МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп’ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПІДТРИМКИ ЕДЬЮТЕЙНМЕНТ-ПРОЕКТУ ДЛЯ НАВЧАННЯ ПРОГРАМУВАННЮ

Здобувач освіти гр. ІК.м – 11 Андрій ЛІЗУНОВ

Науковий керівник, старший викладач кафедри комп’ютерних наук, кандидат фізико-математичних наук Галина ОЛЕКСІЄНКО

В. о. завідувача кафедри комп’ютерних наук, кандидат технічних наук Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Лізунову Андрію Олеговичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія підтримки едьютейнмент-проекту для навчання програмуванню*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Огляд існуючих едьютейнмент-додатків з навчання програмуванню та алгоритмів програмування;
2) Формування вимог до проекту, постановка завдання й вибір програмного інструментарію;

3) Розробка едьютейнмент-додатку з навчання програмуванню
4) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
___

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Оглял додатків-аналогів та алгоритмів програмування		
2.	Постановка задачі, формування вимог до проекту		

3.	Вибір програмного інструментарію		
4.	Розробка додатку за допомогою ігрового двигуна «Unity»		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____

(підпис)

Керівник проекту _____

(підпис)

РЕФЕРАТ

Записка: 65 стор., 2 табл., 33 рис., 20 літературних джерел.

Тема роботи - «Інформаційна технологія підтримки едьютейнмент-проекту для навчання програмуванню».

Мета роботи – розробка та реалізація спеціального ігрового едьютейнмент-додатку для навчання студентів, програмістів та інших зацікавлених осіб алгоритмам програмування.

Об'єктом дослідження є потреба студентів у отриманні знань або їх покращенні у сфері алгоритмів програмування.

Предметом дослідження є процес розробки інформаційної технології едьютейнмент-проекту з навчання алгоритмів програмування.

Результати – проведений аналіз літератури, сучасний стан джерел отримання знань у сфері алгоритмів програмування та тенденції розвитку цих джерел. Також проаналізовано аналогічні едьютейнмент-додатки, їх характеристики, переваги та недоліки. Викладено основні положення технології, що розробляється, список вимог до інформаційної технології, що проектується, викладено постановку задачі роботи, а також здійснено вибір програмного інструментарію для розробки технології. Після проведеного аналізу та формування вимог до проекту було створено едьютейнмент-додаток для платформи «Android» з використанням ігрового двигуна «Unity» та мови програмування «C#». Дана розробка дозволяє користувачам отримувати знання у сфері алгоритмів програмування через ігровий досвід.

ЕДЬЮТЕЙНМЕНТ-ДОДАТОК, АЛГОРИТМИ ПРОГРАМУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, UNITY, НАВЧАННЯ.

ЗМІСТ

ВСТУП

Актуальність. На сьогоднішній день обізнаності у сфері алгоритмів програмування потребує кожен, хто працює з програмним кодом, адже такі алгоритми використовуються дуже часто. Наприклад, для того, щоб знайти користувача через меню пошуку у соціальній мережі, використовують спеціальні алгоритми, які зводять час пошуку до мінімуму. Існує багато способів освоєння таких алгоритмів, наприклад, через книжки, відео-курси, а також спеціальні едьютейнмент-додатки.

Актуальність дипломної роботи зумовлена бажанням підвищити рівень обізнаності студентів, програмістів та зацікавлених осіб у сфері алгоритмів програмування шляхом створення та популяризації ігрового едьютейнмент-додатку.

До списку задач проекту входить аналіз предметної області, порівняння додатку, що проектується, з його аналогами, розгляд технологій досягнення мети, обрання програмного інструментарію для створення едьютейнмент-додатку, а також його розробка.

Практичним значенням результатів даної роботи є задоволення проблеми нестачі знань студентів та усіх бажаючих у сфері алгоритмів програмування шляхом публікації ігрового едьютейнмент на одній з популярних платформ для розміщення додатків. Даний додаток дозволить користувачам навчатись алгоритмам програмування через ігровий досвід, що дозволить їм не тільки отримати знання, а ще трохи розважитись. Такий спосіб навчання дозволить користувачам краще запам’ятати інформацію за допомогою виникнення асоціацій алгоритмів програмування з різноманітними ігровими ситуаціями.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз сучасного стану предметної області

З появою мов програмування також поступово збільшувалася потреба у їх знанні, адже вони дозволяють суттєво скоротити час інженера на вирішення технологічних проблем. На сьогоднійшній день існує дуже багато ресурсів для отримання знань у сфері програмування. Такими ресурсами є книжки, Інтернет-статті, довідники, а також різноманітні освітні додатки. Якщо порівнювати ці ресурси, то серед них на даний момент найбільш зручними та ефективними є книжки та додатки, адже зазвичай вони містять усю необіхну інформацію для успішного освоєння конкретних тем, але ми зупинимось на додатках, адже з появою різноманітних гаджетів їх популярність зростає з кожним днем.

Незважаючи на переваги існуючих додатків, вони мають свої недоліки. До основних недоліків більшості таких додатків можна віднести відсутність в матеріалах для вивчення живих яскравих прикладів використання алгоритмів програмування. Якщо додати до програми інтерактивні яскраві приклади використання алгоритмів програмування, додаток вже стає ігровим. Така технологія розробки додатків дозволяє поєднати дві складові навчальну та розважальну. Для додатків такого типу існує спеціальна назва – «едьютейнмет» технологія (з англ. «education» - навчання, «entertainment» - розвага). Даний термін набув широкого розповсюдження у зарубіжній сфері освіти [1].

Едьютейнментом називають сукупність технік та дидактичних засобів навчання за допомогою розваги. Головною особливістю даної технології є принцип передавання знань у зрозумілій, доступній і цікавій формі. Використання такого принципу дозволяє досягти балансу між інформацією та мультимедійними продуктами.

1.2 Огляд алгоритмів програмування

Алгоритми програмування є наборами інструкцій мовою програмування для виконання певної задачі [1]. За роки існування мов програмування та комп'ютерної техніки була написана велика кількість алгоритмів, до того ж у наш час їх код можна легко знайти в Інтернеті або спеціальних книжках тощо. Такі готові алгоритми значно спрощують час інженера на виконання певної задачі, тому кожному, хто цікавиться темою програмування, буде корисно їх знати.

Серед найбільш популярних та застосовуваних алгоритмів виділяють алгоритми сортування, пошуку, алгоритми для графів, математичні, а також рекурсію. Розглянемо деякі з них та оберемо ті, що найбільш підходять нам, беручи до уваги їх актуальність та час на виконання даної роботи.

