, можна побачити, що він не є зовсім точним і намальовану цифру «один» (рис 1.3) він розпізнає як можливі японські чи корейські літери.

Також хоча сучасні смартфони і є достатньо потужними, але вони не можуть змагатись в потужності з сучасними графічними процесорами, які встановлені в стаціонарних комп’ютерах [30]. Тому можна взяти ідеї з ресурсів та досліджень, що використовував автор даного проєкту [22,23], та розробити власний алгоритм, який буде не так навантажувати систему та зможе працювати навіть на старих смартфонах. Також необхідно покращити точність розпізнавання рисунків, виходячи з кінцевих потреб.

1.3 Результат дослідження аналогів

В результаті ретельного дослідження вже існуючих аналогів були виявлені суттєві недоліки, які повинні бути усунуті у власній розробці. Для систематизації була складена таблиця з порівняльними критеріями (табл. 1.1), де оцінка 1 – погано, 2 – потребує доопрацювань, 3 – добре.

Таблиця 1.1 – Порівняння досліджених аналогів

Критерії оцінювання	Japanese Writing Wizard	Math games	Shapecatcher
Графічне оформлення	1	3	-
Складність завдань	3	3	-
Цікавіть для користувача	1	2	2
Технологічність	2	1	3
Реалізація запропонованих механік	1	1	2
Системні вимоги для смартфону	3	3	1

При порівнянні аналогів між собою можна бачити, на що потрібно звернути увагу в першу чергу, а саме, особливо необхідно зосередити увагу на повну реалізацію запланованих механік, що частково може збільшити цікавість до продукту [31,32].

Також після аналізу Shapecatcher було виявлено, що велика складність

алгоритму може погано працювати навіть на найновіших смартфонах. Тому необхідно приділити особливу увагу оптимізації та тестуванню на різноманітних моделях гаджетів, щоб покращити досвід користування додатком якомога більшій кількості користувачів.

2 ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі дослідження

Мета даного проекту – це створити мобільний ігровий додаток в жанрі «логіка», який буде використовувати технології штучного інтелекту для покращення навчання.

Готовий мобільний додаток має набір завдань на арифметику, знання мови та логіку, що надає можливість інтегрувати навчальний процес в ігрову форму, а для ефективної реалізації вправ та для утримання уваги буде реалізований механізм розпізнавання намальованих користувачем зображень.

Всього ігровий додаток буде мати чотири типи завдань, а саме два - завдання на математичні вправи та два - завдання на знання мови. Всі типи завдань будуть мати таймер, який реагує на неправильні відповіді користувача і в подальшому час проходження рівня буде впливати на загальну оцінку та кількість балів.

Також перед початком розробки ігрового додатку необхідно:

- ретельно проаналізувати існуючі аналоги та предметну область;

- розробити чіткий план проекту, поетапне виконання якого дозволить мінімізувати кількість помилок та досягти успіху.

Ігровий додаток повинен бути функціональним та здатним реалізувати поставлені задачі на великій кількості систем:

- для утримання уваги продукт повинен мати приємний візуальний стиль;

- інтерфейс повинен бути інтуїтивно зрозумілим;

- для різноманіття необхідно мати декілька завдань та рівнів;

- збереження кінцевого результата та результатів проходження окремих рівнів;

- щоб одразу дати користувачу грати в цікаві йому рівні, всі рівні будуть доступні одразу;

- можливість зміни налаштувань параметрів рівнів;
- можливість зміни налаштувань параметрів додатку;
- необхідна гарна оптимізація для можливості роботи на більшості смартфонах.

2.2 Технології та інструменти реалізації

Для реалізації поставлених при розробці мобільного ігрового додатку задач, необхідно обрати зручні та функціональні інструменти реалізації.

Наразі найкращими і найпоширенішими мовами програмування для розробки ігрових додатків є C, C++ та C#.

C – це мова програмування, яка була розроблена у 1972 році. Мова не є об'єктно-орієнтованою, але має перевагу в тому що надає можливість напряму працювати з пам'яттю і може бути скомпільована на більшості існуючих систем [33].

C++ – це об'єктно-орієнтована мова програмування, яка є наслідником мови C, що зумовлено схожим синтаксисом та можливостями, але порівняно з першою має суттєві переваги у вигляді можливості створення класів та слідуванням всім парадигмам об'єктно-орієнтованого підходу [34].

C# - це об'єктно-орієнтована мова програмування, яка належить компанії Microsoft і розроблена для .NET платформи. Вона як і C++ є похідною від мови C, але є більш безпечною для користування завдяки своїй системі типізації та неможливості множинного спадкування [35].

Тому для подальшого розгляду та аналізу було обрано три найпоширеніші ігрові рушії, які підтримують вище розглянуті мови програмування, а також дадуть змогу реалізувати проект на мобільні гаджети.

- Unreal Engine;
- MonoGame;
- Unity.

Unreal Engine – це ігровий рушій, який написаний на мові C++ та є власністю і розробкою компанії Epic Games. Перше практичне використання рушія було продемонстровано 1998 року в комп'ютерній грі Unreal. З того часу компанія розробник поступово покращувала свій продукт і в даний час Unreal Engine є одним із лідерів на ринку, який використовують від розробки казуальних ігор для операційної системи Android до багатьох бюджетних продуктів для сучасних ігрових консолей [36,37].

Також Unreal Engine має підтримку всіх сучасних технологій обробки та рендеру графіки RTX, OpenGL, DirectX та інші. Але найбільша особливість цього ігрового рушія – це можливість зручного візуального програмування за допомогою технології Blueprint (рис 2.1).

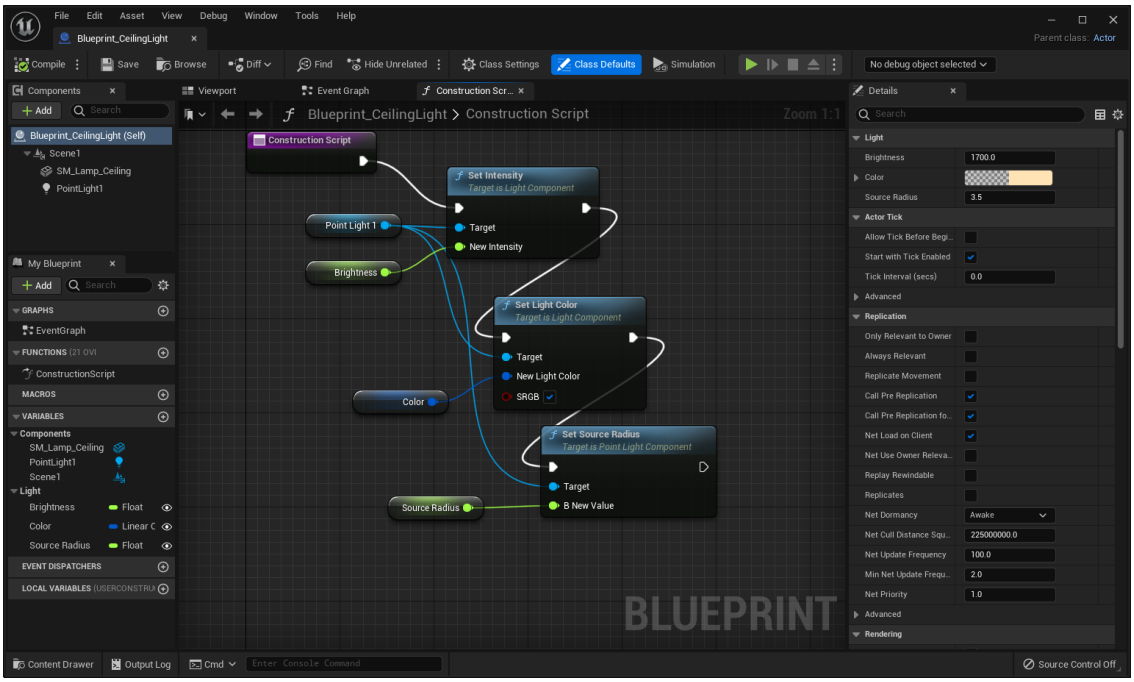


Рисунок 2.1 – Приклад використання технології Blueprint

Такі переваги надають можливість досить гнучко будувати ігрові сценарії без суттєвих навичок програмування. Завдяки відкритому вихідному коду та вдало обраній мові програмування C++ можна без зайвих труднощів створювати свої функції та нові системи. Проте всі вище перелічені переваги можна реалізувати, лише працюючи в досвідченій команді, бо така велика

функціональність та варіативність потребує спеціальних навичок, які складно освоїти одній людині.

MonoGame – це ігровий рушій, який представляє собою Open Source реалізацію фреймворка XNA 4 від компанії Microsoft. Офіційно сам Microsoft XNA 4 починаючи з 2012 не підтримує, але на той час рушій був досить перспективним та мав підтримку мови програмування C# та VB.NET, що дозволяло створювати як двовимірні, так і тривимірні проекти різної складності, але значним мінусом є те, що рушій підтримує розробку тільки для ОС Windows.

Наразі MonoGame є продовжувачем ідей XNA 4 та активно підтримується самими користувачами і має підтримку майже всіх актуальних платформ. Але, на відміну від інших рушіїв по типу Unity та Unreal Engine, сам MonoGame не є типовим набором готовим рішень, а скоріше є початковою платформою для створення власних інструментів та технологій. Такий підхід є досить зручним, але потребує значного часу на реалізацію базових інструментів необхідних для початку роботи.

Unity – це ігровий рушій, який був представлений 2005 року та розроблений компанією Unity Technologies. Наразі цей ігровий рушій має підтримку всіх існуючих на даний момент платформ і є найпопулярнішим вибором для розробки мобільних та інді-ігор. Також, завдяки великій кількості інструментів, рушій надає можливість створювати тривимірні ігри з сучасним графічним оформленням, ігри з доповненою реальністю, проекти для шоломів віртуальної реальності та, навіть, створювати сцени, які можуть використовуватись в кінематографі. Сам рушій має простий інтерфейс (рис 2.2) і пропонує реалізувати скрипти на мові C# та великий вибір з готових бібліотек, написаних розробниками, які постійно оновлюються та доповнюються [38].

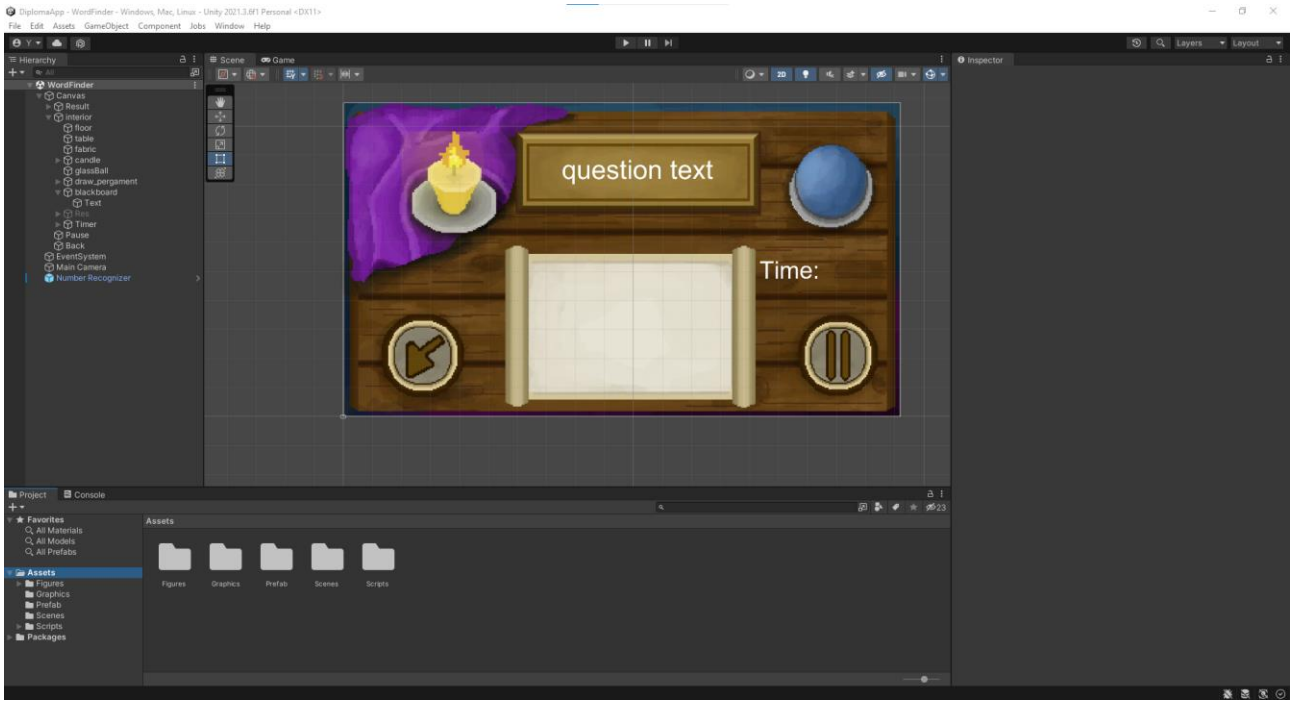


Рисунок 2.2 – Інтерфейс ігрового рушія Unity

В результаті дослідження обраних рушіїв були виявлені переваги та недоліки кожного з них. Для систематизації складена таблиця з порівняльними критеріями (табл. 2.1), де оцінка 1 – погано, 2 – є незначні недоліки, 3 – добре.

Таблиця 2.1 – Порівняння досліджених аналогів

Критерії оцінювання	Unreal Engine	MonoGame	Unity
Складність освоєння для однієї людини	1	1	3
Зручність інтерфейсу	3	1	3
Готові інструменти для мобільних платформ	2	1	3
Підтримка платформ	3	3	3
Доступність документації	2	2	3
Системні вимоги для смартфону	2	3	2

Виходячи з порівняння рушіїв між собою, було вирішено обрати саме Unity, так як він має всі необхідні інструменти для реалізації поставлених задач та не

потребує багато часу на освоєння.

Далі необхідно обрати зручне середовище розробки. Наразі найбільш популярними та зручними є такі середовища:

- Visual Studio Code,.
- Visual Studio

Visual Studio Code та Visual Studio є рішеннями від компанії Microsoft, які підтримують майже всі відомі мови програмування і головною відмінністю є те, що Visual Studio Code значно легший, тому в ньому менше функцій і можливостей. В свою чергу Visual Studio є ультимативним рішенням з безліччю функцій та офіційних плагінів. Самі розробники рушія Unity рекомендують використовувати Visual Studio, тому було вирішено обрати саме його.

Так як цей проект буде працювати на смартфонах, то для зменшення системних вимог вирішено створювати комп’ютерну графіку у стилі піксель-арт, що також допоможе в економії часу розробки. Основним інструментом для створення спрайтів буде Adobe Photoshop, бо він має зручний інтерфейс (рис. 2.3) та гнучкий інструментарій у вигляді стандартних та додаткових плагінів. Головним критерієм у виборі даного інструменту є те, що він підтримує введення для будь-яких графічних планшетів.

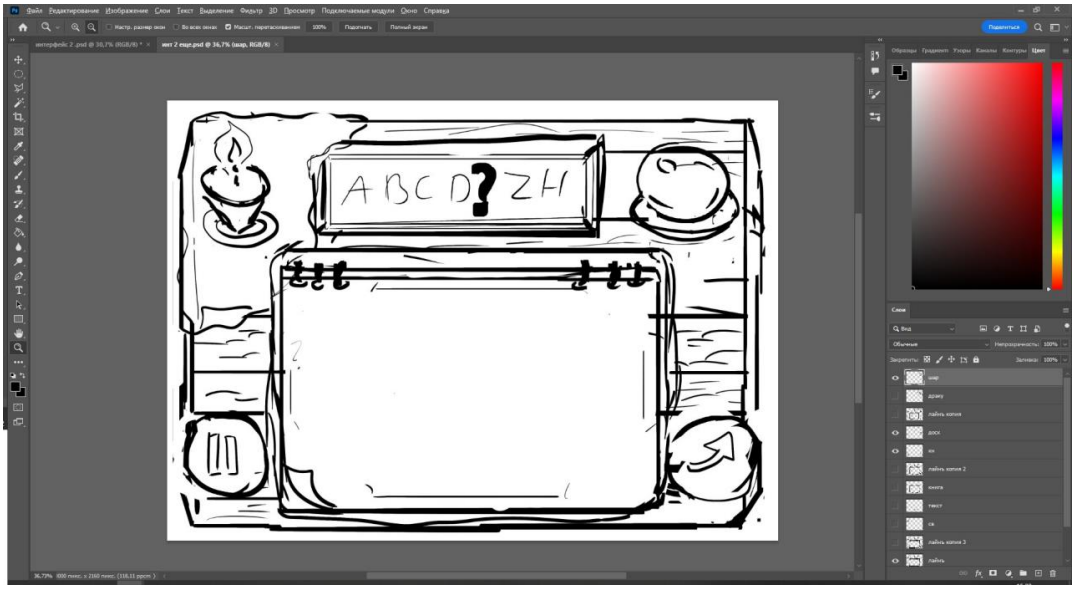


Рисунок 2.3 – Інтерфейс графічного редактора Adobe Photoshop

Для подальшого редагування та створення анімацій буде використовуватись Aseprite (рис 2.4). Хоча він має набагато меншу функціональність, але її вистачить для виправлення помилок на вже готових малюнках.

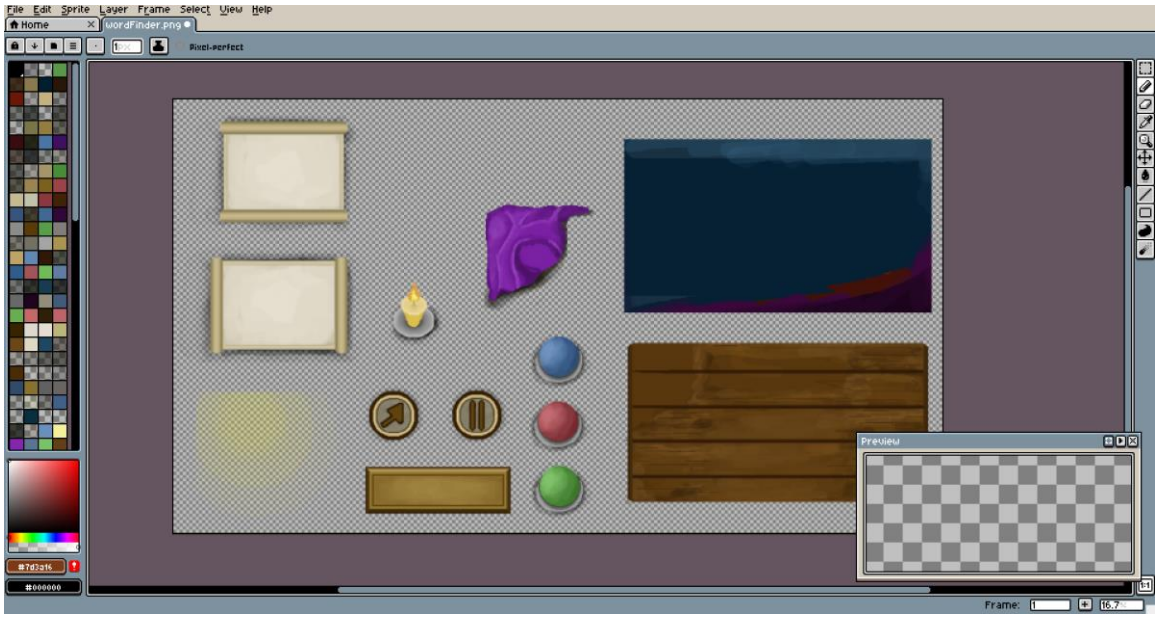


Рисунок 2.4 – Інтерфейс графічного редактора Aseprite

Узагальнення обраних інструментів реалізації наведено в таблиці 2.2.

Таблиця 2.2 – Засоби для реалізації проєкту

Поставлені задачі	**Обраний інструмент реалізації**
Мова програмування	C#
Ігровий рушій	Unity
Текстовий редактор	Visual Studio
Графічний редактор	Adobe Photoshop, Aseprite

3 ПРОЄКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

3.1 Структурно-функціональне моделювання процесу

Для створення структурно-функціонального моделювання буде використовуватися методологія Structured Analysis and Design Technique або SADT [39]. Такий підхід необхідно використовувати, якщо нам потрібно показати, описати та проаналізувати механізми продукту.

Для стислого графічного відображення роботи системи допоможе створення IDEF діаграми. Вона значно спрощує розуміння функціональних процесів та має власну документацію з чіткими вимогами до реалізації. На рисунку 3.1 зображена контекстна діаграма додатку.