1.2.1 Сортування вибором

Алгоритм сортування вибором є алгоритмом для сортування даних у зростаючому чи зменшувальному порядку у залежності від певної особливості елемента даних. Такою особливістю може бути, наприклад, числовий код літери або саме число. За допомогою даного алгоритму можна відсортувати імена, номери телефонів, числа, записи в соціальній мережі (від нових до старих або навпаки) та будь-які об'єкти, якщо до них прив'язаний числовий код. Для того, щоб відсортувати список з елементів, даний алгоритм перевіряє кожний з кожним з елементів, тому час на виповнення цього алгоритму є більшим, ніж у деяких алгоритмів сортування. Тим не менше, даний алгоритм прекрасно справляється з невеликою кількістю елементів, а враховуючи обчислювальні можливості сучасної техніки він непогано працює у багатьох випадках, до того ж він є достатньо простим. Час його виконання становить $F(x^2)$, де F – деяка функція, x – число операцій отримання певного результату. У випадку з сортуванням чисел x – кількість чисел, що потрібно відсортувати.

1.2.2 Бінарний пошук

Розглянемо один з алгоритмів пошуку – бінарний пошук. Даний алгоритм використовують для пошуку елементів у відсортованому наборі даних. Для того, щоб пояснити, як він працює, наведемо один яскравий приклад. Нехай вам потрібно відгати число від 1 до 100 за нескінченне число спроб, при цьому вам би говорили, чи є ваше число меншим або більшим за таємне. Ви можете почати відгадувати по одному числу, починаючи з одиниці. Такий спосіб спрацює, якщо б таємний числом виявилось число 1 або, наприклад 2 тощо. Але якщо б таємним числом було, наприклад, число 70, то вам довелося б витратити 70 спроб, щоб вгадати його.

Алгоритм бінарного пошуку є алгоритмом, що прекрасно підходить для вирішення даної задачі. Його сенс полягає в тому, що кожного разу він намагається відгадати елемент в середині списку, відсікаючи половину елементів, що є більшими або меншими за кодом, ніж шуканий елемент. Саме тому даний алгоритм працює тільки з відсотрованими наборами даних. Для того, щоб вгадати число від 1 до 100 з використанням даного алгоритму, ви кожного разу б відкидали половину чисел, тому алгоритм бінарного пошуку є досить швидким. Час його виконання становить F(log x).

1.2.3 Рекурсія

Далі розглянемо такий алгоритм програмування, як рекурсія. Якщо говорити простими словами, то рекурсія – це функція, яка викликає сама себе. Рекурсію відносять до ітеративних варіантів функцій, тобто таких, що виконуються по декілька разів у залежності від певної умови. Кожна рекурсивна функція має базову та рекурсивну умови. У базовій умові функція сама себе не викликає, а в рекурсивній – навпаки.

Прикладом рекурсивної функції може бути функція визначення факторіалу числа. Для того, щоб знайти факторіал числа рекурсивна функція домножує число на виклик самої себе з числом меншим на одиницю, доки це цисло не дорівнює одиниці. Наприклад, щоб знайти факторіал числа 4,

рекурсивна функція в одній з умов (базовій або рекурсивній) перевірить, чи дорівнює число 4 одиниці. Якщо ні, то виконується рекурсивна частина, коду де функція домножує число 4 на виклик самої себе з числом, меншим на одиницю, тобто числом 3. Даний процес виконується до тих, поки число не буде дорівнювати одиниці, тобто виконається базова умова, коли результат множення попередніх числе з рекурсивного випадку домножить на число 1 і функція поверне результат.

Даний алгоритм є дуже корисним для невеликого для невеликої кількості рекусивнимх викликів, але є поганим для великого, адже кожного разу, коли функція викликає саму себе, її результат виконання зберігається в пам’яті. Таким чином, якщо фунція викликає саму себе велику кількість разів пам’ять (стек викликів) заповнюється, тому даний алгоритм слід використовувати з обережністю.

1.2.4 Алгоритм швидкого сортування

Алгоритм швидкого сортування, як видно з його назви, є одним з різновидів алгоритмів сотртування даних. Його головна перевага над алгоритмом сортування вибором полягає у швидкості виконання, котра становить $F(x \log x)$, що набагато швидше, ніж $F(x^2)$. Така швидкість виконання досягається за рахунок особливості роботи даного алгоритму.

Для того, щоб відсортувати набір даних, наприклад, чисел, цей алгоритм бере деякий опроний елемент – зазвичай перший у списку з даними – та розділяє список з числами на два підписки: елементи, менші опорного, та елементи, більші ніж опорний. Після Цього даний алгоритм рекурсивно здійснює швидке сортування над отриманими підсписками чисел, тобто повторює ті ж самі операції, доки два підписки після розбивання не будуть довжиною в один елемент.

Таким чином, даний алгоритм є досить швидким, але складним в розумінні.

1.2.5 Алгоритм Дейкстри, або алгоритм пошуку найкоротшого шляху

Алгоритм Дейкстри або алгоритм пошуку найкоротшого шляху одним з різновидів алгоритмів, що працюють на основі графів – фігур на площині, що складаються з множини кількох точок та ребер, котрі з’єднують ці точки. Цей алгоритм використовується для пошуку найкоротшого шляху від однієї точки до іншої.

Карту можливих шляхів від однієї до іншої точки представляють у вигляді графу, де ребрам графа призначається довжини (ціни), а точкам – назви.

Вигляд простого графу наведено на рис. 1.1. Точки на графі використовують як точки на місцевості, а ребра – шляхи між цими точками. На реальному прикладі ребрами графу можуть бути вулиці або автомобільні дороги тощо.

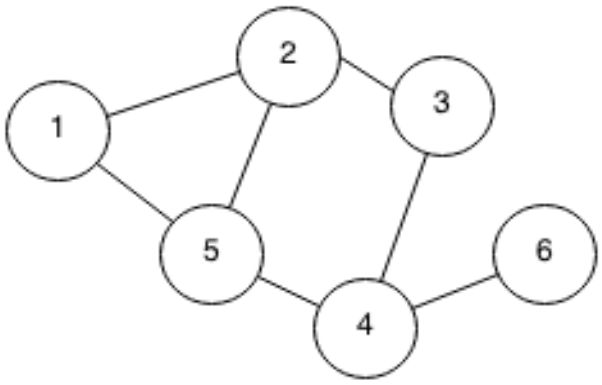

Рисунок 1.1 – Граф з 6 вершинами та 7 ребрами

Алгоритм пошуку найкоротшого шляху виконується наступним чином:

1. Спочатку знаходять вузол (точку) з найменшою ціною, тобто вузол, до якого можна дістатись за мінімальний час від початкового;

2. Оновлюються вартості сусідів цього вузла у випадку, якщо попердня відстань або час до вузла були більшими, ніж знайдена;

3. Повторюються пункти 1 та 2, доки операції не будуть виконані над всіма вузлами графа;

4. Знаходиться сумарний шлях від початкової точки до шуканої.

Даний алгоритм є дещо складним, але дуже корисним для вирішення задачі пошуку найкоротшого шляху.

1.3 Огляд аналогічних додатків

Перш ніж почати роботу над проєктом, необхіно провести порівняльних аналіз зі схожими проєктами для навчання алгоритмів програмування.

Так як доля користувачів мобільних пристроїв на базі «Android» є більшої, ніж для інших операційних систем (ОС), то доцільніше буде проводити розгляд додатків саме для цієї ОС. До найбільш популярних освітніх додатків в «Google Play Store» у сфері програмування можна віднести «Algorithms: Explained and Anim» (рис. 1.2), «Algorithms and Data Structures» (рис 1.3), «AlgoPrep – Algorithms & Data s» (рис. 1.4), «Data Structures & Algorithms» (рис. 1.5).

Для зручності сформуємо ряд критеріїв, котрі повинна задовольняти едьютейнмент-технологія та провоедемо порівняльний аналіз додатків-аналогів з нашим. До таких критеріїв віднесемо наступне: баланс між інформаційною та розважальною складовою, якість інформації, кількість інформації, кофмортність ігрового досвіду тощо. Критерії та назви додатків зведемо у таблицю 1.1.

Таблиця 1.1 – Порівняння схожих додатків з нашим за критеріями

Назва критерію / Назва додатку	Баланс між інформацією та розважальною складовою	Якість інформації	Кількість інформації	Комфортність користувацького досвіду
Інформаційна технологія підтримки едьютейнмент-проекту для навчання програмуванню	Однакова кількість ігрових медіа-даних та інформації	Висока	Середня	Висока
Algorithms: Explained and Anim	Присутня лише інформаційна складова	Висока	Велика	Висока
Algorithms and Data Structures	Присутня лише інформаційна складова	Висока	Велика	Висока

Таблиця 1.1 – Продовження

AlgoPrep – Algorithms & Data s	Пристуня лише інформаційна складова	Висока	Велика	Середня
Data Structures & Algorithms	Пристуня лише інформаційна складова	Середня	Середня	Низька

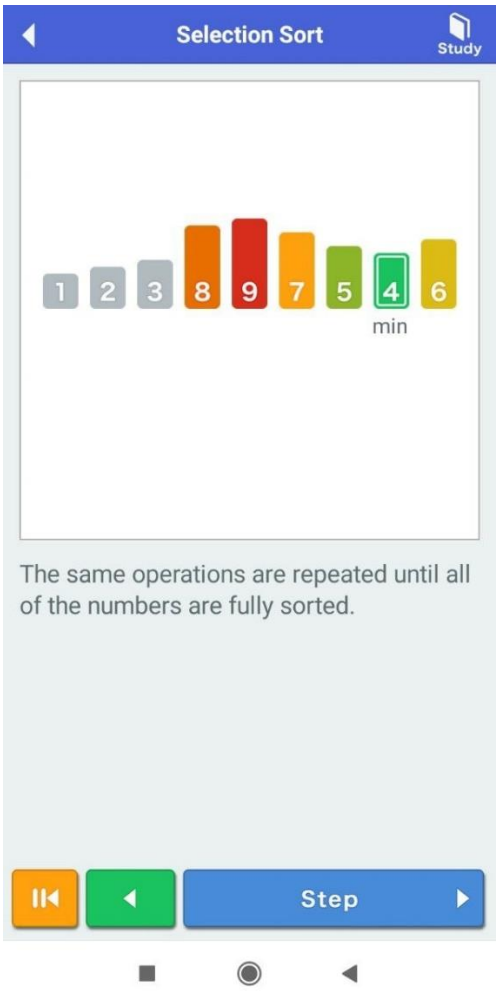

Рисунок 1.2 – Демонстрація додатку «Algorithms: Explained and Anim»

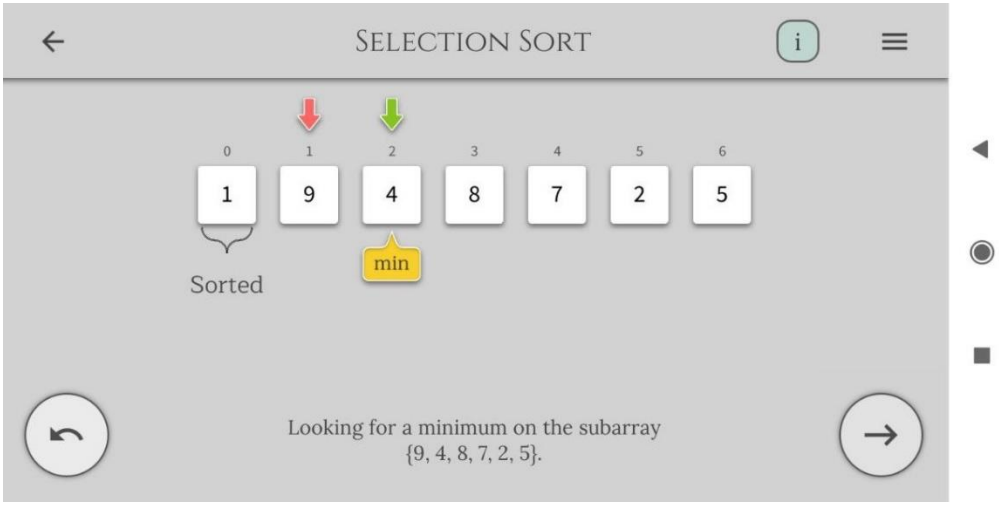

Рисунок 1.3 – Демонстрація додатку «Algorithms and Data Structures»