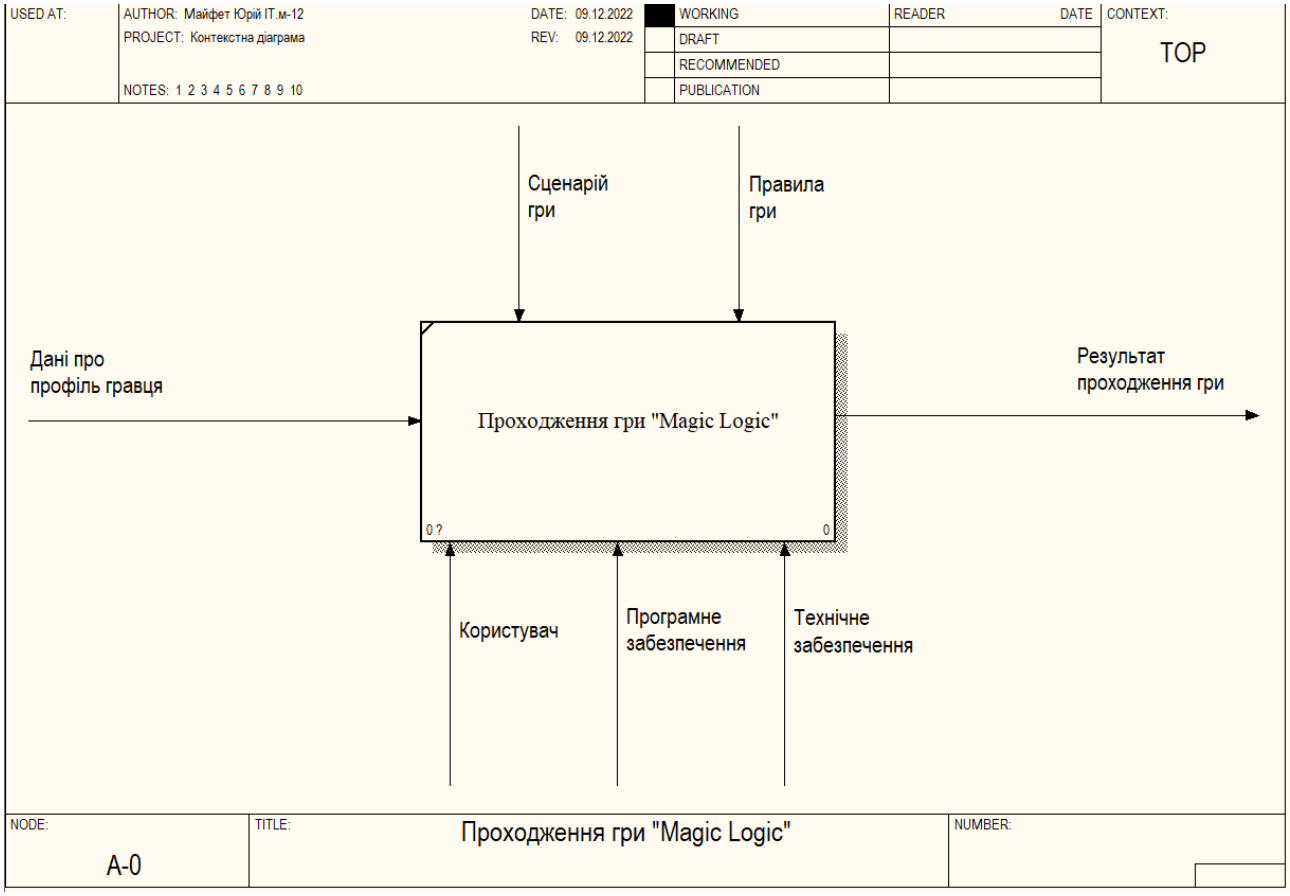


Рисунок 3.1 – Контекстна діаграма додатку «Magic Logic»

Стрілки, що мають назву «сценарій гри» та «правила гри», ілюструють можливості та рамки користування додатком. Стрілки «користувач», «програмне забезпечення» і «технічне забезпечення» - це інструменти, завдяки яким користувач взаємодіє з грою.

Далі більш детально про ці компоненти:

- сценарій проходження – це малювання фігур, відповідей, завдяки яким будуть проходитися рівні;
- правила гри – це компонент який визначає рамки взаємодії гравця з додатком;
- користувач – це гравець;
- програмне забезпечення – це готовий продукт, який гравець повинен запустити для проходження рівнів, і програми та скрипти, які дозволяють запускати та користуватися ігровим додатком. Сюди можуть входити як операційна система гаджету, так і скрипти самого додатку;
- технічне забезпечення – це гаджет з необхідними системними вимогами, які дозволять користуватись додатком;
- результат проходження гри – це кінцева фіксація прогресу проходження рівнів з подальшою обробкою інформації про дане проходження.

Також, для повноти інформації можна розширити діаграму та зробити декомпозицію рівня (рис. 3.2), який включає такі блоки:

- завантаження головного меню,
- вибір рівня гри,
- проходження рівня,
- збереження результату проходження.

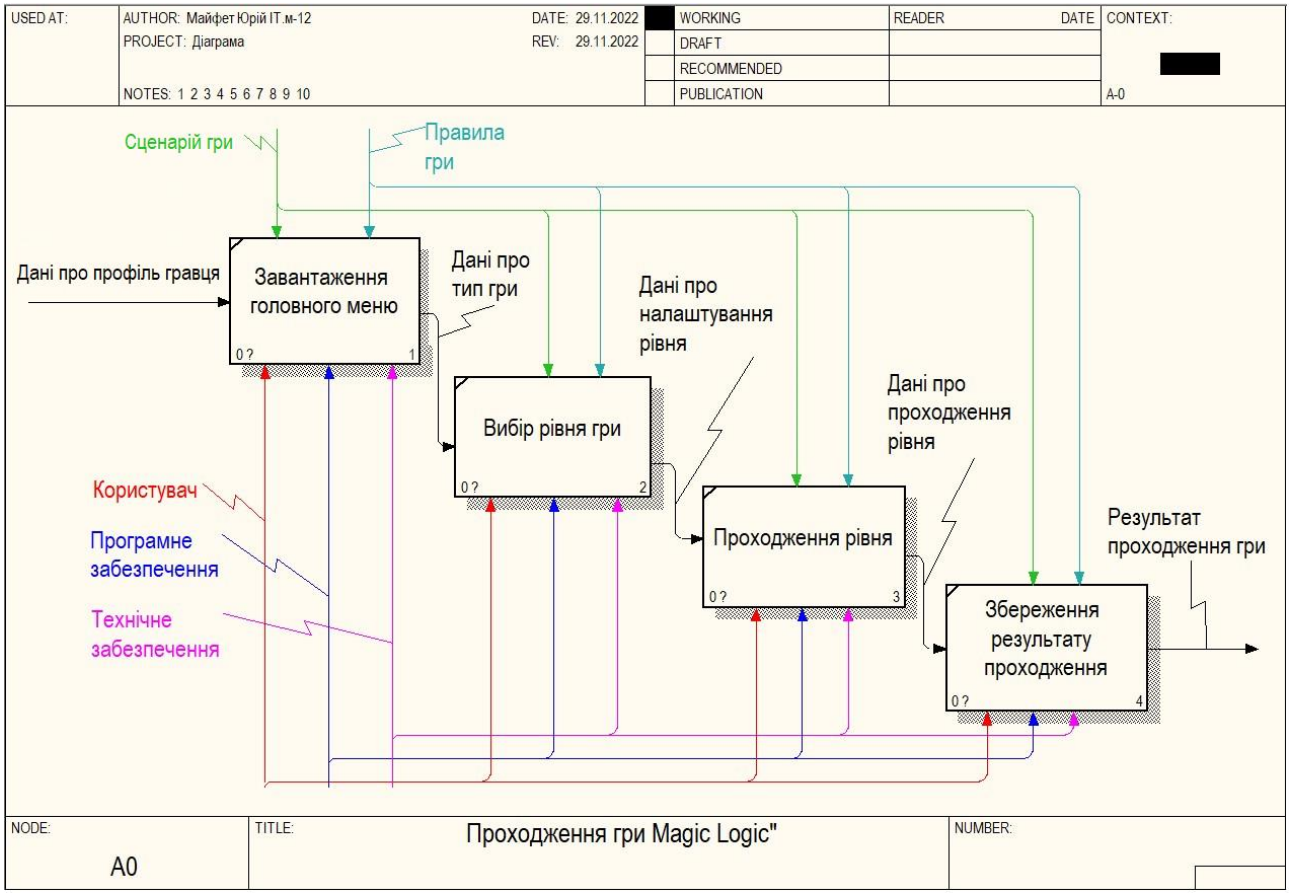


Рисунок 3.2 – Перший рівень декомпозиції

Після завантаження додатку, гравець потрапляє в головне меню, де йому необхідно обрати один з двох типів гри, а саме арифметика або знання мови. Далі обирається рівень, після його проходження дані про дії гравця зберігаються та обробляються для оцінки.

В свою чергу блоки «Вибір рівня гри» та «Проходження рівня» мають складові компоненти, а тому для них теж можна створити декомпозицію.

Декомпозиція блоку «Вибір рівня гри» зображена на рисунку 3.3.

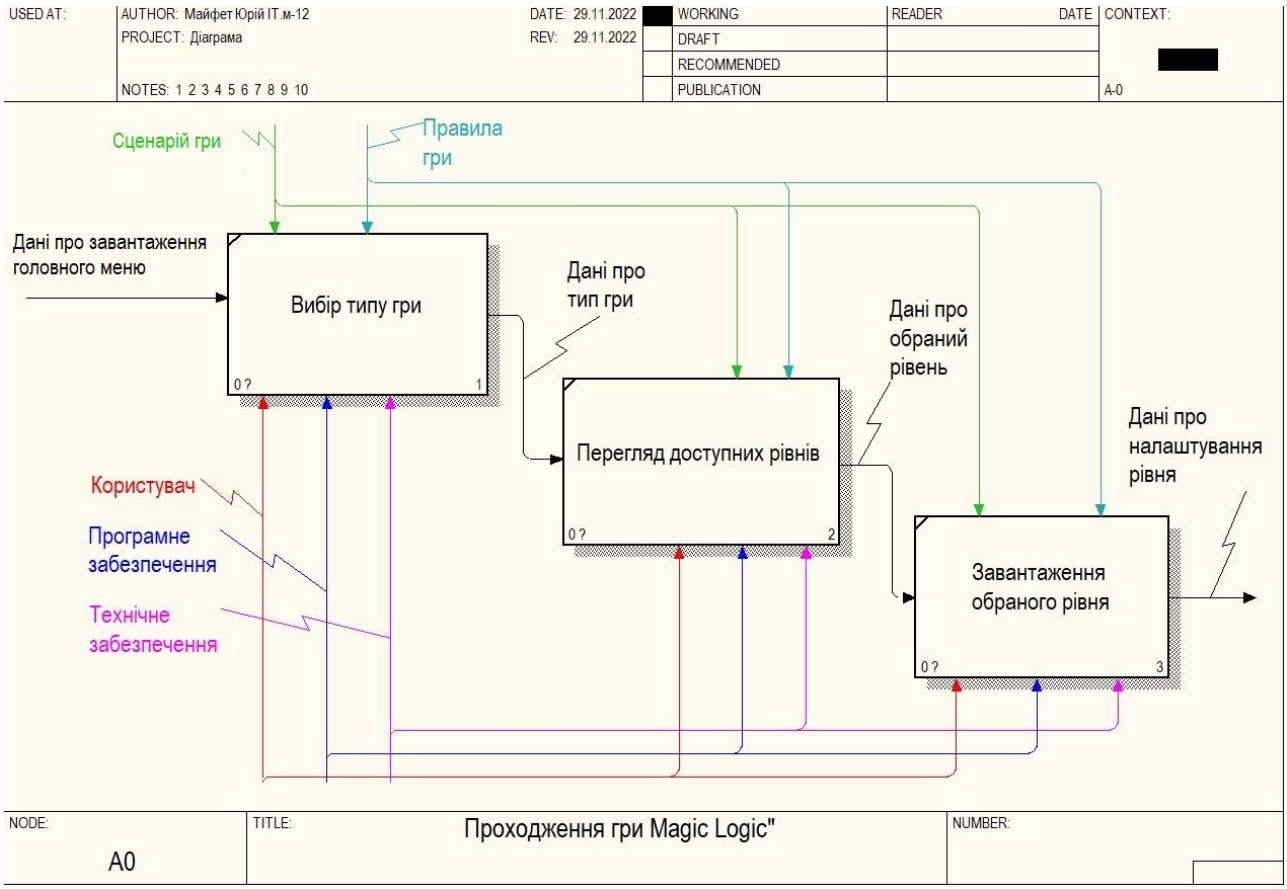


Рисунок 3.3 – Декомпозиція блоку «Вибір рівня гри»

В даній декомпозиції проілюстровані процеси, які виконує гравець при виборі рівня гри. «Дані про вибір початку гри» несуть інформацію про те, що гравець натиснув на кнопку головного меню «Почати гру» і далі йде вікно «Вибір типу гри». Після вибору типу гри гравець може обрати будь-який доступний рівень та налаштувати його згідно своїх потреб. Після всіх процедур з вибором рівня, «Дані про налаштування рівня» йдуть в інший блок «Проходження рівня», декомпозиція якого зображена на рисунку 3.4.

Рисунок 3.4 ілюструє процес проходження рівня гравцем. Спочатку завантажується рівень з налаштуваннями, які перейшли з попереднього блоку (рис. 3.3) і перетворюються в «Дані про вибір початку гри». Далі гравець переглядає завдання і розв’язує їх, що ілюструють блоки «Перегляд завдань» та «Розв’язання завдань». Після успішного виконання всіх задач, гра отримує дані про введені відповіді і обробляє їх для подальшого збереження.

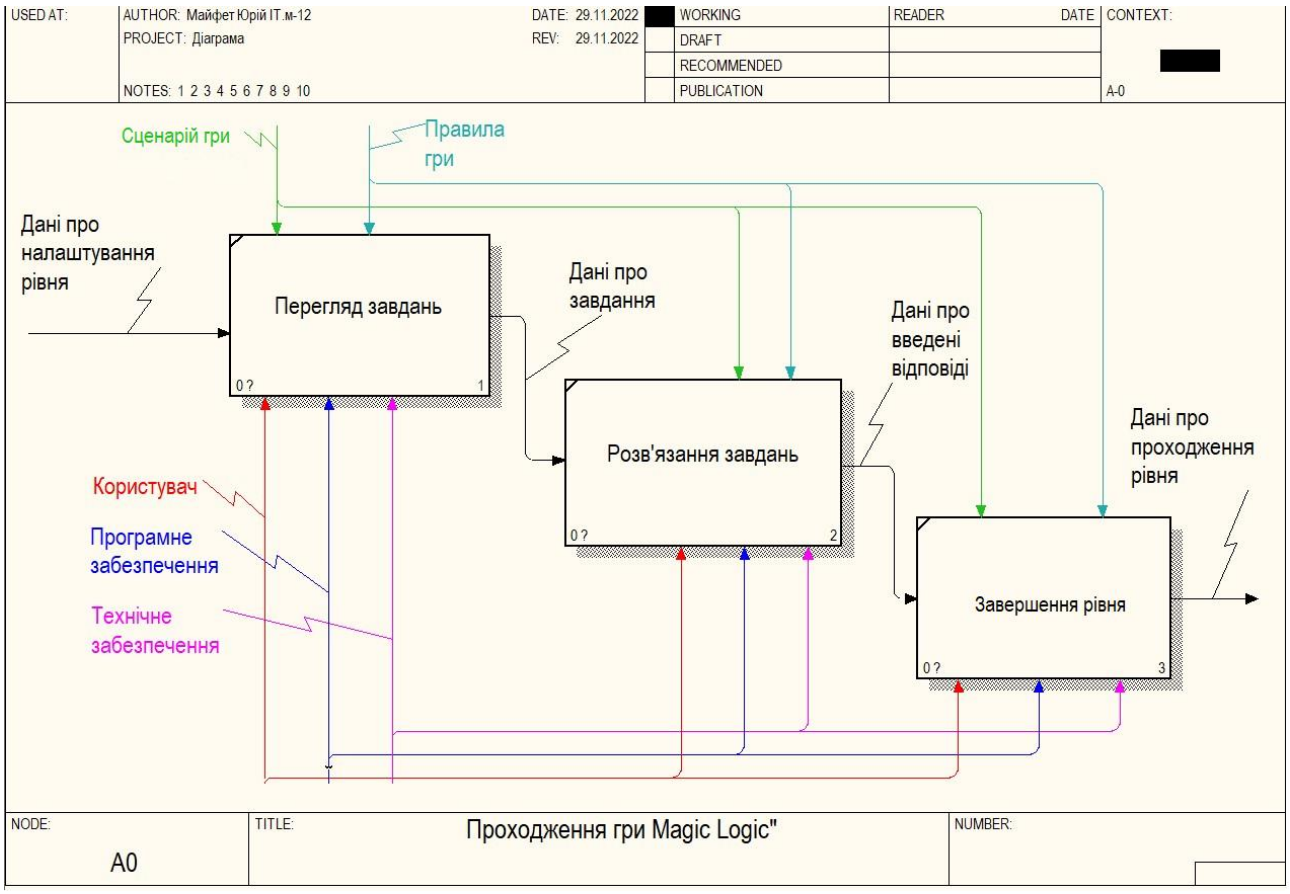


Рисунок 3.4 – Декомпозиція блоку «Проходження рівня»

3.2 Моделювання варіантів використання

Моделювання варіантів використання потрібне для повного опису дій системи у відповідь на взаємодію користувача. Також кожна можлива дія користувача повинна бути описана та мати свій ремзультат [40].

На рисунку 3.5 представлена діаграма варіантів використання для додатку «Magic Logic».

На зображеній діаграмі присутній актор, який є гравцем, та можливі варіанти використання додатку:

- головне меню;
- вихід;
- обрання рівня;

- проходження рівня;
- перевірка відповідей;
- збереження результату.

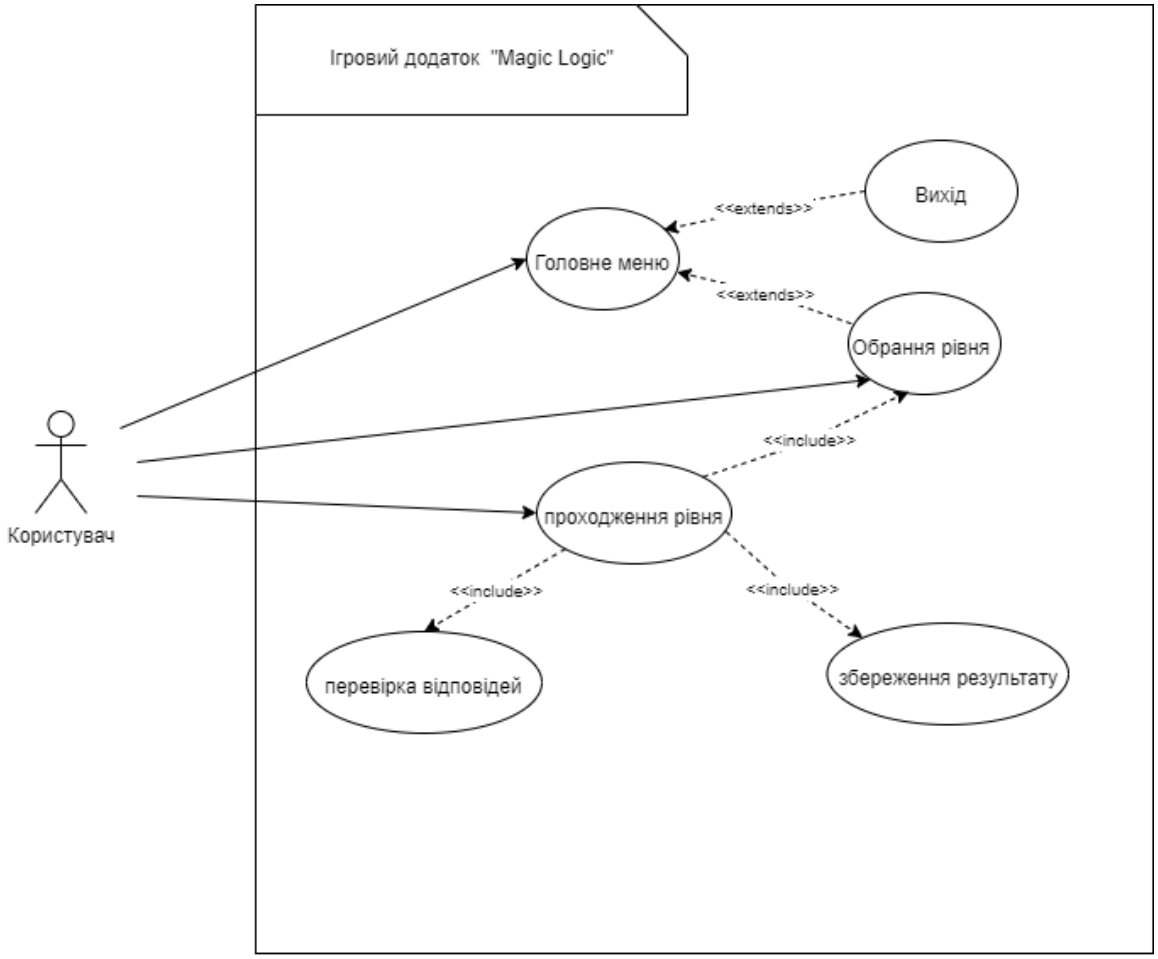


Рисунок 3.5 – Діаграма варіантів використання ігрового додатку «Magic Logic»

Всі варіанти мають <<include>> та <<extends>> зв’язки між собою, які показують варіанти взаємодії між собою. Зв'язок <<extends>> демонструє можливий, нестандартний, варіант використання, а в свою чергу зв'язок <<include>> ілюструє звичайний варіант використання.