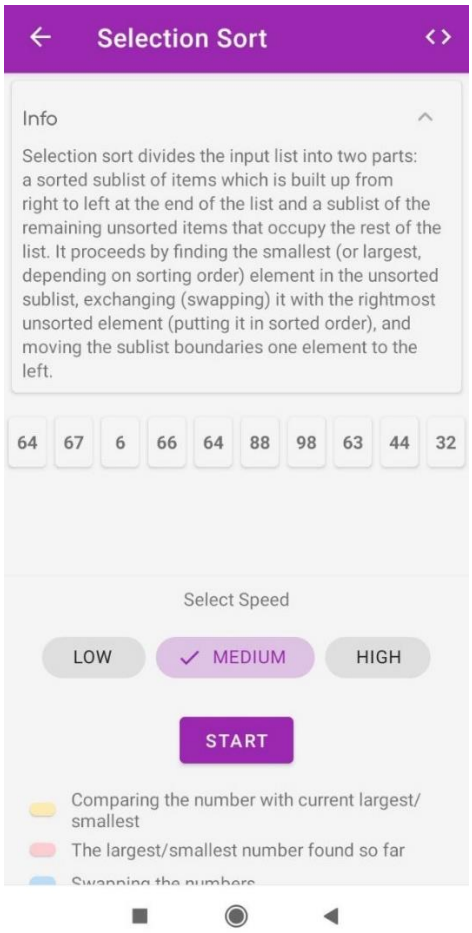

Рисунок 1.4 – Демонстрація додатку «AlgoPrep – Algorithms & Data s»

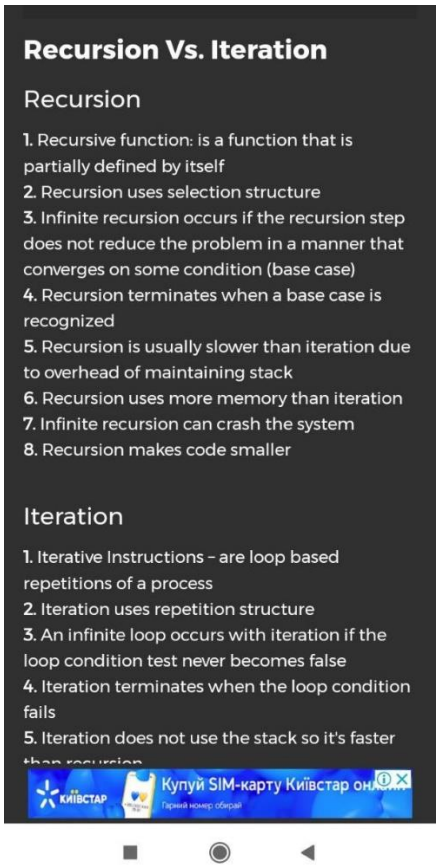

Рисунок 1.5 – Демонстрація додатку «Data Structures & Algorithms»

Усі перелічені в табл. 1.1 додатки містять у своєму складі лише інформаційну складову, в той час як схожих едьютейнмент-додатків для освоєння алгоритмів програмування немає. Перелічені у табл. 1.1 програми-аналоги містять у своєму складі лише інформаційну складову, що не задовольняє вимогам едьютейнмент додатку. Не беручи до уваги цей факт, можна сказати, що якість інформації в усіх додатків висока, окрім «Data Structures & Algorithms». Кількість інформації також є великою, окрім нашого додатку та «Data Structures & Algorithm» - у цих двох програм вона є середньою. Комфортність користувацького досвіду у всіх додатків є високою, окрім двох останніх. У «AlgoPrep – Algorithms & Data s» вона є середньою, адже в цій програмі при перегляді контенту немає кнопок перемотування інформації вперед або назад. Із-за цього доводиться чекати, доки анімації виконаються до кінця, хоча й користувач встиг прочитати сторінку з

матеріалом. «Data Structures & Algorithms» має низьку комфортність користувацьокого досвіду, адже дані у головному меню не достатньо структуровані задля інтуїтивного розуміння списку тем та порядку їх освоєння.

Серед додатків-аналогів найкращим приклаодом програми для навчання алгоритмам програмування є «Algorithms: Explained and Anim», але, на жаль, вона не має у своєму складі розважальної складової.

Після проведеного аналізу ми можемо зробити висновок, що кожна з програм-аналогів має свої особливості, але жодна з них не задовольняє критерій наявності розважальної складової для легшого сприйняття інформації та асоціативного запам’ятовування.

1.4 Постановка задачі

Розглянувши існуютчі алгоритми програмування, а також проаналізувавши додатки-аналого, можем сформувати постановку задачі проекту для досягнення його головної мети:

- обрати програмний інструментарій для розробки едьютейнмент-додатку;

- сформувати вимоги до едьютейнмент-додатку для його успішного виходу на ринок;

- розробити меню, рівні, а також зібрати актуальну інформацію стосовно алгоритмів програмування для використання її як ресурсу для навчання в едьютейнмент-додатку.

2. ОГЛЯД АРХІТЕКТУРИ ТА ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Вимоги до інформаційної технології, що проектується

Після проведення аналізу предметної області та схожих додатків, ми можемо сформувати постановку задачі та вимоги до інформаційної технології підтримки едьютейнмент-проекту. До вимог, котрим повинен відповідати едьютейнмент-додаток для навчання програмуванню можна віднести наступне:

- збалансованість між кількістю інформації та розважальною складовою;

- наявність інуїтивно зрозумілих елементів головного меню;

- наявність якісної та актуальної інформації у сфері алгоритмів програмування;

- наявність прикладів коду та їх пояснення;

- наявність яскравих прикладів роботи алгоритмів програмування через ігровий досвід;

- комфортний користувацький досвід.

2.2 Вибір алгоритмів програмування для навчання

Після аналізу предметної області та порівняння нашого додатку зі схожими, було прийнято рішення обрати алгоритми програмування, що будуть використані під час розробки інформаційної технології підтримки едьютейнмент-проекту для навчання програмуванню. Серед розглянутих найбільш поширених та актуальних алгоритмів можна обрати по одному кожного типу: сортування даних, пошуку даних, рекурсію, а також роботи з графами.

Алгоритм сортування вибором є не досить швидким для обробки великої кількості даних, але працює без будь-яких проблем з невеликою їх кількістю.

Агоритм бінарного пошуку, як один з різновидів пошуку даних є менш швидкодійним, ніж алгоритм швидкого пошуку, але не використовує у своєму коді інші алгоритми.

Масштаб робіт, які може виконувати алгоритм рекурсії обмежується розміром стеку виконання системи. Тим не менше, цей алгоритм є достатньо корисним для виповнення багатьох задач з програмування.

Алгоритм пошуку найкоротшого шляху, або алгоритм Дейкстри, є різновидом алгоритмів, що працюють з графами. Він є актуальним і у наш час та використовується при розрахунку найкоротшого шляху від точки до точки на картах та інших об’єктах місцевості.

Так як серед алгоритмів сортування даних швидке сортування включає до свого складу інший алгоритм – рекурсію – ми оберемо сортування вибором у якості алгоритму сортування.

Відтак, алгоритмами програмування, що будуть використанні під час розробки едьютейнмент додатку є наступні:

- алгоритм сортування вибором;

- алгритм бінарного пошуку;

- алгритм рекурсії;

- алгоритм Дейкстри, або пошуку найкоротшого шляху.

Кожен з алгоритмів буде розглядатись в окремому рівні на прикладі унікальної ігрової ситуації.

2.3 Вибір програмного інструментарію

Для того, щоб спроектувати Android-додаток, необхідно обрати інструментарій, що дозволить це зробити. До уваги слід взяти ігровий двигун, а також мову програмування. Серед найбільш популярних ігрових двигунів на даний момент можна виділити «Unity» та «Unreal Engine». Кожен з них має

свої особливості, тому слід провести їх порівняльний аналіз та обрати той, що більше нам підходить.

Для зручності зведемо порівняння двигунів «Unity» та «Unreal Engine» до табл. 2.1 [2].

Таблиця 2.1 – Порівняння характеристик ігрових двигунів «Unity» та «Enreal Engine»

Назва ігрового двигуна / Назва характеристики	Unity	Unreal Engine
Тип	Кросплатформний	Кросплатформний
Мови програмування	C#	C++
Особливості	Розробка додатків для мобільних пристроїв, ПК, консолей, система частинок	Розробка додатків для мобільних пристроїв, ПК, консолей, система частинок
Доступ до вихідного коду	Закритий	Відкритий
Ціна	Базова версія безкоштовна	Повністю безкоштовний
Графіка	Чудова якість графіки, але менш досконала, ніж в «Unreal Engine»	Фотореалістична графіка, що використовується в AAA-іграх

Серед усіх цих характеристик до уваги мову програмування, а також ціну додатку. Для розробки нашого додатку обидва двигуни мають усі необхідні функції для проектування графічних елементів.

Серед безкоштовних версій в «Unity» є лише базова, але вона містить увесь необхіний функціонал для успішної розробки едьютейнмент-додатку. Повна версія двигуна «Unreal Engine» є безкоштовною, але вона не має якись суттєвих переваг над «Unity», котрі б знадобились нам при розробці програми для мобільних пристроїв.

Якщо брати до уваги мови програмування, котрі використовують дані двигуни, то більшу перевагу має «Unity» та його «C#», адже ця мова програмування має ряд наступних особливостей [3], які знадобляться нам під час розробки додатку:

- швидкість роботи;
- простота використання;
- об’єктно-орієнтованість;
- автоматичний збір сміття;
- структурованість;
- не має витоків пам’яті;
- платформна незалежність;
- типізованість;
- ефективна робота з файловими ситсемами.

Головною переваго даної мови програмування є її простота використання, тому доцільним буде обрання «Unity» у якості двигуна для розробки додатку, адже він використовує «C#».

2.4 Вибір бібліотек мови програмування «C#»

Для розробки системи збереження, взаємодії з елементами користувацького інтерфейсу, обробки ігрових сцен, роботи з файловою системою та даними, системою введення-виведення, форматуванням даних, а також для роботи з системою обробки подій, було вирішено обрати наступні бібліотеки мови програмування, які є сумісними з ігровим двигуном «Unity»:

- UnityEngine.IO;
- System.IO;
- System.Runtime.Serialization.Formatters.Binary;
- UnityEngine.UI;
- UnityEngine.SceneManagement;
- UnityEngine.EventSystems;

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ЕДЬЮТЕЙНМЕНТ-ДОДАТКА

Даний розділ присвячено опису процесу розробки едьютейнмент-додатку для навчання програмуванню для платформи Android.

3.1 Алгоритм дій користувача для навчання через додаток

У результаті продумування способу навчання через ігровий досвід було прийнято рішення створити схему виконання дій користувачем, що зображена на рис. 3.1.

Згідно з алгоритмом дій, показаним на рис. 3.1, для того, щоб користувач міг освоїти один з алгоритмів програмування, він відкриває додаток, після чого він повинен бачити головне меню, де будуть присутні кнопки налаштувань, переходу в меню рівнів та інформації про додаток. Після переходу в меню рівнів користувач повинен побачити меню з чотирма рівнями, які присвячено алгоритму сортування вибором, бінарного пошуку, рекурсії, а також алгоритму Дейкстри, або пошуку найкоротшого шляху від однієї точки до іншої. Після запуску одного з рівнів користувач може програти або пройти рівень. Якщо для рівня увімкнено режим навчання, то при виграші відкривається екран з роз’ясненнями щодо того, яким чином ігрова конкретна ситуація пов’язана з конкретним алгоритмом програмування. Якщо режим для навчання вимкнено, після виграшу з’явиться виграшне меню з пропозицією перезапуску рівня або виходу в меню рівнів. У випадку програшу в обох випадках – при увімкненому та вимкненому режимах для навчання – відкриється меню програшу з пропозицією перезапуску рівня або виходу в меню рівнів.