4 ПРАКТИЧНА РОЗРОБКА МОБІЛЬНОГО ДОДАТКА

4.1 Розробка прототипу мобільного ігрового додатку

Щоб дати оцінку можливості реалізацій запланованих ігрових механік та створити ядро проекту, в першу чергу, необхідно створити працюючий прототип [32].

Так як розроблюваний ігровий додаток має жанр «логіка», то спершу потрібно створити примітивний інтерактивний рівень з декількома ключовими ігровими функціями:

- функція виводу запитань гравцю;
- функція вибору відповіді на запитання;
- функція малювання.

Створюємо новий проект в ігровому рушії Unity та призначаємо початкові налаштування під операційну систему Android.

Для розробки основних елементів створено елемент Canvas, який прив’язаний до головної камери. Цей елемент відповідатиме за відображення інтерфейсу гри та ігрових об’єктів. Далі створено декілька інтерактивних кнопок, які будуть надавати інформацію про відповідь гравця, а також текст з питанням. Для інтерактивності кнопок також потрібен елемент EventSystem.

Також в існуючому прототипі потрібно створити область для малювання та додати скрипти для функціональності. Функція малювання реалізується за допомогою компоненту Line, який у свою чергу має компонент Line Renderer. Додаємо обраний інструмент та налаштовуємо необхідний колір та товщину створеної лінії (рис. 4.1).

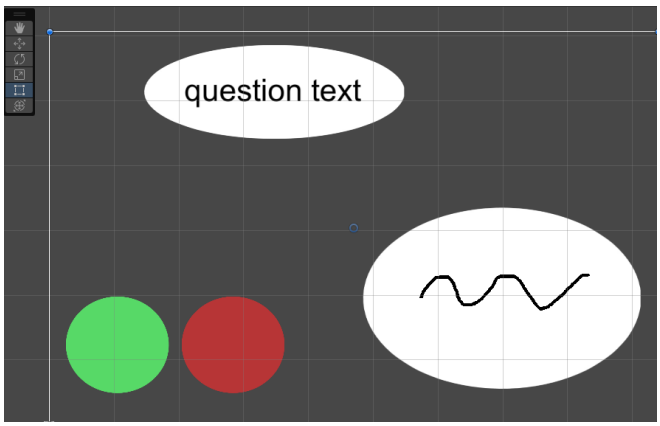


Рисунок 4.1 – Готова робоча область зі створеною лінією Line

Наступним кроком є реалізація скриптів, що описує поводження створених компонентів, в ігровій області, та додавання їх в елементи за допомогою вікна Inspector. Для цього створюємо два C# скрипта (додаток Б):

- скрипт відповіді на запитання;
- скрипт малювання лінії.

При використанні компоненту Line виявлені проблеми з неможливістю повного налаштування цього компоненту згідно поставлених задач, а саме:

- компонент Line може створити лише одну лінію у визначеній області;
- неможливо зазначити координати області, що в подальшому може викликати суттєві проблеми.

Тому в прототипі залишено дану реалізацію, а в подальшій розробці додатка створено власний аналог компоненту Line, який вирішує дані проблеми.

Після отримання працюючого прототипу додатка, на нього нарощено новий функціонал. Приклад працюючого прототипа додатка наведено на рисунку 4.2.

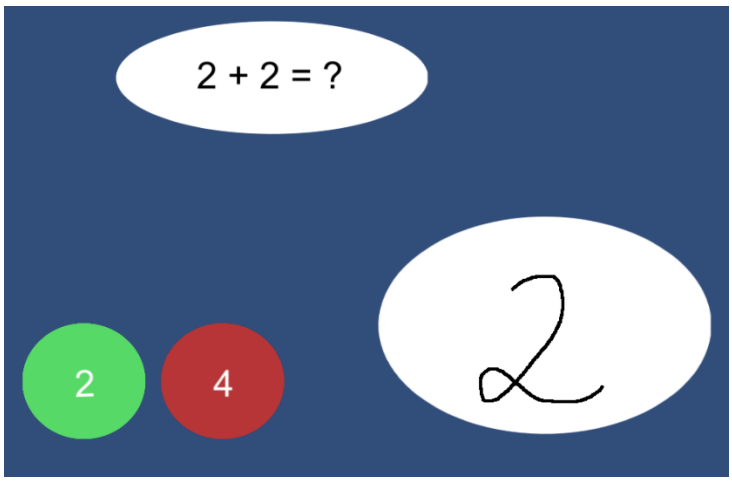


Рисунок 4.2 – Демонстрація працюючого прототипа додатка

4.2 Створення інтерфейса додатка

Після створення прототипу, наступним кроком є створення спрайтів додатку за допомогою графічного редактора Adobe Photoshop. Спочатку потрібно створити ескізи та обрати серед них найкращий варіант.

В першу чергу створені спрайти для головного меню додатку. Процес створення ескізів головного меню додатку зображено на рисунку 4.3.

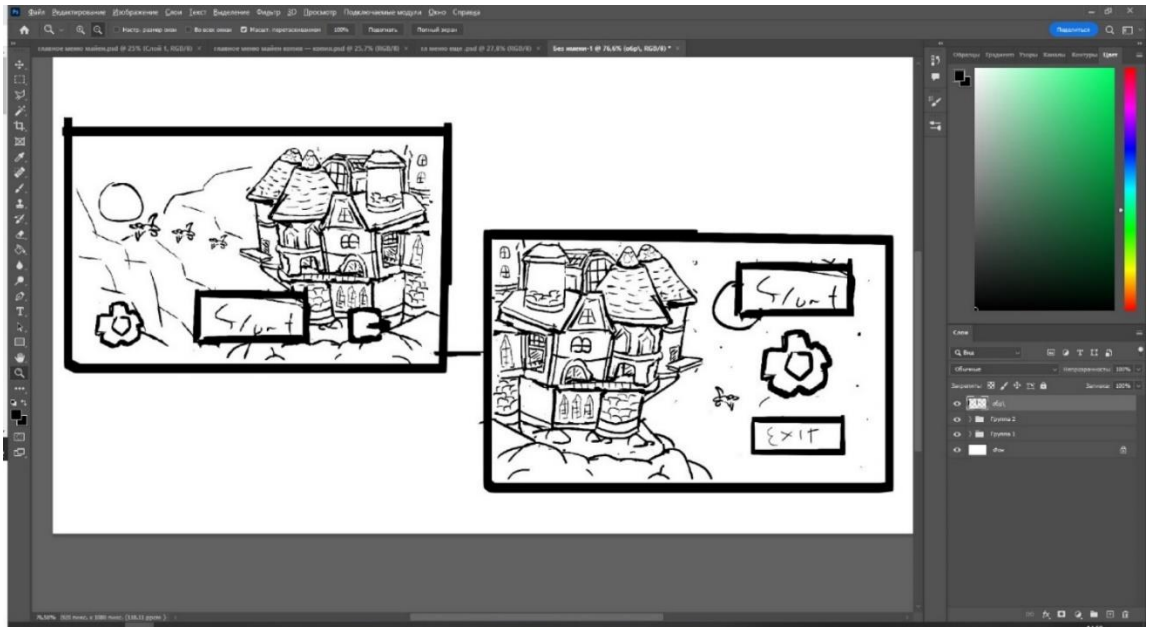


Рисунок 4.3 – Створення спрайтів головного меню

Після створення ескізу, його потрібно розбити на окремі компоненти та відредагувати в спеціалізованій програмі для створення піксель-арт спрайтів «Aseprite», що зображено на рисунку 4.4.

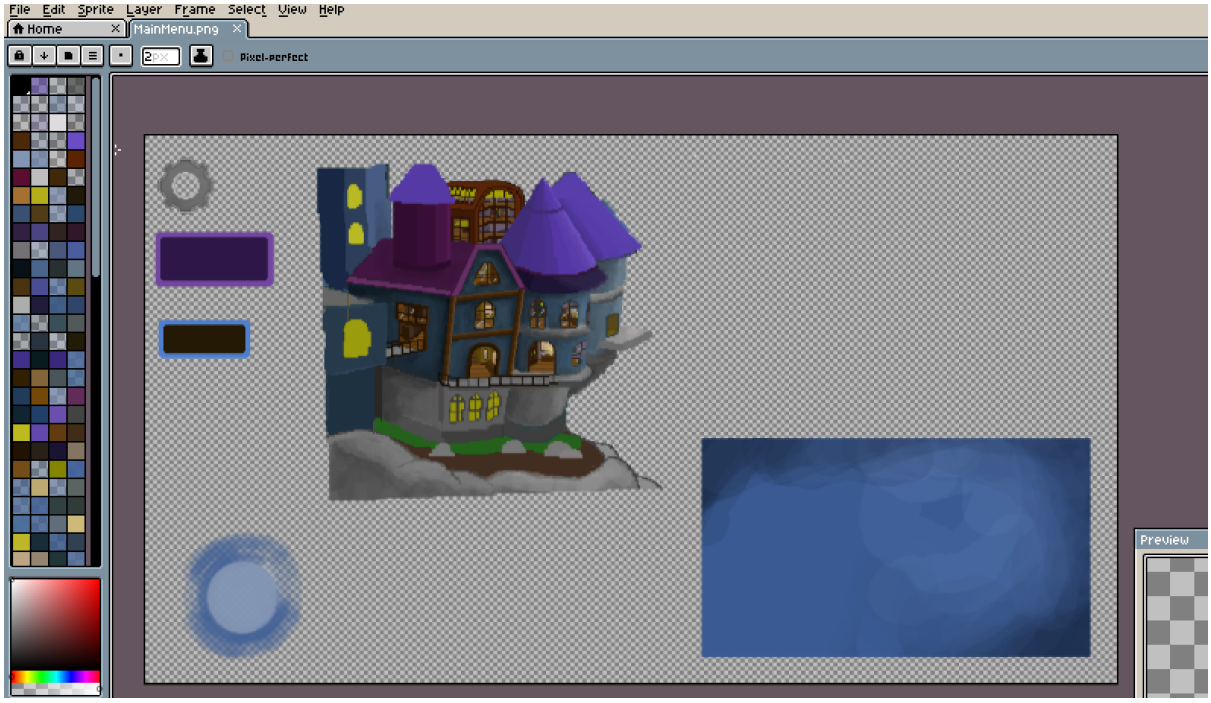


Рисунок 4.4 – Редагування спрайтів головного меню в «Aseprite»

Після імпорту створеної графіки в ігровий рушій, розміщуємо частини спрайтів на головному екрані (рис. 4.5). В результаті ми маємо шаблон головного меню з активним кнопками:

1. **Start –** відсилає гравця на меню вибору типу гри;
2. **Settings –** відсилає гравця в меню налаштувань додатку;
3. **Exit –** вихід з додатку.

Рисунок 4.5 – Головне меню додатку

Далі такі самі дії виконуємо для меню налаштувань (рис. 4.6), яке має:

1. **Sound –** відповідає за налаштування звуку;
2. **Language –** відповідає за налаштування мови;
3. **AI Settings –** відповідає за роботу алгоритму розпізнавання зображень;
4. **About –** відповідає за надання інформації про додаток та розробника;
5. **Стрілка вліво –** повертає гравця на головне меню.

Рисунок 4.6 – Меню налаштувань гри в розділі «Language»

Наступним є меню вибору типів гри (рис. 4.7), що містить такі активні компоненти:

1. **Calculations –** гравець, використовуючи зону для малювання у вигляді пергаменту, повинен намалювати правильну цифру або математичний знак у

відповідь на арифметичне питання.

2. **Lost letter –** гравець, використовуючи зону для малювання у вигляді пергаменту, повинен намалювати правильну пропущену букву щоб утворилося слово. Мова слів залежить від обраної мови в налаштуваннях додатку.

3. **Equality –** гравець, використовуючи інтерактивні кнопки посередині екрану, повинен обрати один з трьох знаків рівності чисел, порівнюючи два числа з лівої та правої сторони.

4. **Odd word out –** гравцеві пропонується набір слів з яких він повинен обрати зайве та дати відповідь натиснувши на нього.

5. **Next –** робить перехід на меню вибору рівня.

6. **Стрілка вліво –** повертає гравця на головне меню.

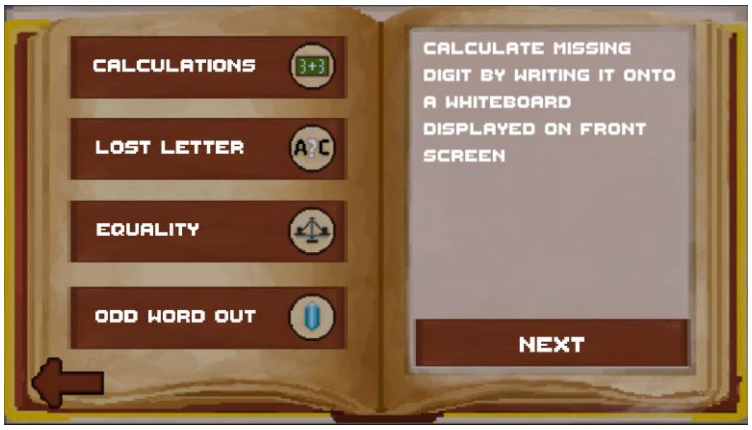


Рисунок 4.7 – Меню вибору типу гри

Обравши один з чотирьох типів гри, гравець потрапляє в меню вибору рівня гри (рис. 4.8). Меню вибору рівня гри має такі активні компоненти:

1. **Шість кнопок для вибору рівня –** гравець обирає один з шести бажаних рівнів обраного типу гри.

2. **Play –** кнопка початку гри.

3. **Стрілки вгору та вниз –** відповідають за перегляд дошки результатів гравців, що грали в обраний рівень, та кількість отриманих балів.

4. **Checkbox No Timer –** налаштування рівня, що дозволяє вимкнути таймер при проходженні рівня, але вимикання таймеру віднімає десять балів від

фінального результату.

5. **Checkbox No Fail –** налаштування рівня, що дозволяє додати ігнорування помилок гравця при проходженні рівня, в результаті рівень не завершується, якщо набрати більше шести помилок, але віднімає п’ять балів від фінального результату.

Рисунок 4.8 – Меню вибору рівня гри

Далі створено інтерфейси для кожного типу гри. Перший тип гри «Calculations» – серед усіх намальованих ескізів рівня розглянуто два найвдаліших (рис. 4.9).

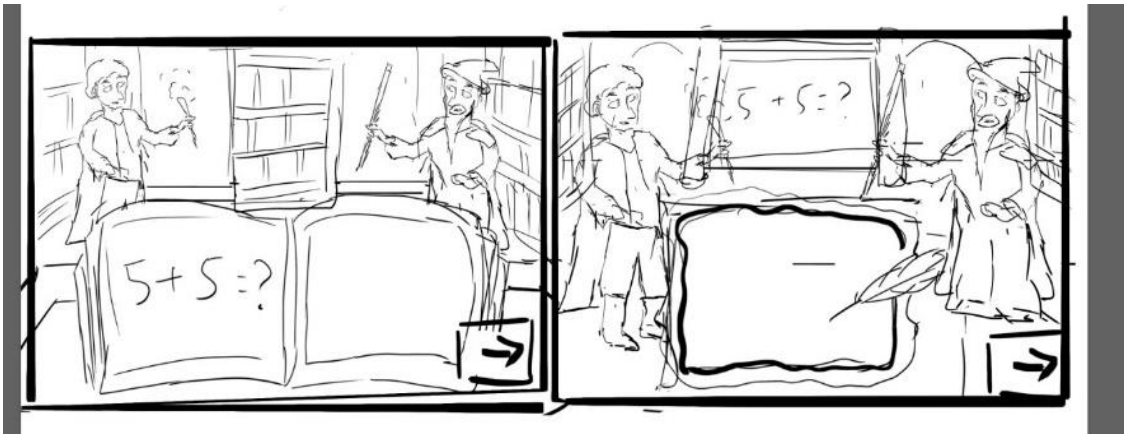


Рисунок 4.9 – Ескізи для типу гри «Calculations»

Після тестування обох варіантів на екрані смартфону, зручності вводу та масштабування тексту питань, було вирішено обрати другий варіант та допрацювати його.

Далі розділяємо та редагуємо отримані спрайти в редакторі «Aseprite» та отримуємо готовий результат, який імпортовано в рушій Unity (рис. 4.10).

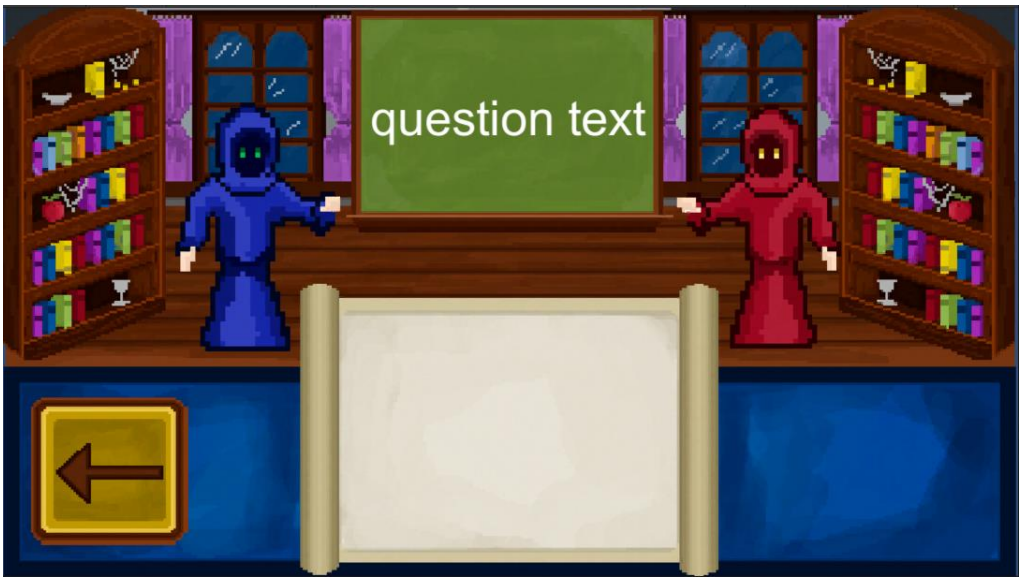


Рисунок 4.10 – Готовий шаблон типу гри «Calculations»

Далі так само створюємо ескіз для типу гри «Lost letter» (рис. 4.11)

Рисунок 4.11 – Ескіз для гри «Lost letter»

Та імпортуємо готовий відредагований варіант в ігровий рушій (рис. 4.12).

Рисунок 4.12 – Готовий шаблон гри «Lost letter»

Для економії часу останні два типи гри «Equality» та «Odd word out» будуть мати спільні спрайти, тому створюємо для них ескізи (рис. 4.13) та тестуємо кожен на сумісність з маленьким екраном смартфону. Фінальний обраний варіант після редагування імпортуємо в Unity (рис. 4.14).

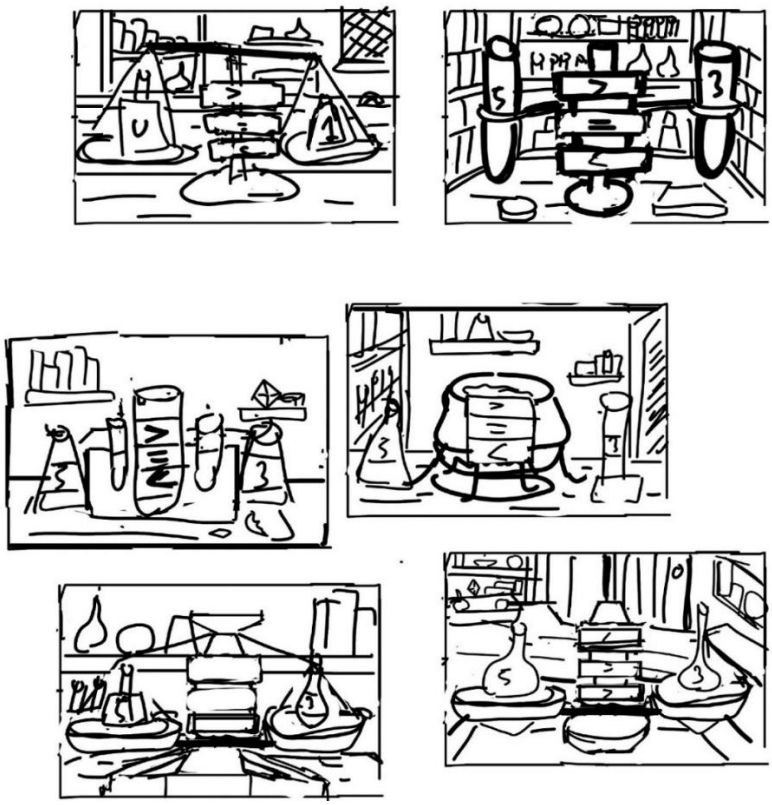


Рисунок 4.13 – Створення ескізів для типів гри «Equality» та «Odd word out»

Рисунок 4.14 – Готовий шаблон для типів гри «Equality» та «Odd word out»

Імпортовані зображення розбиваємо на компоненти за допомогою інструмента «Sprite editor». Для цього обираємо зображення і ставимо налаштування «Sprite Mode» на «Multiple» та «Point (no filter)» в налаштуванні «Filter Mode» (рис. 4.15), щоб прибрати розмиття зображення по краях. В результаті ми отримуємо розділені між собою зображення, які можна використовувати незалежно одне від одного.

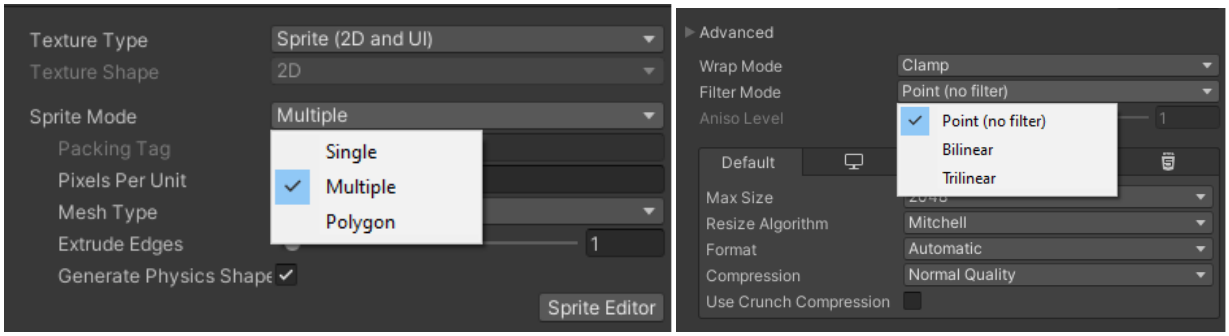


Рисунок 4.15 – Налаштування параметрів

4.3 Створення алгоритму розпізнавання зображень

Створюваний алгоритм розпізнавання зображень буде працювати за допомогою визначених координат в області малювання.

Для цього замінимо стандартний для Unity компонент Line на власну розробку. Першим кроком необхідно створити об’єкт рендеру лінії. Створюємо C# скрипт з назвою «LineRenderResolver», спираючись на офіційну документацію Unity для визначення стандартних методів. Також додаємо власний функціонал у вигляді виведення координат, які перетнула лінія, в області малювання. Фрагмент кода скрипту наведено на рисунку 4.16.

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

namespace DrawnFiguresRecognizer {

    public class LineRenderResolver : MaskableGraphic {

        [SerializeField]
        Texture lineTexture;

        public float LineDiametrSize = 3;
        public Vector2[] LinesPoints;

        public override Texture mainTexture {
            get {
                return m_LineTexture == null ? s_WhitTexture : lineTexture;
            }
        }

        public Texture texture {
            get {
                return m_LineTexture;
            }
            set {
                if (m_LineTexture == value)
                    return;
                m_LineTexture = value;
            }
        }


        protected override void OnPopulateMesh(VertexHelper vertexHelp) {
```

Рисунок 4.16 – Частина коду скрипту LineRenderResolver

Наступним кроком створимо «ScriptableObjects» об’єкт фігури, яку в майбутньому потрібно буде розпізнати. Створюємо C# скрипт з назвою «FiguresData» та створюємо сховище координат, за якими буде визначатись фігура. Код скрипта наведено у додатку Б.

Далі для отримання координат фігури запустимо проект та намалюємо цифру «2» та збережемо координати, які перетнула лінія в області для малювання (рис. 4. 17), а далі заносимо результат в створене сховище фігури (рис. 4.18).

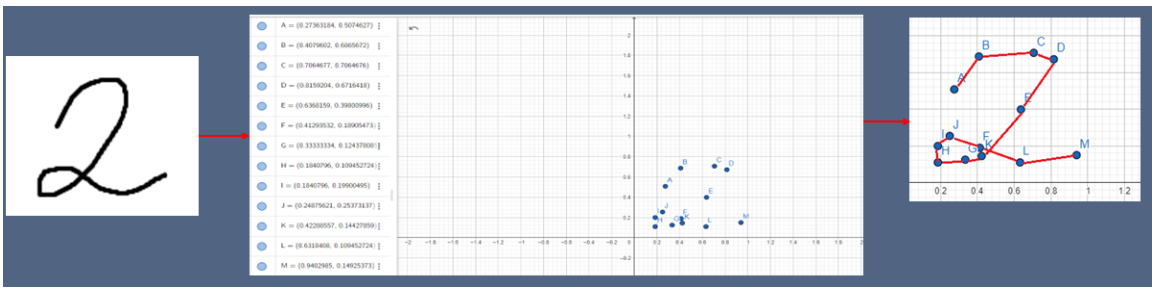


Рисунок 4.17 – Отримання координат

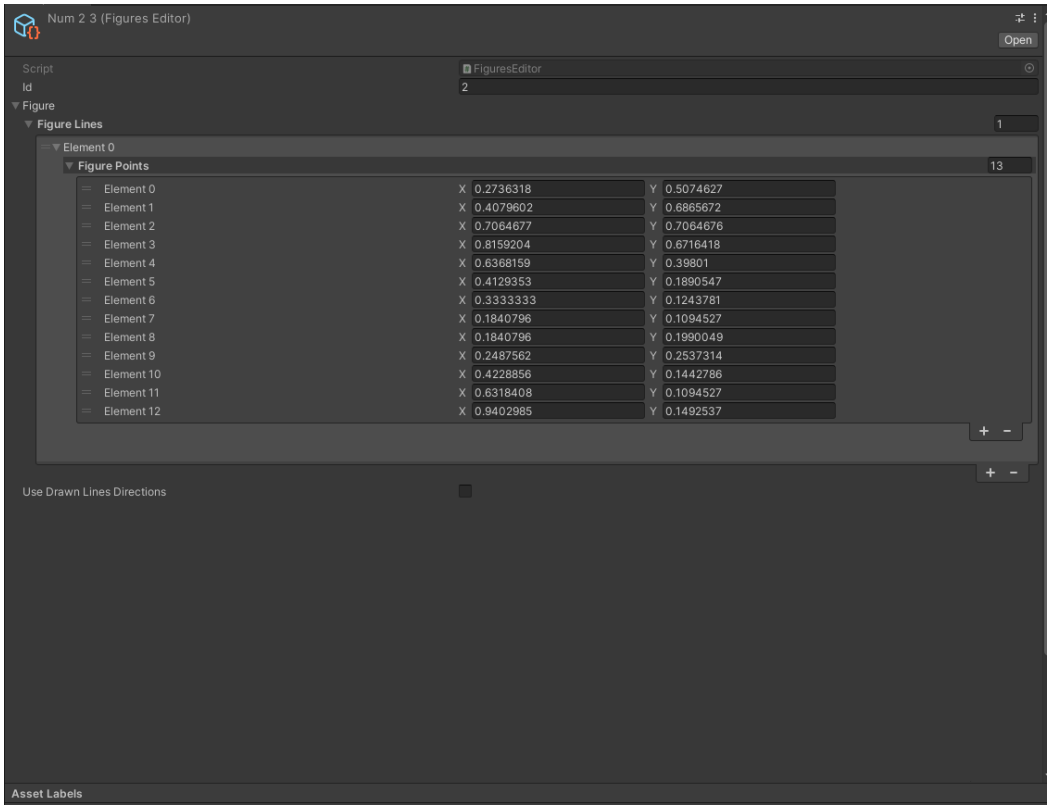


Рисунок 4.18 – Сховище фігури реалізоване скриптом «FiguresData»

Останнім кроком створюємо скрипт «FiguresRecognizer», який буде порівнювати координати лінії та видавати результат малювання (рис. 4.19).

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

namespace DrawnFiguresRecognizer
{
    public struct RegonizeScore
    {
        public float lineDistancePosition;
        public float lineDistanceAngle;


        public void InitiateMax()
        {
            lineDistancePosition = lineDistanceCurvature = lineDistanceAngle = float.MaxValue;
        }


        public static bool operator >(RegonizeScore score1, RegonizeScore score2)
        {
            return score1.recognizeScore > score2.recognizeScore;
        }
        public static bool operator <(RegonizeScore score1, RegonizeScore score2)
        {
            return score1.recognizeScore < score2.recognizeScore;
        }
        public static bool operator >=(RegonizeScore score1, RegonizeScore score2)
```

Рисунок 4.19 – Фрагмент коду скрипту «FiguresRecognizer»

Переносимо регулювання якості розпізнавання алгориму в налаштування. Для цього підв’язуємо скрипт «FiguresRecognizer» до створеного інтерактивного повзунка (рис. 4.20).

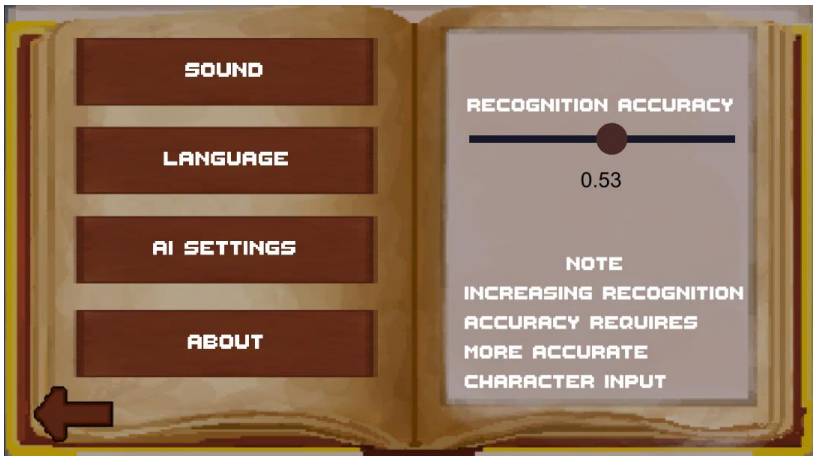


Рисунок 4.20 – Регулювання алгоритму скрипту «FiguresRecognizer» в налаштуваннях

Тестуємо алгоритм в реальному часі. Як бачимо, при малюванні фігури, виконується розпізнавання, що показано через функцію «Debug» в консолі (рис. 4.21), тобто алгоритм працює.

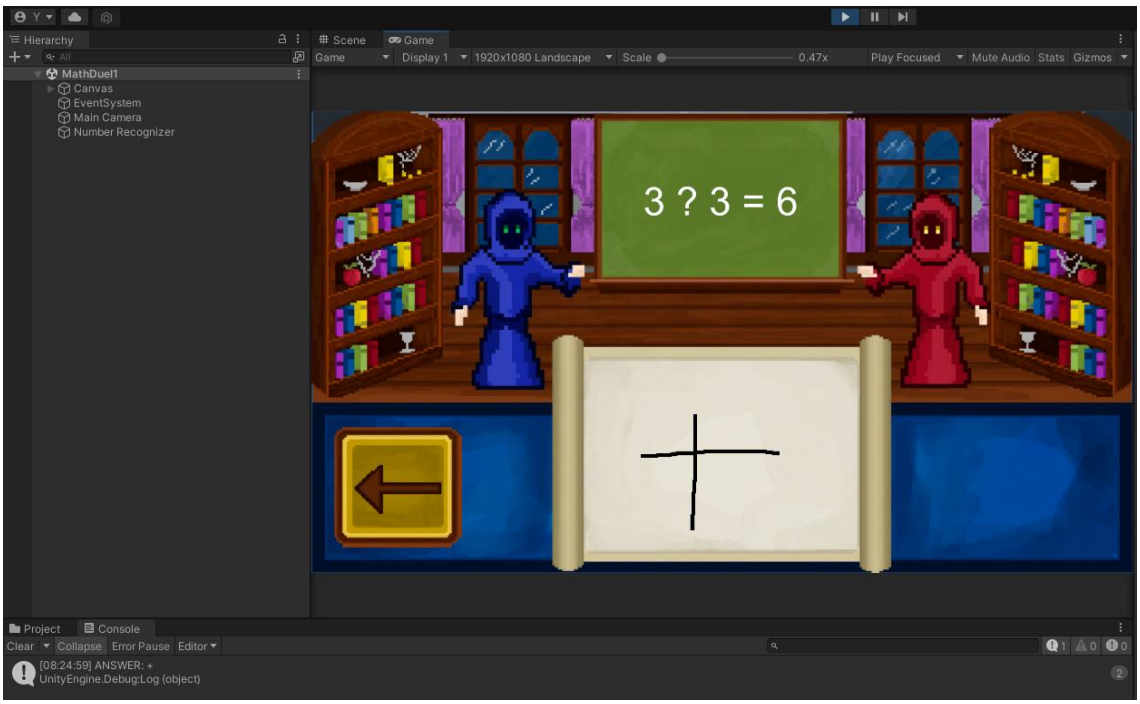


Рисунок 4.21 – Тестування алгоритму розпізнавання зображень

4.4 Створення рівнів та сховища завдань

Після готового алгоритму розпізнавання зображень, в нас є всі компоненти для створення рівнів. В якості ядра використаємо заздалегідь заготовлений прототип та імпортовані спрайти, які розташуємо згідно створеному шаблону.

І почнемо з головного меню. Для переходу між сценами створимо новий C# скрипт з назвою «Buttons», який буде відповідати за натискання кнопок. Імпортуємо бібліотеку «UnityEngine.SceneManagement» для можливості використання методу «LoadScene» та дописуємо функціонал для преходу між сценами (рис. 4.22).

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Buttons : MonoBehaviour
{
    public void loadScene (string sceneName)
    {
        SceneManager.LoadScene(sceneName);
    }