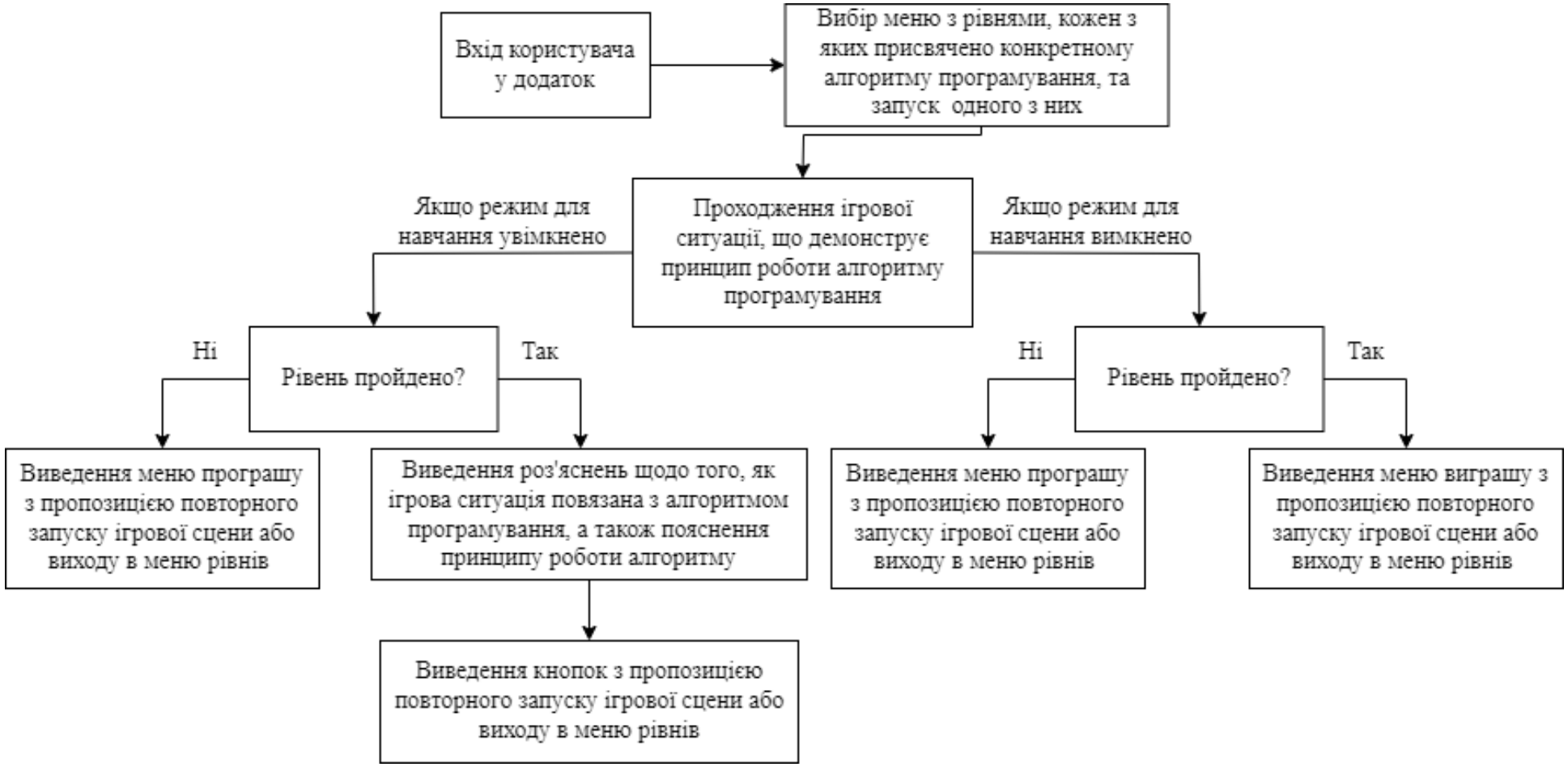

Рисунок 3.1 – Алгоритм дій користувача для взаємодії з едьютейнмент-додатком

3.2 Головне меню

Даний додаток усього містить п’ять ігрових сцен у середовищі розробки «Unity», однією з яких є сцена «MainMenu», а інших чотири – це сцени для рівнів з навчання алгоритмам програмування.

При запуску додатку з самого початку користувач буде бачити головне меню, де присутні три кнопки: кнопка налаштувань, переходу в меню рівнів та інформації про додаток. Демонстрація головного меню наведена на рис. 3.2.

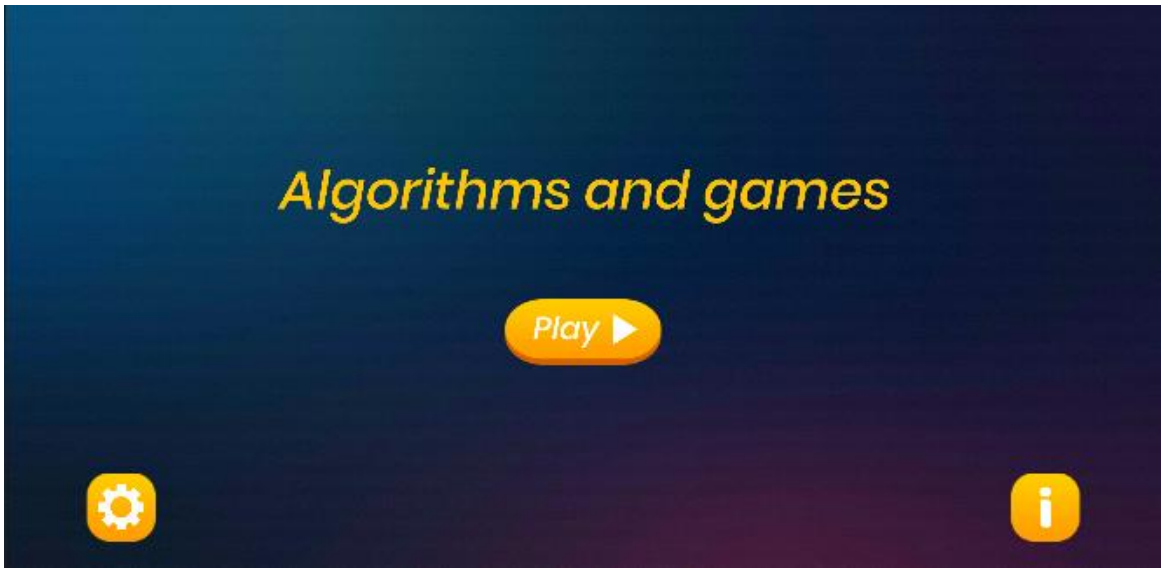

Рисунок 3.2 – Демонстрація головного меню едьютейнмент-додатку для навчання програмуванню

У сцені з головним меню до головної камери, що виводить зображення на екран користувача прікріплено C#-скрипти «MainMenu.cs», «Main.cs», «SaveSystem.cs», «Settings.cs» та «LevelLoader.cs». Ми розберемо прицип роботи цих файлів поступово. Зміст усіх файлів-скриптів проєкту наведено у Додатку А.

Зараз зупинимось на скриптах та їх методах, що спрацьовують при натисненні кнопок у головному меню. При натисненні кнопки налаштувань

(кнопка з шестернею на скриншоті рис. 3.2) спрацьовує метод «OpenSettings()» скрипта «MainMenu.cs» котрий відкриває меню налаштувань (рис. 3.3). При натисненні на кнопку «Play» спрацьовує метод «OpenLevelsMenu()» скрипта «MainMenu.cs», який відкриває меню з рівнями, присвяченими алгоритмам програмування. При натисненні на кнопку з інформацією про додаток спрацьовує метод «OpenAboutMenu()» скрипта «MainMenu.cs», котрий відкриває екран з додатковою інформацією стосовно додатку (рис. 3.4).