    public void closeApp()
    {
        Application.Quit();
    }
}
```

Рисунок 4.22 – Скрипт переходу між створеними сценами проекту

Далі нам необхідно додати об’єкт «EventSystem» та перекинути створений скрипт через інспектора до компоненту у створеній ієрархії, для зручності додамо його в головну камеру (рис. 4.23).

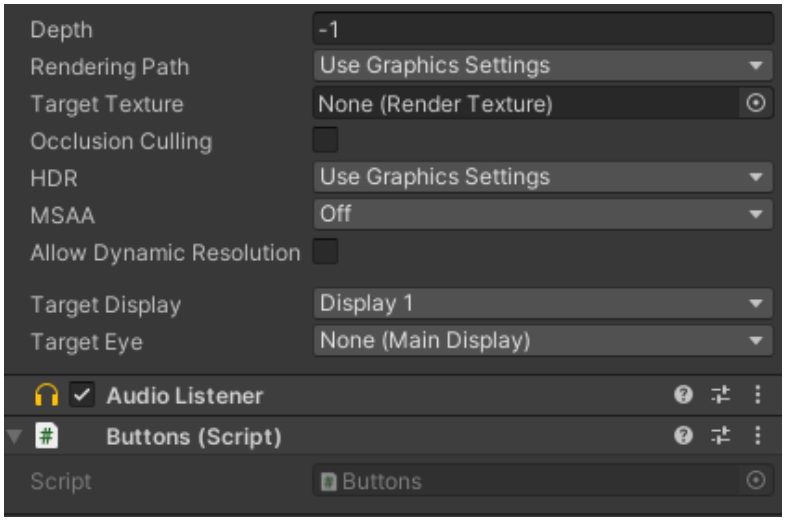


Рисунок 4.23 – Додавання скрипту «Buttons» до компонентів Main Camera

Та створимо подію переходу між сценами при натисканні на кнопу «Start» (рис. 4.24).

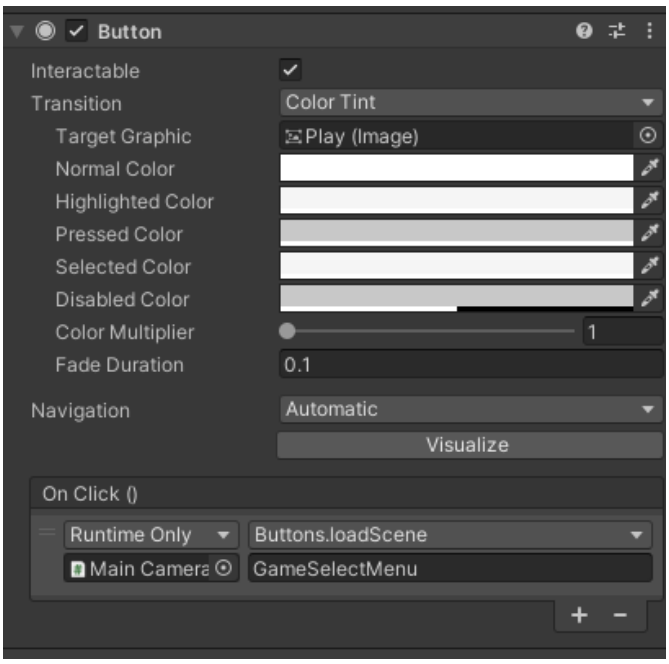


Рисунок 4.24 – Створення події переходу

Перед тестуванням створеного скрипту, нам необхідно додати сцену в збірку, перетягнувши її з вікна ассетів до вікна збірки (рис. 4.25). Бо якщо сцена для переходу відсутня в списку використаних сцен, отримаємо повідомлення про помилку з неможливістю знайти потрібну сцену за введеним ім’ям.

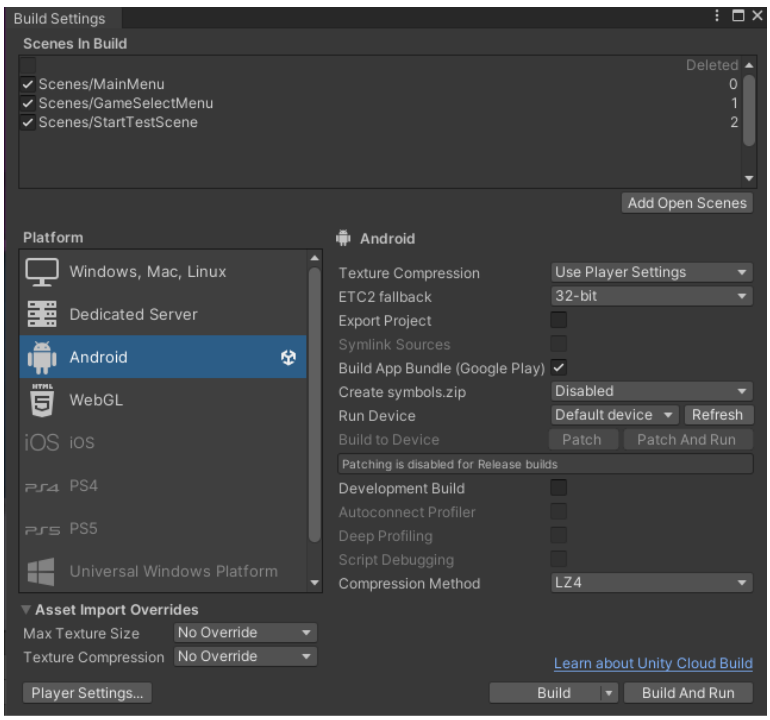


Рисунок 4.25 – Додавання створених сцен до збірки проекту

Після успішного тестування скрипту, аналогічно додаємо перехід на наступну сцену для налаштувань та виходу з додатку.

Фінальна версія головного меню з повною ієрархією об’єктів зображена на рисунку 4.26.

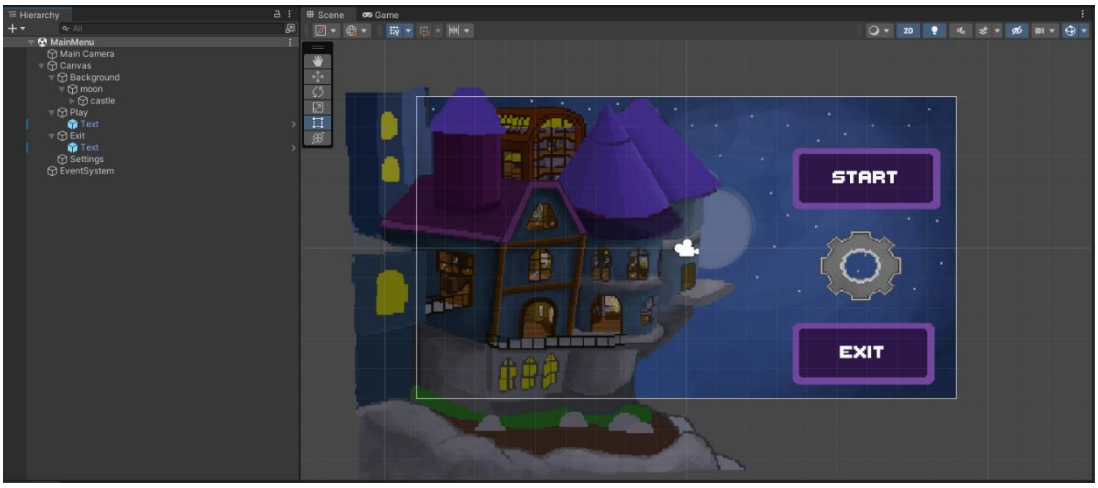


Рисунок 4.26 – Фінальна версія головного меню з ієрархією ігрових об’єктів

Далі створюємо меню вибору режимів гри. Такий тип меню відрізняється від головного тим, що при виборі типу гри, гравця не повинно одразу відсилати на наступну сцену. Для вирішення проблеми створюємо новий C# скрипт з назвою «SelectGameMode» (код скрипта в додатку Б). Даний скрипт робить активними лише ті ігрові об’єкти, на які натиснув гравець. На рисунку 4.27 зображено неактивний об’єкт вибору типу гри (позначено цифрою 1).

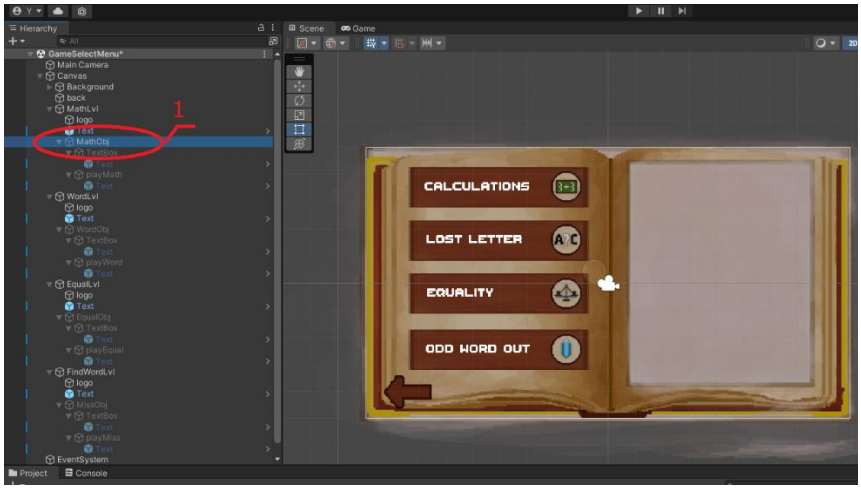


Рисунок 4.27 – Демонстрація неактивних ігрових об’єктів

При натисканні гравцем на категорію з необхідними назвами, скрипт автоматично вмикає опцію відображення тексту опису гри та кнопки «Next» (рис. 4.28) (позначено цифрою 2).

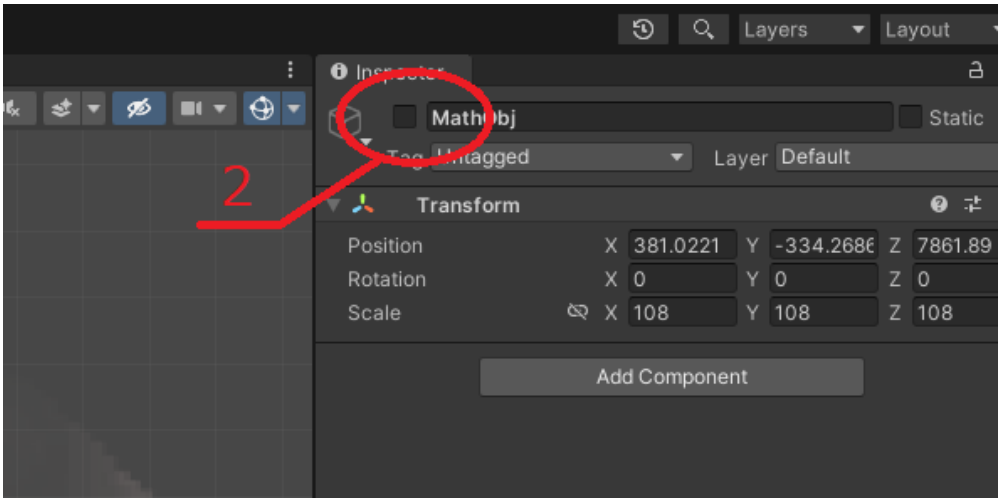


Рисунок 4.28 – Опція перемикання відображення ігрових об’єктів

Демонстрація роботи скрипту в меню вибору типів гри при натисканні на категорію «Lost Letter» зображено на рисунку 4.29.

Рисунок 4.29 – Реалізація перемикання видимості ігрових об’єктів

Створений скрипт є універсальним і може бути інтегрований в будь-яке меню. Тож виконуємо аналогічні дії для додавання функціональності в меню вибору рівнів та тестуємо результат (рис. 4.30).

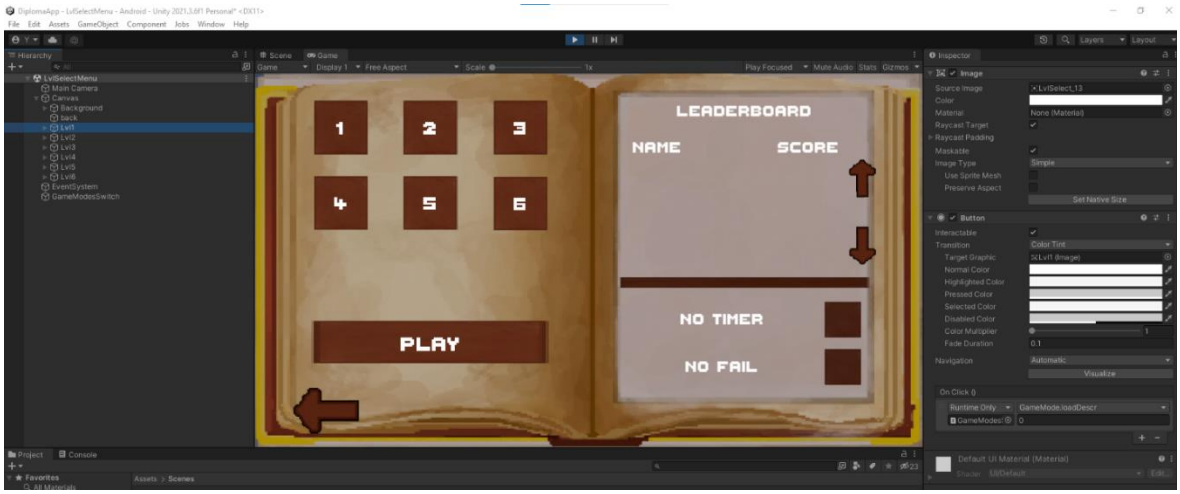


Рисунок 4.30 – Перемикання видимості об’єктів для меню вибору рівнів

Наступним кроком буде створення сховища даних завдань. Саме сховище буде у форматі NoSql бази даних, та реалізовано за допомогою вбудованих функцій ігрового рушія й мови програмування C#. Схожа реалізація розроблена для збереження координат фігур алгоритму розпізнавання зображень.

Спочатку створюємо C# скрипт з назвою Tasks, формат якого буде «ScriptableObjects», та створюємо об’єкт «TaskData». Далі створюємо ще один скрипт з налаштуваннями відображення створеної бази даних (код скриптів у додатку Б).

Зовнішній вигляд вмісту бази даних та файли бази у форматі «ScriptableObjects» зображено на рисунку 4.31.

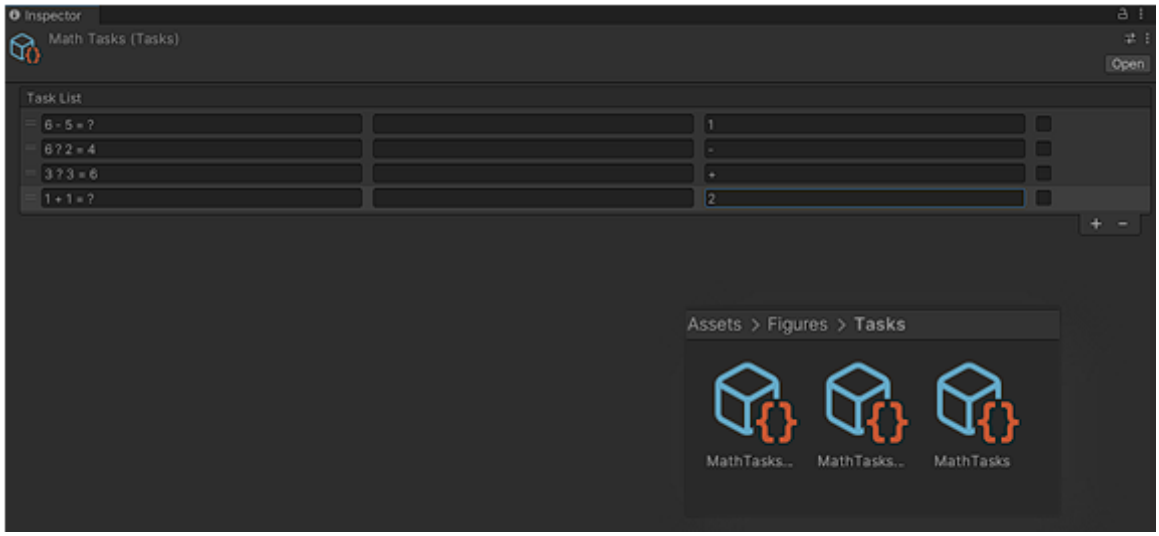


Рисунок 4.31 – Зовнішній вигляд вмісту бази даних завдань

Далі необхідно створити універсальниий скрипт, який міг би забирати будь-яку інформацію зі створеної бази даних. Створюємо скрипт з назвою «Result» та розробляємо логіку обробки інформації (код скрипта у додатку Б).

4.5 Створення файлу-інсталятора проекту

Після налаштування всіх основних елементів гри, потрібно створити файл-інсталятор, який можна запустити в операційній системі Android.

По-перше, заблокуємо орієнтацію екрану на «Landscape» для коректного відображення ігрових об’єктів (рис. 4.32).

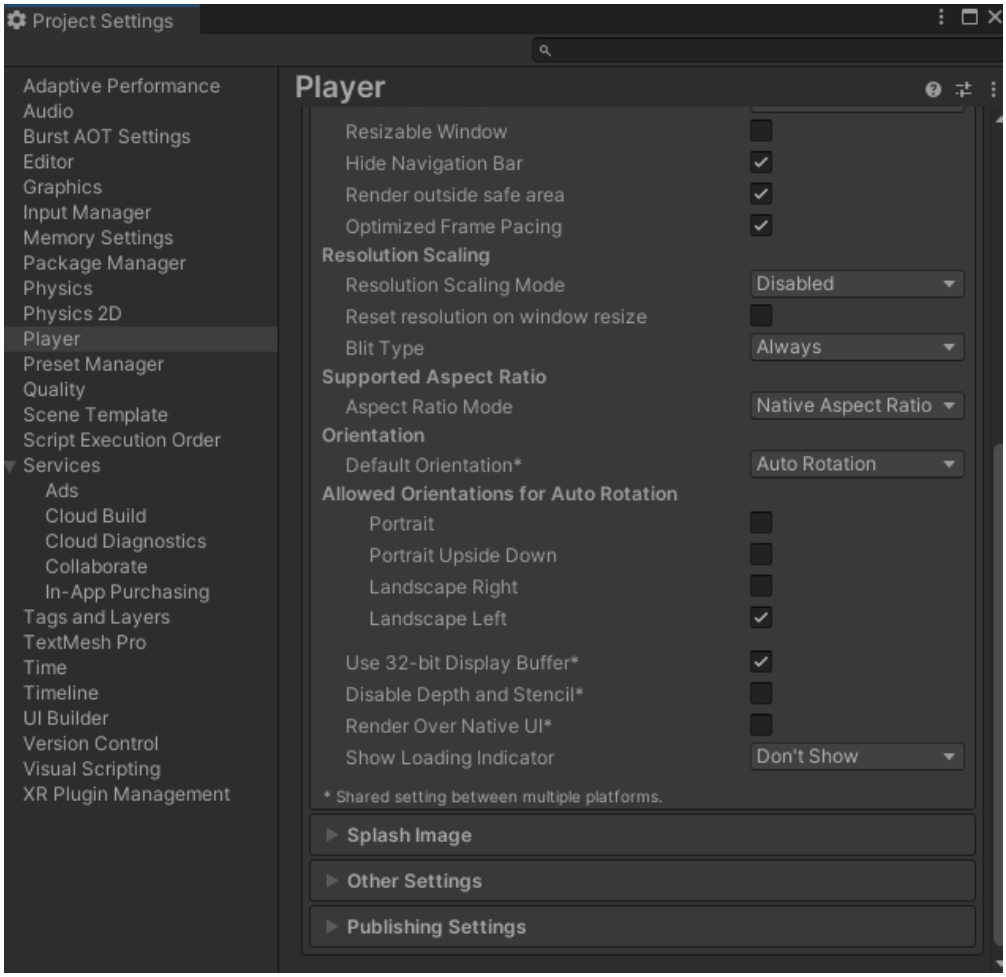


Рисунок 4.32 – Вікно налаштування орієнтації екрану

По-друге, додамо головне зображення додатку, обравши вікно Player Setting та праметр Default Icon (рис. 4.33).

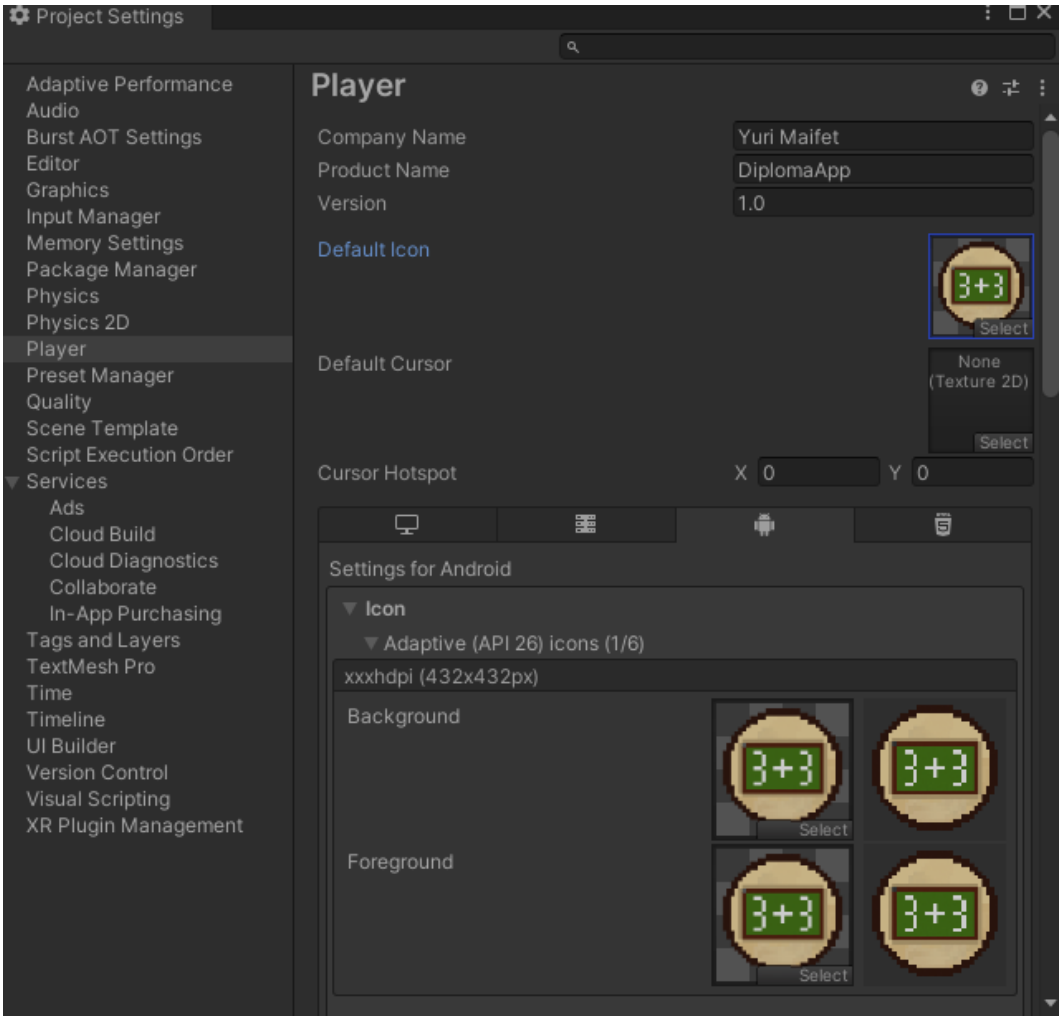


Рисунок 4.33 – Вікно налаштування вибору зображення додатку

4.6 Тестування фінальної збірки проекту

Після створення та налаштування всіх основних елементів гри, потрібно протестувати мобільний додаток безпосередньо на смартфоні.

Встановлюємо додаток завдяки створеному .apk файлу та запускаємо гру (рис. 4.34).

Рисунок 4.34 – Демонстрація некоректного налаштування розширення екрану

Першою поміченою проблемою є некоректне розташування елементів інтерфейсу. Щоб виправити даний недолік, потрібно додати в налаштування фіксоване розширення екрану. Після повторної збірки бачимо, що проблема зникла (рис. 4.35).

Рисунок 4.35 – Демонстрація виправленого налаштування розширення екрану

Далі тестуємо переходи між сценами головного меню та рівні додатку (рис. 4.36, 4.37).

Рисунок 4.36 – Демонстрація працюючого рівня Equality

Рисунок 4.37 – Демонстрація працюючого Lost Letter

Також тестуємо працездатність налаштувань (рис. 4.38).

Рисунок 4.38 – Демонстрація функції зміни мови

Тож в результаті були протестовані всі вікна меню, вікна налаштувань та працездатність ігрових рівнів і, виходячи з результату тестування, суттєвих проблем не було помічено.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи був проведений аналіз предметної області, знайдено та вивчено переваги і недоліки існуючих аналогів. Також розглянуто існуючі інструменти реалізації, які б допомогли в реалізації поставлених задач. В результаті обрано ігровий рушій Unity, редактор зображень Adobe Photoshop та текстовий редактор Microsoft Visual Studio.

Було проведено планування робіт проекту з формулюванням мети проекту та створенням відповідної структури робіт. Також створено та побудовано календарний план, матрицю відповідальності і визначено ризики проекту.

Створено повну структурно-функціональну декомпозицію проекту та схему варіантів використання додатку в графічному вигляді.

Створено комп’ютерну графіку та шаблони інтерфейсу, які потім було імпортовано до ігрового рушія.

Створено алгоритм розпізнавання зображень, який був інтегрований в створюваний додаток.

Розроблено власну NoSql базу даних завдань, у форматі «ScriptableObjects», та фігур для алгоритму розпізнавання зображень.

Також проведено тестування в ході якого були виявлені проблеми, виправлені в фінальній версії ігрового додатку.

В майбутньому даний проект може бути допрацьований і дану систему розпізнавання зображень можливо використовувати не тільки для розважального контенту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Reasons Why Mobile Gaming Is Becoming More And More Popular [Electronic resource] // https://www.gamegrin.com/articles/3-reasons-why-mobile-gaming-is-becoming-more-and-more-popular/. 2022.
2. Kids prefer to play games on mobile devices [Electronic resource] // https://www.engadget.com/2015-09-23-npd-study-on-kids-and-gaming.html. 2022.
3. Why Children Love Gaming on Phones [Electronic resource] // https://www.eriicta.am/why-children-love-gaming-on-phones/. 2022.
4. Kirginas S. et al. Studying children's experience during free-form and formally structured gameplay // Int J Child Comput Interact. 2021. Vol. 28.
5. Educational Game Apps for Kids & Adults in 2022 [Electronic resource] // https://www.capermint.com/blog/educational-mobile-games/. 2022.
6. Educational Android Games For Kids [Electronic resource] // https://techlog360.com/educational-android-games-for-kids/. 2022.
7. Best Educational Game Apps [Electronic resource] // https://www.educationalappstore.com/best-apps/our-5-best-apps-for-teaching-through-fun. 2022.
8. Taulli T. Artificial intelligence Basics - A Non-Technical Introduction // Apress. 2019. № 3.
9. Taulli T. Artificial Intelligence Basics // Artificial Intelligence Basics. 2019.
10. Luongo F. et al. Deep learning-based computer vision to recognize and classify suturing gestures in robot-assisted surgery // Surgery (United States). 2021. Vol. 169, № 5.
11. Tew A.I., Gray C.J. A real-time gesture recognizer based on dynamic programming // J Biomed Eng. 1993. Vol. 15, № 3.
12. Madhavan J. et al. Wheat seed classification using neural network pattern recognizer // Mater Today Proc. 2021.

13. Where is Artificial Intelligence Used Today? [Electronic resource] // https://itchronicles.com/artificial-intelligence/where-is-ai-used-today/. 2022.
14. Wang J., Liu X. Medical image recognition and segmentation of pathological slices of gastric cancer based on Deeplab v3+ neural network // Comput Methods Programs Biomed. 2021. Vol. 207.
15. Czajkowska J. et al. Automated segmentation of epidermis in high-frequency ultrasound of pathological skin using a cascade of DeepLab v3+ networks and fuzzy connectedness//Computerized Medical Imaging and Graphics.2022.Vol. 95.
16. John C.H.S., Balakrishnan N., Fiet J.O. Modeling the relationship between corporate strategy and wealth creation using neural networks // Comput Oper Res. 2000. Vol. 27, № 11–12.
17. Dr. Davide Aversa. Unity Artificial Intelligence Programming. 8th ed. 2022.
18. Zhang L. Hand-drawn sketch recognition with a double-channel convolutional neural network // EURASIP J Adv Signal Process. 2021. Vol. 2021, № 1.
19. Field M. et al. Sketch recognition algorithms for comparing complex and unpredictable shapes // IJCAI International Joint Conference on Artificial Intelligence. 2011.
20. Yesilbek K.T., Sezgin T.M. On training sketch recognizers for new domains // IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. 2021.
21. Uzun C., Çolakoğlu M.B. Architectural Drawing Recognition A case study for training the learning algorithm with architectural plan and section drawing images // Proceedings of the International Conference on Education and Research in Computer Aided Architectural Design in Europe. 2019. Vol. 2.
22. Belongie S., Malik J., Puzicha J. Shape matching and object recognition using shape contexts // IEEE Trans Pattern Anal Mach Intell. 2002. Vol. 24, № 4.
23. Mori G., Belongie S., Malik J. Efficient shape matching using shape contexts // IEEE Trans Pattern Anal Mach Intell. 2005. Vol. 27, № 11.
24. Hsieh H.C.L. Integration of environmental sustainability issues into the “game

design theory and practice” design course // Sustainability (Switzerland). 2020. Vol. 12, № 16.

25. Iii R., Ogden S. Game Design : Theory & Practice // Design. 2005.
26. Game design theory: a new philosophy for understanding games // Choice Reviews Online. 2013. Vol. 50, № 10.
27. NVIDIA. CUDA Toolkit 7.5 Documentation [Electronic resource] // Developer Zone. 2015.
28. Komura Y., Okabe Y. Improved CUDA programs for GPU computing of Swendsen–Wang multi-cluster spin flip algorithm: 2D and 3D Ising, Potts, and XY models // Comput Phys Commun. 2016. Vol. 200.
29. Araujo H., Dias J.M. An introduction to the log-polar mapping [image sampling]. 2002.
30. Computer vs. smartphone [Electronic resource] // https://www.computerhope.com/issues/ch001398.htm. 2022.
31. Laakso N.L., Korhonen T.S., Hakkarainen K.P.J. Developing students’ digital competences through collaborative game design // Comput Educ. 2021. Vol. 174.
32. Gallego-Durán F.J. et al. A guide for game-design-based gamification // Informatics. 2019. Vol. 6, № 4.
33. Ritchie D.M. et al. C PROGRAMMING LANGUAGE. // Western Electric engineer. 1981. Vol. 25, № 1.
34. Souli J. C ++ Language Tutorial // Cognition Technology Work. 2007. Vol. 5, №3.
35. Microsoft. A tour of the C# language [Electronic resource] // 2021. 2021.
36. Agrahari V., Chimalakonda S. What’s Inside Unreal Engine? - A Curious Gaze! // ACM International Conference Proceeding Series. 2021.
37. Twinmotion. Twinmotion - Unreal Engine // Epic Games. 2020.
38. Šmíd A. Comparison of Unity and Unreal Engine // Comparison of unity and unreal engine. 2017. № May.
39. Heaslip G., Mangan J., Lalwani C. Modelling a Humanitarian Supply Chain using the Structured Analysis and Design Technique (SADT) // University of

Hull Logistics Institute. 2010.

40. Hutauruk M.K. UML Diagram : Use Case Diagram // BINUS University. 2019.

ДОДАТОК А

А.1 Ідентифікація мети ІТ-проекту

Готовим продуктом проекту є мобільний ігровий додаток для розвитку логічного мислення «Magic Logic», який використовує штучний інтелект для розпізнавання зображень. Для ілюстрування результату, буде використано метод SMART (Specific, Measurable, Achievable, Relevant, Time-framed) у таблиці А.1.

Таблиця А.1 – Ідентифікація мети методом SMART

Specific	Створити мобільний ігровий додаток для розвитку логічного мислення школярів. Завдяки технології розпізнавання зображень ігровий процес буде цікавішим та зможе утримувати уваги на протязі всієї ігрової сесії.
Measurable	Відсутність готових відповідей та необхідність в самостійному вводі, спонукає до ретельного обмірковування правильної відповіді, що покращує ефективність навчання на 50%.
Achievable	Щоб створити цікавий додаток, який здатен привернути до себе увагу, необхідно ретельно вивчити та ознайомитися з функціоналом існуючих аналогів.
Relevant	Створити ігровий додаток, який буде мати штучний інтелект, що розпізнає намальовані користувачем фігури.
Time-Specific	Створення проекту ігрового додатку, який буде дотримуватись дедлайнів на основі календарного плану

А.2 Планування змісту структури робіт інформаційної системи

Для розуміння кожного етапу проекту була розроблена WBS структура розподілу робіт. На рисунку А.1 зображена WBS діаграма, яка має ієрархічну структуру, завдяки якій простіше зрозуміти процес створення проекту.

Рисунок А.1 – WBS структура проекту

Також окрім WBS структури, необхідно створити OBS (Organization Breakdown Structure) структуру виконавців.

OBS структура включає в себе всі сторони, які приймали участь в розробці і згідно правилам – чим вище блок, тим більше обов’язків має людина, яка зазначена в цьому блоці.

OBS-структура проекту наведена на рисунку А.2.

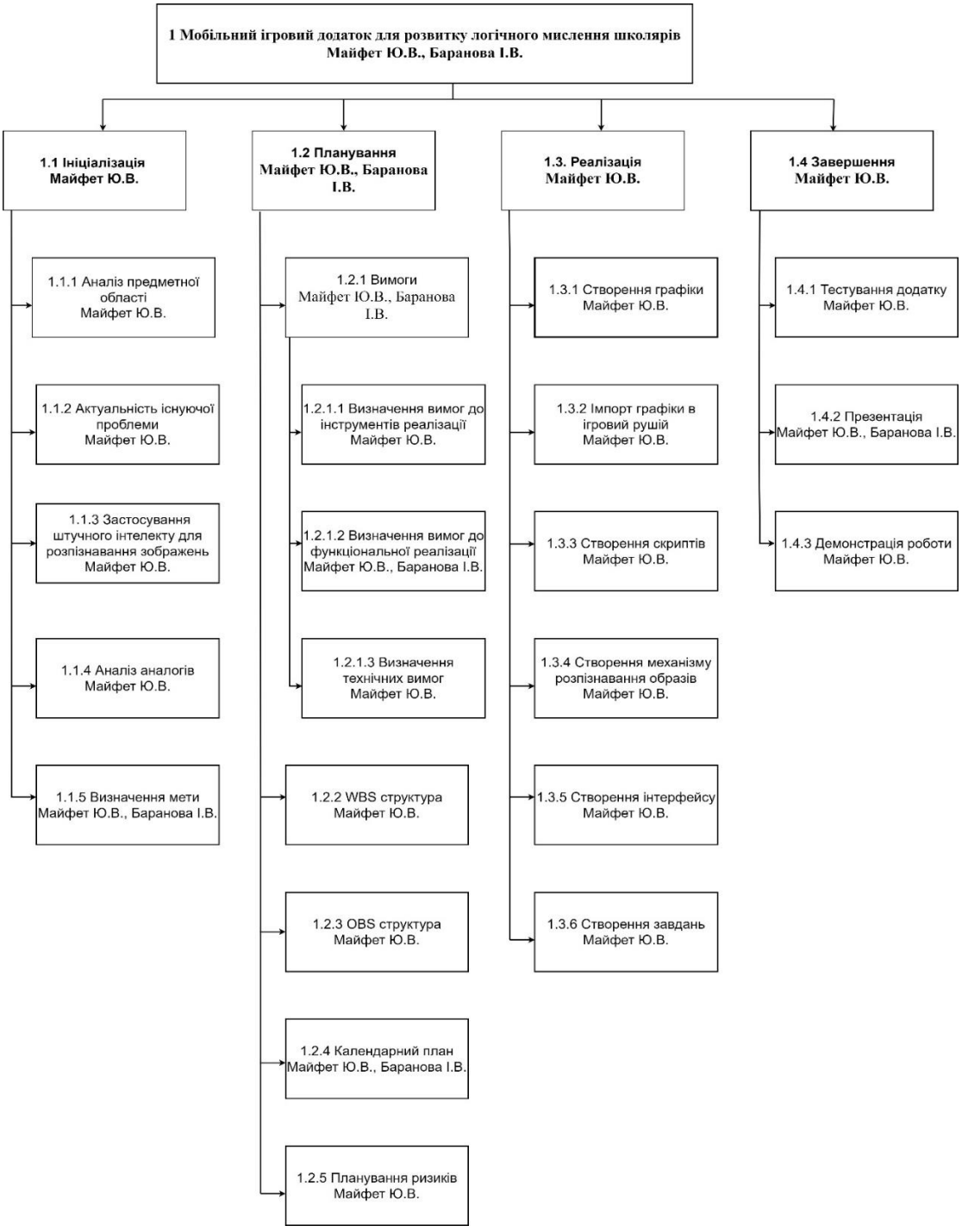


Рисунок А.2 – WBS структура проекту

Після проектування WBS та OBS структур необхідно побудувати матрицю відповідальності, що показує відповідальність виконавців залежно від ролі. Матрицю відповідальності проекту наведено у таблиці А.2.

Таблиця А.2 – Матриця відповідальності

Назва	**Виконавці**	
	Майфет Ю.В.	**Баранова І.В.**
Аналіз предметної області	+	
Актуальність існуючої проблеми	+	
Застосування штучного інтелекту для розпізнавання зображень	+	
Аналіз аналогів	+	
Аналіз аналогів	+	
Визначення мети	+	+
Планування	+	+
Визначення вимог до інструментів реалізації	+	
Визначення вимог до функціональної реалізації	+	+
Визначення технічних вимог	+	
WBS структура	+	
OBS структура	+	
Календарний план	+	+
Планування ризиків	+	
Створення графіки	+	
Імпорт графіки в ігровий рушій	+	
Створення скриптів	+	
Створення інтерфейсу	+	
Створення завдань	+	
Тестування додатку	+	
Презентація	+	+
Демонстрація роботи	+	

А.3 Побудова календарного графіку виконання інформаційної системи

Для оцінки тривалості виконання роботи над проектом було створено діаграму Ганта. Діаграма Ганта необхідна для відображення завдань та часу запланованого на ці завдання.

Завдяки зручній структурі діаграми, вона не потребує багато місця і є досить інформативною. Також вона допомагає відстежувати дати початку та тривалості виконання кожного етапу розробки.

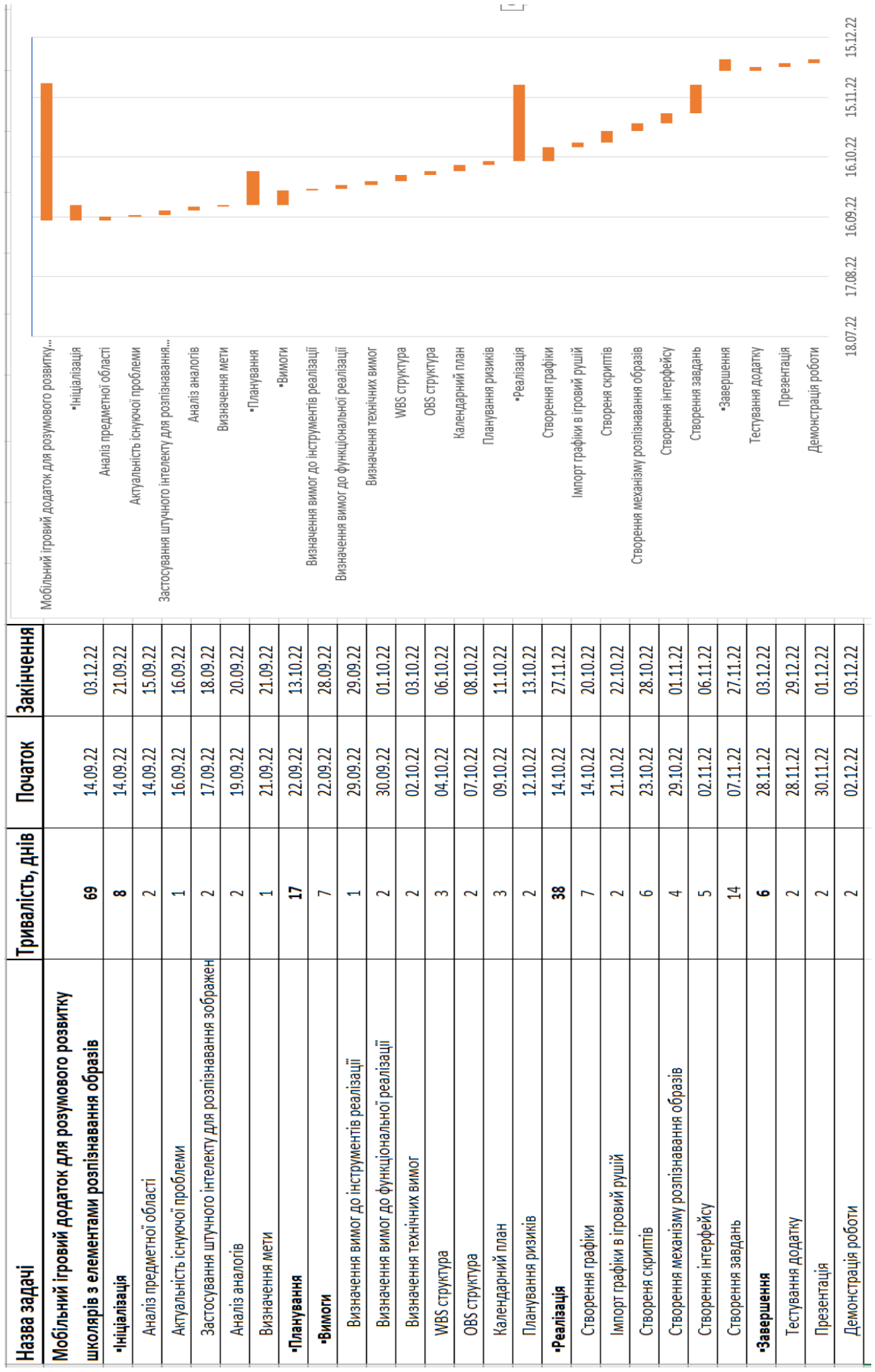
Побудована діаграма Ганта зображена на рисунку А.3.

А.4 Планування ризиків проекту

Розробка кожного проекту потребує планування ризиків. Якісне планування допоможе заздалегідь виявити майбутні проблеми та вжити заходів для їх усунення.

До головних ризиків під час розробки мобільного ігрового додатку можна віднести:

- відсутність електроенергії;
- поломка техніки;
- закінчення строку дії ліцензії на програмні продукти;
- відходження від строків виконання роботи зазначеного в календарному плані;
- людський фактор;
- відсутність інтернет зв’язку.



Назва задачі	Тривалість, днів	Початок	Закінчення
Мобільний ігровий додаток для розумового розвитку школярів з елементами розпізнавання образів	**69**	14.09.22	03.12.22
•Ініціалізація	**8**	14.09.22	21.09.22
Аналіз предметної області	2	14.09.22	15.09.22
Актуальність існуючої проблеми	1	16.09.22	16.09.22
Застосування штучного інтелекту для розпізнавання зображен	2	17.09.22	18.09.22
Аналіз аналогів	2	19.09.22	20.09.22
Визначення мети	1	21.09.22	21.09.22
•Планування	**17**	22.09.22	13.10.22
•Вимоги	7	22.09.22	28.09.22
Визначення вимог до інструментів реалізації	1	29.09.22	29.09.22
Визначення вимог до функціональної реалізації	2	30.09.22	01.10.22
Визначення технічних вимог	2	02.10.22	03.10.22
WBS структура	3	04.10.22	06.10.22
OBS структура	2	07.10.22	08.10.22
Календарний план	3	09.10.22	11.10.22
Планування ризиків	2	12.10.22	13.10.22
•Реалізація	**38**	14.10.22	27.11.22
Створення графіки	7	14.10.22	20.10.22
Імпорт графіки в ігровий рушій	2	21.10.22	22.10.22
Створення скриптів	6	23.10.22	28.10.22
Створення механізму розпізнавання образів	4	29.10.22	01.11.22
Створення інтерфейсу	5	02.11.22	06.11.22
Створення завдань	14	07.11.22	27.11.22
•Завершення	**6**	28.11.22	03.12.22
Тестування додатку	2	28.11.22	29.12.22
Презентація	2	30.11.22	01.12.22
Демонстрація роботи	2	02.12.22	03.12.22

Рисунок А.3 – Діаграма Ганта проекту

Згідно правил є такі критерії оцінки ризиків:

1. Ймовірність виникнення
2. Рівень ризику
3. Ступінь впливу
4. Ступінь втрат

В свою чергу кожен критерій включає в себе градацію по значущості

Ймовірність виникнення:

1. Мінімальна ймовірність
2. Низка ймовірність
3. Середня ймовірність
4. Висока ймовірність
5. Максимальна ймовірність

Рівень ризику:

1. Прийнятні
2. Виправдні
3. Недопустимі

Ступінь впливу:

1. Мінімальний
2. Незначний
3. Допустимий
4. Значний
5. Максимальний

Ступінь втрат:

1. Мінімальний
2. Незначний
3. Допустимий
4. Значний
6. Максимальний

Прийнявши до уваги кожен критерій, була складена матриця декомпозиції RBS (Risk Breakdown Structure) (рис. А.4) і наступні таблиці: таблиця А.3 для

ймовірності виникнення, А.4 для рівня ризику, А.5 для ступеня впливу, А.6 для ступеня втрат.

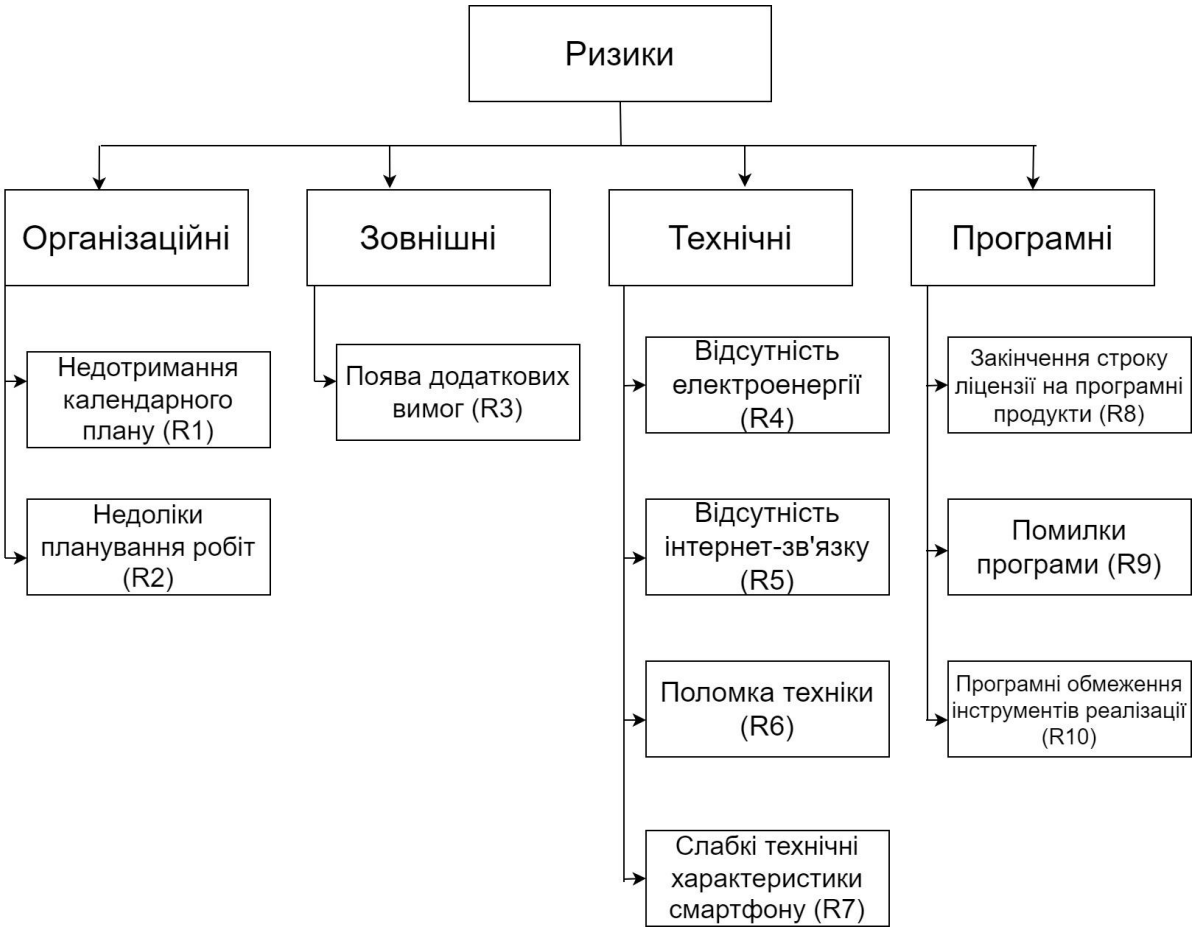


Рисунок А.4 – RBS матриця декомпозиції проекту

Таблиця А.3 – Матриця відповідальності ймовірності виникнення

Ймовірність виникнення		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Мінімальна										
2	Низька										
3	Середня										
4	Висока										
5	Максимальна										

Таблиця А.4 – Матриця відповідальності рівня ризику

Рівень ризику		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Прийнятний										
2	Виправданий										
3	Недопустимий										

Таблиця А.5 – Матриця відповідальності ступеня впливу

Ступінь впливу		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Мінімальний										
2	Незначний										
3	Допустимий										
4	Значний										
5	Максимальний										

Таблиця А.6 – Матриця відповідальності ступеня втрат

Ступінь втрат		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	Мінімальні										
2	Незначні										
3	Допустимі										
4	Значні										
5	Максимальні										

ДОДАТОК Б

Скрипт «Result»

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


namespace DrawnFiguresRecognizer
{

    public class Result : MonoBehaviour
    {
        public Tasks tasks;

        [SerializeField] public Timer _timer;

        private bool canAsk = false;

        [SerializeField] public Text _questionText;

        public FiguresDetector _drawDetector;

        public bool GetCanAsk()
        {
            return this.canAsk;
```

```
}

void Start()
{

    AskQuestion();
    ClearQuestions();
}

public void ReadyScreen(string answer)
{

}

public void ShowResults(string answer)
{
    this.canAsk = false;
    if (tasks.taskList[tasks.currentTask].answer == answer)
    {
        _drawDetector.ClearDrawnLines();
        StartCoroutine(ShowResults(true));
    } else
    {
        Debug.Log("ANSWER: " + answer);
        StartCoroutine(ShowResults(false));
    }
}

private IEnumerator ShowResults(bool correct)
{
```

```
        if (correct)
        {
            _questionText.text = _questionText.text.Replace("?",
tasks.taskList[tasks.currentTask].answer);
            tasks.taskList[tasks.currentTask].isQuestioned = true;
            yield return new WaitForSeconds(1.0f);
            AskQuestion();
        }
        else
        {
            //_questionText.text = " fail ";
            _drawDetector.ClearDrawnLines();
            this.canAsk = true;
        }
    }

    public void AskQuestion()
    {
        if (CountValidQuestions() == 0)
        {
            _questionText.text = string.Empty;
            ClearQuestions();
            SceneManager.LoadScene("MainMenu");
            return;
        }
        var randomIndex = 0;

        do
        {
            randomIndex = UnityEngine.Random.Range(0, tasks.taskList.Count);
```

```
            } while (tasks.taskList[randomIndex].isQuestioned == true);
            tasks.currentTask = randomIndex;
            _questionText.text = tasks.taskList[randomIndex].task1;
            tasks.taskList[tasks.currentTask] = tasks.taskList[randomIndex];
            this.canAsk = true;
        }

        public void ClearQuestions()
        {
            foreach (var question in tasks.taskList)
            {
                question.isQuestioned = false;
            }
        }

        private int CountValidQuestions()
        {
            int validQuestion = 0;
            foreach (var question in tasks.taskList)
            {
                if (question.isQuestioned == false)
                    validQuestion++;
            }
            return validQuestion;
        }
    }
}
```