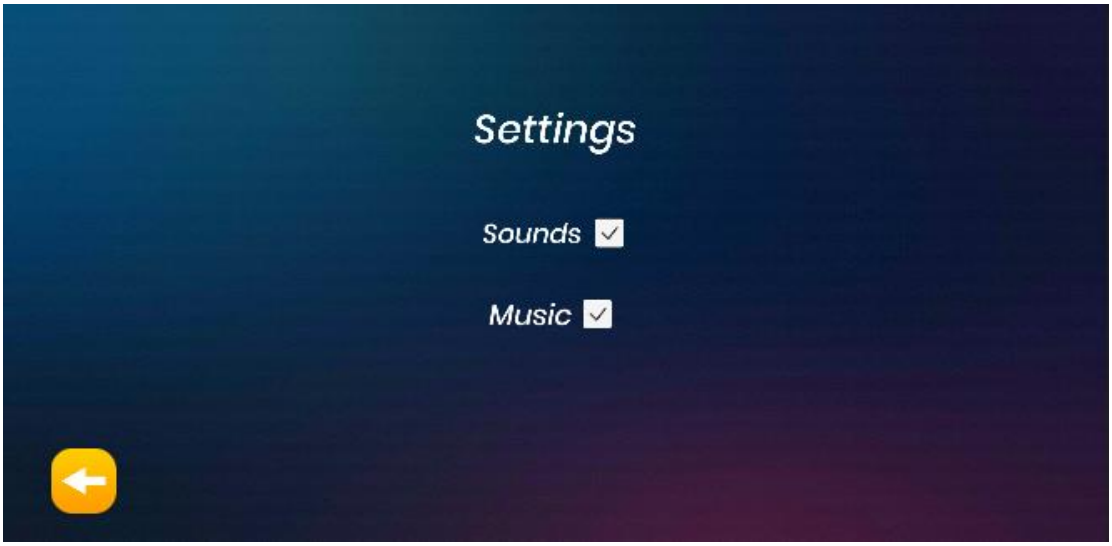

Рисунок 3.3 – Демонстрація меню налаштувань додатка

Рисунок 3.4 – демонстрація меню з додатковою інформацією стосовно програми

У меню налаштувань є дві кнопки-перемикачі, стан яких перевіряється у методі системному методі «Update()» скрипта «Settings.cs», котрий перевіряє стан кнопок. Дані кнопки відповідають за відтворення звуків та музики в рівнях. Їх значення фіксуються та заносяться до статичного скипта «StateMonitor.cs», за допомогою якого ми будемо вмикати та вимикати музику та звуки в рівнях. Також у меню налаштувань є кнопка виходу в головне меню, при натисненні на яку спрацьовує метод «OpenMainMenu()» скрипта «MainMenu.cs».

У меню з додатковою інформацією (рис. 3.4) є лише одна кнопка, при натисненні на яку спрацьовує метод «OpenMainMenu()» скрипта «MainMenu.cs».

На рис. 3.5 наведена ієрархія елементів, присутніх у даній сцені. Ієрархія елементів усіх сцен та значень змінних, прикріплених до елементів скриптів, наведені у Додатку Б.

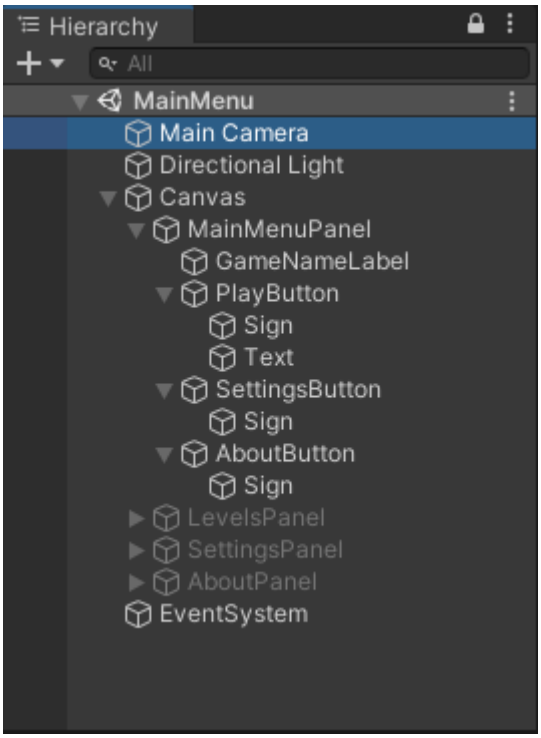

Рисунок 3.5 – Демонстрація ієрархії елементів ігрової сцени «MainMenu» у середовищі розробки «Unity»

3.3 Меню з вибором рівнів для навчання алгоритмам програмування

Меню з вибором рівнів виглядає так, як показано на рис. 3.6. У даному меню присутні кнопки у вигляді зображень для запуску рівнів, кнопка-перемикач для ввімкнення режиму навчання для кожного рівня окремо, а також кнопка виходу в головне меню.

При натисненні на зображення рівня з назвою «SelectionSort» спрацьовує метод «LoadSelectionSortLevel()» скрипта «LevelLoader.cs», котрий завантажує ігрову сцену «SelectionSort», присвячену алгоритму сортування вибором.

При натисненні на зображення рівня з назвою «BinarySearch» спрацьовує метод «LoadBinarySearchLevel()» скрипта «LevelLoader.cs», котрий завантажує ігрову сцену «BinarySearch», присвячену алгоритму бінарного пошуку.

Рисунок 3.6 – Демонстрація меню з рівнями додатка

При натисненні на зображення рівня з назвою «Recursion» спрацьовує метод «LoadRecursionLevel()» скрипта «LevelLoader.cs», котрий завантажує ігрову сцену «Recursion», присвячену алгоритму рекурсії.

При натисненні на зображення рівня з назвою «Dijkstra’s alg.» спрацьовує метод «LoadDijkstrasAlgorithmLevel()» скрипта «LevelLoader.cs», котрий завантажує ігрову сцену «DijkstrasAlgorithm», присвячену алгоритму рекурсії.

Кнопки-перемикачі з назвою «Learning mode» використовуються для відстеження того, який режим увімкнено для конкретного рівня. Їх значення фіксуються у методах завантаження ігрових сцен скрипта «LevelLoader.cs». До значення цих кнопок зберігаються навіть при виході з гри. Така поведінка реалізована за допомогою системи збереження, котра для своєї роботи використовує скрипти «SaveSystem.cs», а також «GameData.cs», який є контейнером для усієї інформації, що зберігається. Принцип роботи системи збереження розглядається у наступному розділі.