Скрипт «FiguresData»

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEditor;
namespace DrawnFiguresRecognizer {

	[CustomEditor(typeof(FiguresEditor))]
	public class FiguresData : Editor{

	int index;

	SerializedProperty figures;

	SerializedProperty useDirection;


		void OnEnable()
		{
			figures = serializedObject.FindProperty("figure").FindPropertyRelative("figureLines");
			useOrder = serializedObject.FindProperty("useDrawnLinesOrder");
			useDirection = serializedObject.FindProperty("useDrawnLinesDirections");
		}

		int dragIndex;
```

```
        private static bool snap = false;

        int FindNear(SerializedProperty array, Vector2 pos)
        {
            int nearIndex = 0;
            float minDist = Vector2.Distance(array.GetArrayElementAtIndex(0).vector2Value, pos);
            for (int i = 1; i < array.arraySize; i++)
            {
                var dist = Vector2.Distance(array.GetArrayElementAtIndex(i).vector2Value, pos);
                if (dist < minDist)
                {
                    minDist = dist;
                    nearIndex = i;
                }
            }
            return nearIndex;
        }

        int CalcAboveLineIndex(SerializedProperty points, Vector2 pos, float lineDist)
        {
            float minDist = float.MaxValue;
            int minDistIndex = -1;

            for (int i = 0; i < points.arraySize - 1; i++)
            {
                var p1 = points.GetArrayElementAtIndex(i).vector2Value;
                var p2 = points.GetArrayElementAtIndex(i +
```

```
1).vector2Value;

                        float dist = HandleUtility.DistancePointToLineSegment(pos,
p1, p2);
                        if (dist < lineDist)
                        {
                                if (dist < minDist)
                                {
                                        minDist = dist;
                                        minDistIndex = i;
                                }
                        }
                }

                return minDistIndex;
        }

        const int RoundFactor = 16;

        public void RoundDrawing(SerializedProperty points)
        {
                for (int i = 0; i < points.arraySize; i++)
                {
                        var v = points.GetArrayElementAtIndex(i).vector2Value;
                        v.x = Mathf.Round(v.x * RoundFactor) / RoundFactor;
                        v.y = Mathf.Round(v.y * RoundFactor) / RoundFactor;
                        points.GetArrayElementAtIndex(i).vector2Value = v;
                }
        }
        public void NegYDrawing(SerializedProperty points)
```

```
{
        for (int i = 0; i < points.arraySize; i++)
        {
                var e = points.GetArrayElementAtIndex(i);
                var v = e.vector2Value;
                v.y = 1f - v.y;
                e.vector2Value = v;
        }
}
public void NegXDrawing(SerializedProperty points)
{
        for (int i = 0; i < points.arraySize; i++)
        {
                var e = points.GetArrayElementAtIndex(i);
                var v = e.vector2Value;
                v.x = 1f - v.x;
                e.vector2Value = v;
        }
}
public void NormalizeDrawing(SerializedProperty points)
{
        Vector2 min = points.GetArrayElementAtIndex(0).vector2Value;
        Vector2 max = min;
        for (int i = 1; i < points.arraySize; i++)
        {
                var v = points.GetArrayElementAtIndex(i).vector2Value;
                min.x = Mathf.Min(min.x, v.x);
                min.y = Mathf.Min(min.y, v.y);
                max.x = Mathf.Max(max.x, v.x);
                max.y = Mathf.Max(max.y, v.y);
```

```
                }

                Rect rect = Rect.MinMaxRect(min.x, min.y, max.x, max.y);

                float rectsize = Mathf.Max(rect.width, rect.height);
                rect = new Rect(rect.center - new Vector2(rectsize / 2, rectsize / 2),
new Vector2(rectsize, rectsize));

                for (int i = 0; i < points.arraySize; i++)
                {
                        var element = points.GetArrayElementAtIndex(i);
                        element.vector2Value = Rect.PointToNormalized(rect,
element.vector2Value);
                }
        }
        public void ReverseDrawing(SerializedProperty points)
        {
                for (int i = 0; i < points.arraySize / 2; i++)
                {
                        var a = points.GetArrayElementAtIndex(i).vector2Value;
                        var b = points.GetArrayElementAtIndex(points.arraySize - 1
- i).vector2Value;
                        points.GetArrayElementAtIndex(i).vector2Value = b;
                        points.GetArrayElementAtIndex(points.arraySize - 1 -
i).vector2Value = a;
                }
        }
        public void ChangeOrder(int gestureIndex, int orderChange)
        {
                figures.MoveArrayElement(gestureIndex, gestureIndex +
```

```
orderChange);
        }

        Vector2 NegY1(Vector2 v)
        {
            return new Vector2(v.x, 1f - v.y);
        }

        int currentGestureIndex = 0;

        void DrawGrid(Rect rect, int divs)
        {
            for (int i = 0; i <= divs; i++)
            {
                float t = Mathf.InverseLerp(0, divs, i);
                float ty = Mathf.Lerp(rect.yMin, rect.yMax, t);
                float tx = Mathf.Lerp(rect.xMin, rect.xMax, t);
                Handles.DrawLine(new Vector3(rect.xMin, ty), new
Vector3(rect.xMax, ty));
                Handles.DrawLine(new Vector3(tx, rect.yMin), new
Vector3(tx, rect.yMax));
            }
        }

        public override void OnInspectorGUI()
        {

            {
                var current = target as FiguresEditor;
                if (current != null && current.figurePoints != null &&
```

```
current.figurePoints.Count > 0 && current.figure != null && current.figure.figureLines.Count == 0)
                {
                    if (GUILayout.Button("Convert to new GesturePattern", GUILayout.ExpandWidth(true), GUILayout.Height(100)))
                    {

                        this.serializedObject.Update();

                        figures.arraySize = 1;

                        var points = figures.GetArrayElementAtIndex(0).FindPropertyRelative("figurePoints");
                        points.arraySize = current.figurePoints.Count;

                        for (int i = 0; i < current.figurePoints.Count; i++)
                        {

    points.GetArrayElementAtIndex(i).vector2Value = current.figurePoints[i];
                        }


    serializedObject.FindProperty("figurePoints").arraySize = 0;

                        serializedObject.ApplyModifiedProperties();
                    }
                    return;
                }
            }
```

```
                DrawDefaultInspector();

                this.serializedObject.Update();

                float w = EditorGUIUtility.currentViewWidth * 0.75f;
                w = 201;

                if (figures.arraySize > 0)
                {
                        currentGestureIndex = Mathf.Clamp(currentGestureIndex, 0,
figures.arraySize - 1);
                }
                else
                {
                        currentGestureIndex = 0;
                }


                EditorGUILayout.Space();


                if (figures.arraySize > 0)
                {

                        currentGestureIndex = Mathf.Clamp(currentGestureIndex, 0,
figures.arraySize - 1);

                        for (int gestureIndex = 0; gestureIndex < figures.arraySize;
gestureIndex++)
                        {
```

```
                    var points = figures.GetArrayElementAtIndex(gestureIndex).FindPropertyRelative("figurePoints");

                    bool closedLine = figures.GetArrayElementAtIndex(gestureIndex).FindPropertyRelative("closedDrawnLine").boolValue;
                    int closedLineInt = closedLine ? 1 : 0;

                    for (int i = 0; i < points.arraySize - 1 + closedLineInt; i++)
                    {
                        var v1 = points.GetArrayElementAtIndex(i).vector2Value;
                        var v2 = points.GetArrayElementAtIndex((i + 1) % points.arraySize).vector2Value;
                        v1 = Rect.NormalizedToPoint(rect, NegY1(v1));
                        v2 = Rect.NormalizedToPoint(rect, NegY1(v2));

                        if (gestureIndex == currentGestureIndex)
                        {
                            if (i + 1 == points.arraySize)
                            {
                                Handles.color = new Color(0, 1, 1, 0.2f);
                            }
                            else
```

```
                {
                    Handles.color = Color.cyan;
                }
                Handles.DrawLine(v1, v2);
                HandlesDrawLineStrong(v1, v2);
            }
            else
            {
                Handles.color = Color.cyan;
                Handles.DrawLine(v1, v2);
            }
        }

        if (useOrder.boolValue)
        {
            var v = points.GetArrayElementAtIndex(0).vector2Value;
            v = Rect.NormalizedToPoint(rect, NegY1(v));
            v.x = Mathf.Clamp(v.x, rect.xMin, rect.xMax - 10);
            v.y = Mathf.Clamp(v.y, rect.yMin, rect.yMax - 15);
            Handles.Label(v + new Vector2(-1, -1), (gestureIndex + 1).ToString(), new GUIStyle() { normal = new GUIStyleState() { textColor = Color.gray } });
            Handles.Label(v + new Vector2(-1, 1), (gestureIndex + 1).ToString(), new GUIStyle() { normal = new GUIStyleState() { textColor = Color.gray } });
            Handles.Label(v + new Vector2(1, -1), (gestureIndex + 1).ToString(), new GUIStyle() { normal = new GUIStyleState()
```

```
{ textColor = Color.gray } });
                                    Handles.Label(v + new Vector2(1, 1),
(gestureIndex + 1).ToString(), new GUIStyle() { normal = new GUIStyleState()
{ textColor = Color.gray } });
                                    Handles.Label(v, (gestureIndex + 1).ToString(),
new GUIStyle() { normal = new GUIStyleState() { textColor = Color.white } });
                                }
                                if (useDirection.boolValue)
                                {
                                    if (points.arraySize >= 2)
                                    {
                                        var v1 =
points.GetArrayElementAtIndex(points.arraySize - 2).vector2Value;
                                        var v2 =
points.GetArrayElementAtIndex(points.arraySize - 1).vector2Value;
                                        v1 = Rect.NormalizedToPoint(rect,
NegY1(v1));
                                        v2 = Rect.NormalizedToPoint(rect,
NegY1(v2));
                                        var dir = (v2 - v1).normalized;
                                        var va = v2 +
(Vector2)(Quaternion.Euler(0, 0, 90 + 45) * dir) * 10;
                                        var vb = v2 +
(Vector2)(Quaternion.Euler(0, 0, -90 - 45) * dir) * 10;
                                        Handles.DrawLine(v2, va);
                                        Handles.DrawLine(v2, vb);
                                        if (gestureIndex == currentGestureIndex)
                                        {
                                            HandlesDrawLineStrong(v2, va);
                                            HandlesDrawLineStrong(v2, vb);
```

```
                }
            }
        }

        if (gestureIndex == currentGestureIndex)
        {

            float border = 5;
            var borderRect = Rect.MinMaxRect(rect.xMin -
border, rect.yMin - border, rect.xMax + border, rect.yMax + border);

            var mousePos = Event.current.mousePosition;
            var localMousePos =
NegY1(Rect.PointToNormalized(rect, mousePos));

            bool insideBorderRect =
borderRect.Contains(Event.current.mousePosition);

            if (snap && insideBorderRect)
            {
                localMousePos.x =
Mathf.Round(localMousePos.x * RoundFactor) / RoundFactor;
                localMousePos.y =
Mathf.Round(localMousePos.y * RoundFactor) / RoundFactor;
                mousePos =
Rect.NormalizedToPoint(rect, NegY1(localMousePos));
            }

            bool onePoint = points.arraySize >= 1;
            bool twoPoints = points.arraySize >= 2;
```

```
                var nearIndex = onePoint ? FindNear(points, localMousePos) : -1;
                var nearPos = onePoint ? points.GetArrayElementAtIndex(nearIndex).vector2Value : Vector2.zero;
                float nearDist = onePoint ? Vector2.Distance(nearPos, localMousePos) : 0;

                float nearDistLimit = 0.05f;
                bool isNearPoint = onePoint && nearDist < nearDistLimit;

                int aboveLineIndex = -1;
                if (insideBorderRect && points.arraySize >= 2)
                {
                    aboveLineIndex = CalcAboveLineIndex(points, localMousePos, 0.02f);
                }
                bool isAboveLine = aboveLineIndex >= 0;

                if (Event.current.type == EventType.Repaint)
                {
                    if (insideBorderRect)
                    {
                        if (isNearPoint)
                        {
                            var globalNearPos = Rect.NormalizedToPoint(rect, NegY1(nearPos));
                            if (Event.current.control)
                            {
```

```
                    Handles.color = Color.red;
                }
                else
                {
                    Handles.color = Color.cyan;
                }

        Handles.DrawSolidDisc(globalNearPos, Vector3.forward, 5f);
            }
            else if (isAboveLine)
            {
                Handles.color = Color.cyan;

                var p1 = points.GetArrayElementAtIndex(aboveLineIndex).vector2Value;
                var p2 = points.GetArrayElementAtIndex(aboveLineIndex + 1).vector2Value;
                var gp1 = Rect.NormalizedToPoint(rect, NegY1(p1));
                var gp2 = Rect.NormalizedToPoint(rect, NegY1(p2));
                Vector2 dir = Quaternion.Euler(0, 0, 90) * (gp2 - gp1).normalized;

        HandlesDrawLineStrong(mousePos + dir * 10, mousePos - dir * 10);

            }
```

```
                else
                {
                    Handles.color = Color.cyan;
                    if (points.arraySize > 0)
                    {
                        DrawDots(mousePos,
Rect.NormalizedToPoint(rect,
NegY1(points.GetArrayElementAtIndex(points.arraySize - 1).vector2Value)), 10);
                        if (closedLine)
                        {

    DrawDots(mousePos,                Rect.NormalizedToPoint(rect,
NegY1(points.GetArrayElementAtIndex(0).vector2Value)), 10);
                        }
                    }

    Handles.DrawSolidDisc(mousePos, Vector3.forward, 5f);
                }
            }
            Repaint();
        }

        if (Event.current.type ==
EventType.MouseDown && Event.current.button == 0 && insideBorderRect)
        {
            if (isNearPoint)
            {
                if (Event.current.control)
                {
                    if (nearIndex >= 0)
```

```
                                        {
                                            if (points.arraySize ==
2)
                                            {
                                                points.arraySize
= 0;
                                                Repaint();
                                            }
                                            else
                                            {

        points.DeleteArrayElementAtIndex(nearIndex);

        currentGestureIndex = Mathf.Clamp(currentGestureIndex, 0, figures.arraySize -
1);
                                                Repaint();
                                            }
                                        }
                                    }
                                    else
                                    {
                                        dragIndex = nearIndex;

        points.GetArrayElementAtIndex(dragIndex).vector2Value = localMousePos;
                                    }
                                }
                                else if (isAboveLine)
                                {

        points.InsertArrayElementAtIndex(aboveLineIndex + 1);
```

```
        points.GetArrayElementAtIndex(aboveLineIndex + 1).vector2Value = localMousePos;
                                        dragIndex = aboveLineIndex + 1;
                                    }
                                    else
                                    {

        points.InsertArrayElementAtIndex(points.arraySize);

        points.GetArrayElementAtIndex(points.arraySize - 1).vector2Value = localMousePos;
                                        dragIndex = points.arraySize - 1;
                                    }
                                }

                                if (Event.current.type == EventType.MouseDrag && Event.current.button == 0 && insideBorderRect)
                                {

        points.GetArrayElementAtIndex(dragIndex).vector2Value = localMousePos;
                                }

                                if (Event.current.type == EventType.MouseUp && Event.current.button == 0)
                                {
                                    dragIndex = -1;
                                }

                                snap = EditorGUILayout.Toggle("Snap", snap);
```

```
                GUILayout.Label("Hold [Ctrl] to delete.");


GUILayout.BeginHorizontal(GUILayout.Width(w));
                if (GUILayout.Button("Clear"))
                {
                        points.ClearArray();
                }
                if (GUILayout.Button("Snap All"))
                {
                        RoundDrawing(points);
                }
                GUILayout.EndHorizontal();

GUILayout.BeginHorizontal(GUILayout.Width(w));
                if (GUILayout.Button("Expand"))
                {
                        NormalizeDrawing(points);
                }
                if (GUILayout.Button("Flip Y"))
                {
                        NegYDrawing(points);
                }
                if (GUILayout.Button("Flip X"))
                {
                        NegXDrawing(points);
                }
                GUILayout.EndHorizontal();
```

```
        GUILayout.BeginHorizontal(GUILayout.Width(w));
                                        if (GUILayout.Button("Reverse Direction"))
                                        {
                                                ReverseDrawing(points);
                                        }
                                        GUILayout.EndHorizontal();

        GUILayout.BeginHorizontal(GUILayout.Width(w));
                                        GUI.enabled = gestureIndex > 0;
                                        if (GUILayout.Button("-"))
                                        {
                                                ChangeOrder(gestureIndex, -1);
                                                currentGestureIndex = gestureIndex - 1;
                                        }
                                        GUI.enabled = true;
                                        GUILayout.Label("Order", new GUIStyle()
{ alignment = TextAnchor.MiddleCenter });
                                        GUI.enabled = gestureIndex < figures.arraySize
- 1;
                                        if (GUILayout.Button("+"))
                                        {
                                                ChangeOrder(gestureIndex, 1);
                                                currentGestureIndex = gestureIndex + 1;
                                        }
                                        GUI.enabled = true;
                                        GUILayout.EndHorizontal();

                                }
```

```
                    }

                }

                    serializedObject.ApplyModifiedProperties();

            }

        }
}
```

Скрипт «FiguresRecognizer»

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

namespace DrawnFiguresRecognizer
{
   public struct RegonizeScore
   {
      public float lineDistancePosition;
      public float lineDistanceAngle;

      public void InitiateMax()
      {
         lineDistancePosition = lineDistanceCurvature = lineDistanceAngle =
float.MaxValue;
      }
```

```
    public static RegonizeScore MaxLineDistance
    {
        get
        {
            var result = new RegonizeScore();
            result.InitiateMax();
            return result;
        }
    }

    public static bool operator >(RegonizeScore score1, RegonizeScore score2)
    {
        return score1.recognizeScore > score2.recognizeScore;
    }
    public static bool operator <(RegonizeScore score1, RegonizeScore score2)
    {
        return score1.recognizeScore < score2.recognizeScore;
    }
    public static bool operator >=(RegonizeScore score1, RegonizeScore score2)
    {
        return score1.recognizeScore >= score2.recognizeScore;
    }
    public static bool operator <=(RegonizeScore score1, RegonizeScore score2)
    {
        return score1.recognizeScore <= score2.recognizeScore;
    }
}

public class RecognizeResult
```

```
    {
        public FiguresEditor figure;
        public RegonizeScore score;
        public float recognizeTime;
        private static RecognizeResult emptyRezult = new RecognizeResult();
        public static RecognizeResult EmptyRezult { get { return emptyRezult; } }
    }

    public class FiguresRecognizer : MonoBehaviour
    {

        private const int FigureDetail = 100;

        [Range(1, 4)]
        public int countOfThreads = 1;

        public List<FiguresEditor> figurePatterns;

        public RecognizeResult FigureRecognize(FiguresData data, bool
normalizeScaleFlag = true)
        {

            var drawTimer = new System.Diagnostics.Stopwatch();

            drawTimer.Start();

            var normaizeData = NormalizeData(data, normalizeScaleFlag);

            var findFigure = findFigurePattern(normaizeData, normalizeScaleFlag);
```

```
            drawTimer.Stop();

                        findFigure.recognizeTime                                    =
(float)(drawTimer.ElapsedMilliseconds / 1000.0);

            return findFigure;
        }

        private FiguresData NormalizeData(FiguresData data, bool normalizeScaleFlag)
        {
            if (normalizeScaleFlag)
            {
                return
FigureNormalizeDistribution(NormFigureScale(NormFiguresClosedLines(data)),
FigureDetail);
            }
            else
            {
                return          FigureNormalizeDistribution(NormFiguresClosedLines(data),
FigureDetail);
            }
        }

        private static List<List<int>> GenerateFigurePermutations(List<int> figureList,
int lowVal = 0)
        {

            System.Action<int, int> swapValues = (int a, int b) =>
            {
                var temp = figureList[a];
```

```
        figureList[a] = figureList[b];
        figureList[b] = temp;
    };

    var figureResult = new List<List<int>>();

    if (lowVal + 1 >= figureList.Count)
    {
        figureResult.Add(new List<int>(figureList));
    }
    else
    {
        foreach (var p in GenerateFigurePermutations(figureList, lowVal + 1))
        {
            figureResult.Add(new List<int>(p));
        }
        for (int i = lowVal + 1; i < figureList.Count; i++)
        {
            swapValues(lowVal, i);
            foreach (var p in GenerateFigurePermutations(figureList, lowVal + 1))
            {
                figureResult.Add(new List<int>(p));
            }
            swapValues(lowVal, i);
        }
    }
    return figureResult;
}

private RecognizeResult findFigurePattern(FiguresData queryFiguresData, bool
```

```
normalizeScaleFlag)
    {
        var bestFigure = default(FiguresEditor);
        var bestScoreVal = RegonizeScore.MaxLineDistance;

        var indexesList = Enumerable.Range(0, queryFiguresData.figureLines.Count).ToList();
        List<List<int>> permutationFiguresIndexes = GenerateFigurePermutations(indexesList);

        var permutations = permutationFiguresIndexes.Select(e => makeFigurePermutation(e, queryFiguresData)).ToList();
        var singleFigurePermutation = permutations.GetRange(0, 1);

        int n_threadsNumbers = Mathf.Min(this.countOfThreads, figurePatterns.Count);

        var threadsList = new List<Thread>();

        for (int index = 0; index < n_threadsNumbers; index++)
        {
            int beginThreadIndex = index * figurePatterns.Count / n_threadsNumbers;
            int endThreadIndex = (index + 1) * figurePatterns.Count / n_threadsNumbers - 1;

            threadsList.Add(new Thread(() =>
            {
                var figureResult = SearchFiguresPatterns(beginThreadIndex, endThreadIndex, queryFiguresData, normalizeScaleFlag, permutations, singleFigurePermutation);
```

```
            lock (this)
            {
                if (figureResult.score > bestScoreVal)
                {
                    bestScoreVal = figureResult.score;
                    bestFigure = figureResult.figure;
                }
            }
        }));
    }

    for (int i = 0; i < threadsList.Count; i++)
        threadsList[i].Start();

    for (int i = 0; i < threadsList.Count; i++)
        threadsList[i].Join();

    return new RecognizeResult() { figure = bestFigure, score = bestScoreVal };
}
```

```
RecognizeResult SearchFiguresPatterns(int beginFigureIndex, int endFigureIndex, FiguresData queryFigureData, bool normalizeFigureScale, List<FiguresData> figurePermutations, List<FiguresData> singleFigurePermutation)
{
    var bestFigure = default(FiguresEditor);
    var bestFigureScore = RegonizeScore.MaxLineDistance;

    for (int i = beginFigureIndex; i <= endFigureIndex; i++)
    {
```

```
            var figureAsset = figurePatterns[i];
            var assetFigureData = NormalizeData(figureAsset.figure, normalizeFigureScale);

            if (assetFigureData.figureLines.Count != queryFigureData.figureLines.Count)
            {
               continue;
            }

            var figurePermutatLook = figureAsset.useDrawnLinesOrder ? singleFigurePermutation : figurePermutations;

            foreach (var dataVal in figurePermutatLook)
            {

               var permutationfigureScore = CalculateFigureScore(dataVal, assetFigureData, figureAsset.useDrawnLinesDirections);

               float positionDistance = permutationfigureScore.lineDistancePosition;
               float curvatureDistance = permutationfigureScore.lineDistanceCurvature;
               float angleDistance = permutationfigureScore.lineDistanceAngle;

               if (permutationfigureScore > bestFigureScore)
               {
                  bestFigureScore = permutationfigureScore;
                  bestFigure = figureAsset;
               }
            }
         }
```

```
        return new RecognizeResult() { figure = bestFigure, score = bestFigureScore };
    }


    private RegonizeScore CalculateFigureScore(FiguresData figuresData1,
FiguresData figuresData2, bool useLinesDirectionsFlag)
    {

        if (figuresData1.figureLines.Count == figuresData2.figureLines.Count)
        {

            var figureLineScores = new List<RegonizeScore>();

            for (int i = 0; i < figuresData1.figureLines.Count; i++) {
                var figureLine1 = figuresData1.figureLines[i].figurePoints;
                var figureLineToForward = figuresData2.figureLines[i].figurePoints;
                var figureScoreForward = CalculateFigureListScore(figureLine1,
figureLineToForward, figuresData2.figureLines[i].closedDrawnLine);
                if (useLinesDirectionsFlag) {
                    figureLineScores.Add(figureScoreForward);
                }
                else {
                    var figureLineToBackwards =
figureLineToForward.AsEnumerable().Reverse().ToList();
                    var figureScoreBackwards = CalculateFigureListScore(figureLine1,
figureLineToBackwards, figuresData2.figureLines[i].closedDrawnLine);
                    var figureScore = figureScoreForward > figureScoreBackwards ?
figureScoreForward : figureScoreBackwards;
                    figureLineScores.Add(figureScore);
```

```
            }
        }

        return new RegonizeScore()
        {
            lineDistancePosition = figureLineScores.Select(e => e.lineDistancePosition).Sum() / figureLineScores.Count,
            lineDistanceAngle = figureLineScores.Select(e => e.lineDistanceAngle).Sum() / figureLineScores.Count,
            lineDistanceCurvature = figureLineScores.Select(e => e.lineDistanceCurvature).Sum() / figureLineScores.Count
        };

    }
    else
    {
        return RegonizeScore.MaxLineDistance;
    }
}

private List<float> CalculateFigureCurvature(List<Vector2> figurePoints)
{
    int checkStep = 10;
    List<float> result = new List<float>();
    for (int i = 0; i < checkStep; i++)
        result.Add(0);
    for (int i = checkStep; i < figurePoints.Count - checkStep; i++)
    {
        var points1 = figurePoints[i - checkStep];
        var points2 = figurePoints[i];
```

```
            var points3 = figurePoints[i + checkStep];
            var lineData1 = points2 - points1;
            var lineData2 = points3 - points2;
            var lineAngle1 = Mathf.Atan2(lineData1.y, lineData1.x) * Mathf.Rad2Deg;
            var lineAngle2 = Mathf.Atan2(lineData2.y, lineData2.x) * Mathf.Rad2Deg;
            var deltaAngle = Mathf.DeltaAngle(lineAngle1, lineAngle2);
            result.Add(deltaAngle);
        }
        for (int i = 0; i < checkStep; i++)
            result.Add(0);
        return result;
    }

    private RegonizeScore CalculateCircularFiguresListScore(List<Vector2>
linePoints1, List<Vector2> linePoints2)
    {
        List<Vector2> figurePointsToOffset = new List<Vector2>();

        RegonizeScore figureMaxScore = new RegonizeScore();

        for (int j = 0; j < linePoints2.Count; j++)
        {
            figurePointsToOffset.Clear();

            for (int i = 0; i <= linePoints2.Count; i++)
            {
                figurePointsToOffset.Add(linePoints2[(i + j) % linePoints2.Count]);
            }

            RegonizeScore figureScore = CalculateLinearFiguresListScore(linePoints1,
```

```
figurePointsToOffset);

            if (figureScore > figureMaxScore || j == 0)
            {
                figureMaxScore = figureScore;
            }
        }

        return figureMaxScore;
    }

    private float CalculateLinePositionDistance(List<Vector2> linePoints1,
List<Vector2> linePoints2)
    {
        float square = Mathf.Sqrt(2);

        float sumLinesDistance = 0;

        int n = linePoints1.Count;
        for (int i = 0; i < n; i++)
        {
            float resultDistance = Vector2.Distance(linePoints1[i], linePoints2[i]) /
square;
            sumLinesDistance += resultDistance;
        }

        return sumLinesDistance;
    }

    private float CalculateLineAngleDistance(List<Vector2> linePoints1,
```

```
List<Vector2> linePoints2)
    {

        int count = linePoints1.Count;

        var lineAngles1 = CalculateLineAngles(linePoints1);
        var lineAngles2 = CalculateLineAngles(linePoints2);

        float totalAngleDistance = 0;

        for (int i = 0; i < count; i++)
        {
            float resultAngle = Mathf.Abs(Mathf.DeltaAngle(lineAngles1[i],
lineAngles2[i])) / 360f;
            totalAngleDistance += resultAngle;
        }

        return totalAngleDistance;
    }

    private Rect CalculateLineRect(FiguresData figuresData)
    {
        float minPointx, minPointy, maxPointx, maxPointy;
        minPointx = maxPointx = figuresData.figureLines[0].figurePoints[0].x;
        minPointy = maxPointy = figuresData.figureLines[0].figurePoints[0].y;

        for (int j = 0; j < figuresData.figureLines.Count; j++)
        {
            var linePoints = figuresData.figureLines[j].figurePoints;
```

```
            for (int i = 0; i < linePoints.Count; i++)
            {

                var points = linePoints[i];
                minPointx = Mathf.Min(minPointx, points.x);
                maxPointx = Mathf.Max(maxPointx, points.x);
                minPointy = Mathf.Min(minPointy, points.y);
                maxPointy = Mathf.Max(maxPointy, points.y);
            }
        }
        Rect rect = Rect.MinMaxRect(minPointx, minPointy, maxPointx, maxPointy);
        float resultRectSize = Mathf.Max(rect.width, rect.height);
        rect = new Rect(rect.center - new Vector2(resultRectSize / 2, resultRectSize / 2), new Vector2(resultRectSize, resultRectSize));
        return rect;
    }

    private Vector2 FindFigureByNorm(List<Vector2> vectors, List<float> data, float num)
    {
        for (int i = 0; i < data.Count - 1; i++)
        {
            var data1 = data[i];
            var data2 = data[i + 1];
            if (data1 <= num && num <= data2)
            {
                var vector1 = vectors[i];
                var vector2 = vectors[i + 1];
                float total = Mathf.InverseLerp(data1, data2, num);
                return Vector2.Lerp(vector1, vector2, total);
```

```
            }
        }
        return num > 0.5f ? vectors[vectors.Count - 1] : vectors[0];
    }

    private FiguresData FigureNormalizeDistribution(FiguresData figuresData, int count)
    {
        var resultFigure = new FiguresData();
        foreach (var line in figuresData.figureLines)
        {
            resultFigure.figureLines.Add(new FiguresLines()
            {
                figurePoints = FigureNormalizeDistribution(line.figurePoints, count),
                closedDrawnLine = line.closedDrawnLine
            });
        }
        return resultFigure;
    }

        List<float> currPosition = new List<float>();

        currPosition.Add(0);
        for (int i = 1; i < direction.Count; i++)
        {
            var vector1 = direction[i - 1];
            var vector2 = direction[i];
            currPosition.Add(currPosition[i - 1] + Vector2.Distance(vector1, vector2));
        }
```

```
            float totalDist = currPosition.Last();

            var normalizedPosition = currPosition.Select(e => e / totalDist).ToList();

            var resultTotal = new List<Vector2>();

            for (int i = 0; i <= count; i++)
            {
                float calculatedResult = (float)i / count;
                resultTotal.Add(FindFigureByNorm(direction, normalizedPosition,
calculatedResult));
            }

            return resultTotal;
        }
    }
}
```

Скрипт «GameMode»

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

public class GameMode : MonoBehaviour
{
    [SerializeField] public GameObject first;
```

```
    [SerializeField] public GameObject second;

    [SerializeField] public GameObject third;

    [SerializeField] public GameObject fourth;

    [SerializeField] public GameObject fifth;

    [SerializeField] public GameObject sixth;

    public void loadDescr(int choise)
    {
        GameObject[] objects = {first, second, third, fourth, fifth, sixth};

        first.SetActive(false);
        second.SetActive(false);
        third.SetActive(false);
        fourth.SetActive(false);
        fifth.SetActive(false);
        sixth.SetActive(false);

        objects[choise].SetActive(true);
    }
}
```

Скрипт «TaskDraw»

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

```
using UnityEditor;
using UnityEditorInternal;
using UnityEngine.UIElements;

[CustomEditor(typeof(Tasks))]
[CanEditMultipleObjects]
[System.Serializable]
public class TaskDraw : Editor
{
    private Tasks TaskInstance => target as Tasks;
    private ReorderableList TaskList;

    private void OnEnable()
    {
        InitializeReordableList(ref TaskList, "taskList", "Task List");
    }

    public override void OnInspectorGUI()
    {
        serializedObject.Update();
        TaskList.DoLayoutList();
        serializedObject.ApplyModifiedProperties();
    }


    private void InitializeReordableList(ref ReorderableList list, string propertyName, string listLable)
    {
        list = new ReorderableList(serializedObject, serializedObject.FindProperty(propertyName), true, true, true, true);
```

```
        list.onAddCallback = reorderableList => TaskInstance.AddTask();

        list.drawHeaderCallback = (Rect rect) =>
        {
            EditorGUI.LabelField(rect, listLable);
        };

        var l = list;
        list.drawElementCallback = (Rect rect, int index, bool isActive, bool isFocused)
=>
        {
            var element = l.serializedProperty.GetArrayElementAtIndex(index);
            rect.y += 2;
            EditorGUI.PropertyField(
                new Rect(rect.x, rect.y, 300, EditorGUIUtility.singleLineHeight),
                element.FindPropertyRelative("task1"), GUIContent.none);

            EditorGUI.PropertyField(
                new Rect(rect.x + 310, rect.y, 300, EditorGUIUtility.singleLineHeight),
                element.FindPropertyRelative("task2"), GUIContent.none);

            EditorGUI.PropertyField(
                new Rect(rect.x + 620, rect.y, 300, EditorGUIUtility.singleLineHeight),
                element.FindPropertyRelative("answer"), GUIContent.none);

            EditorGUI.PropertyField(
                new Rect(rect.x + 930, rect.y, 300, EditorGUIUtility.singleLineHeight),
                element.FindPropertyRelative("isQuestioned"), GUIContent.none);
        };
    }
```

```
}
```

Фрагмент скрипту «Result»

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


namespace DrawnFiguresRecognizer
{

    public class Result : MonoBehaviour
    {
        public Tasks tasks;

        [SerializeField] public Timer _timer;

        private bool canAsk = false;

        [SerializeField] public Text _questionText;

        public FiguresDetector _drawDetector;

        public bool GetCanAsk()
        {
            return this.canAsk;
```

```
}

void Start()
{

    AskQuestion();
    ClearQuestions();
}

public void ReadyScreen(string answer)
{

}

public void ShowResults(string answer)
{
    this.canAsk = false;
    if (tasks.taskList[tasks.currentTask].answer == answer)
    {
        _drawDetector.ClearDrawnLines();
        StartCoroutine(ShowResults(true));
    } else
    {
        Debug.Log("ANSWER: " + answer);
        StartCoroutine(ShowResults(false));
    }
}

private IEnumerator ShowResults(bool correct)
{
```

```
            if (correct)
            {
                _questionText.text = _questionText.text.Replace("?",
tasks.taskList[tasks.currentTask].answer);
                tasks.taskList[tasks.currentTask].isQuestioned = true;
                yield return new WaitForSeconds(1.0f);
                AskQuestion();
            }
            else
            {
                //_questionText.text = " fail ";
                _drawDetector.ClearDrawnLines();
                this.canAsk = true;
            }
        }

        public void AskQuestion()
        {
            if (CountValidQuestions() == 0)
            {
                _questionText.text = string.Empty;
                ClearQuestions();
                SceneManager.LoadScene("MainMenu");
                return;
            }
            var randomIndex = 0;

            do
            {
                randomIndex = UnityEngine.Random.Range(0, tasks.taskList.Count);
```

```
            } while (tasks.taskList[randomIndex].isQuestioned == true);
           // Debug.Log("VALUE: " + randomIndex);
            tasks.currentTask = randomIndex;
           _questionText.text = tasks.taskList[randomIndex].task1;
            tasks.taskList[tasks.currentTask] = tasks.taskList[randomIndex];
            this.canAsk = true;
        }

        public void ClearQuestions()
        {
            foreach (var question in tasks.taskList)
            {
                question.isQuestioned = false;
            }
        }

        private int CountValidQuestions()
        {
            int validQuestion = 0;
            foreach (var question in tasks.taskList)
            {
                if (question.isQuestioned == false)
                    validQuestion++;
            }
            return validQuestion;
        }
    }
}
```