3.4 Реалізація системи збереження даних

Система збереження даних у додатку відповідає за збереження режимів для рівнів. Усього в програмі існує два режими – навчальний та звичайний. Коли для рівня увімкнено навчальний режим (активна кнопка-перемикач «Learning mode» під іконкою рівня в меню рівнів) при отриманні перемоги в ігровій сцені відкривається меню з поясненням того, як ігрова ситуація стосується певного алгоритму програмування, а також роз'ясненням принципу роботи алгоритму та сферами застосування алгоритму. Після роз'яснень наводиться код алгоритму на мові програмування «Python» - одній з найбільш популярних та широко використовуваних.

Булеві змінні зі станом активності навчального режиму для кожного рівня містяться в класі (скрипті) «GameData». Цей клас є серіалізованим для того, щоб його об'єкт можна було конвертувати в потік байтів та завантажити в файл.

Також частиною системи збереження є скрипт «SaveSystem.cs». Він містить метод «Awake()» бібліотеки «UnityEngine», метод «SaveGame()», а також метод «LoadGame()».

У методі «Awake()», котрий виконується на початку завантаження ігрової сцени, перевіряється, щоб в один момент часу було створено тільки один екземпляр класу «SaveSystem». У програмуванні такий підхід називають «singleton». Ми використовуємо його у своїй програмі для того, щоб униктути випадкових ситуацій, коли в ігровій сцені присутні декілька екземплярів класу системи збереження. Така ситуація є недопустимою та може призвести до некоректного запису даних, тому ми уникаємо її, використовуючи принцип «singleton» в програмному коді.

Також ми створюємо змінну «filePath», яка буде зберігати шлях до файлу на мобільному пристрої, в якому будуть зберігатись ігрові дані зі статусами активності навчальних режимів для гри. Система сама обирає найбільш

зручний шлях для збереження файлу за допомогою бібліотеки «System.IO». Метод «Awake()» виглядає наступним чином:

```
private void Awake()
{
    if (!instance)
    {
            instance = this;
    }
    else
    {
            Destroy(gameObject);
    }

    filePath = Application.persistentDataPath +
    "/save.data";
}
```

Метод «SaveGame()» виконує функцію збереження даних у грі та працює наступним чином:

- на свій вхід у якості параметру даний метод приймає екземпляр класу «GameData», котрий містить булеві змінні зі значеннями активності навчального режиму для кожного з рівнів;

- далі він відкриває потік даних для запису ігрових даних в файлову систему;

- наступним кроком є переведення даних з потоку в бінарний формат за допомогою бібліотеки «System.Runtime.Serialization.Formatter.Binary» для подальшого збереження у файл;

- після цього потік даних закривається, цим самим записуючи файл з ігровими даними у файлову систему пристрою.

Метод «SaveGame()» виглядає наступним чином:

```
public void SaveGame(GameData saveData)
{
    FileStream dataStream = new FileStream(filePath,
    FileMode.Create);

    BinaryFormatter converter = new BinaryFormatter();
    converter.Serialize(dataStream, saveData);
    dataStream.Close();
}
```

Метод «LoadGame()» виконує функцію завантаження даних з пам’яті пристрою та працює наступним чином:

- даний метод перевіряє, чи існує на пристрої файл за шляхом, прописаним в змінній «filePath», значення якій присвоюється в методі «Awake()»;
- якщо такий файл існує, виконується завантаження даних з цього файлу:
    - спочатку відкривається потік даних для запису в файл за шляхом, вказаним у «filePath», після чого дані з цього файл конвертуються з бінарного вигляду у текстовий;
    - потім створюється екземпляр класу «GameData», булеві змінні зі значеннями активності навчального режиму для рівнів якого зберігаються записуються до цього створеного екзепляру;
    - далі потік даних закривається та метод повертає екземпляр класу «GameData» у якості локальної змінної «saveData».
- якщо файл не існує, то повертається значення типу «null».

Метод «LoadData()» виглядає наступним чином:

```
public GameData LoadGame()
{
     if (File.Exists(filePath))
     {
          FileStream dataStream = new
          FileStream(filePath, FileMode.Open);

          BinaryFormatter converter = new
          BinaryFormatter();
          GameData saveData =
          converter.Deserialize(dataStream) as GameData;

          dataStream.Close();
               return saveData;
     }
     else
     {
          Debug.LogError("Save file not found in " +
          filePath);
          return null;
     }
}
```

Під час запуску сцени з головним меню скрип «Main.cs» створює об’єкт класу «GameData» та намагається знайти файл зі збереженими даними на пристрої за допомогою звернення до змінної «filePath» єдиного в один момент часу класу «SaveSystem». Якщо файл існує, об’єкт класу «GameData» заповнюється даними з файлу за допомогою методу «LoadGame» класу «SaveSystem». Якщо ж файл не існує, то нічого не відбувається. Умова

перевірки наявності файлу потрібна для уникнення виникнень проблем при першому запуску додатку. Усе це виконується в методі «Start()» бібліотеки «UnityEngine», котрий виконується під час рендерингу першого кадру сцени. Даний метод виглядає наступним чином:

```
private void Start()
{
    Time.timeScale = 1f;

    if (File.Exists(SaveSystem.instance.filePath))
    {
            gameData = SaveSystem.instance.LoadGame();
    }
}
```

Збереження даних, тобто запуск методу «SaveGame» класу «SaveSystem» виконується під час натискання на іконки рівнів у меню рівнів. Які саме методи класу «LevelLoader» спрацьовують при натисканні на іконки рівнів розповідається у розділі 3.3 цієї роботи. Слід лише додати, що клас «LevelLoader» містить посилання на клас «Main» для того, щоб використовувати його об’єкт класу «GameData», що збегірається у змінній «gameData».

Приклад коду одного з методів, що завантажують ігрові сцени рівнів та виконують збереження даних перед відкриттям цих сцен, має наступний вигляд:

```
public void LoadSelectionSortLevel()
{
    SaveSystem.instance.SaveGame(main.gameData);
    SceneManager.LoadScene("SelectionSort");
}
```

3.5 Рівень, присвячений навчанню алгоритму сортування вибором

Під час запуску рівня «SelectionSort» в меню рівнів відкривається нова ігрова сцена «Unity» під назвою «SelectionSort». Перше, що бачить користувач – це екран з підсказкою того, що потрібно робити в цьому рівні та кнопка «Start», котра запускає гру, як показано на рис. 3.7.

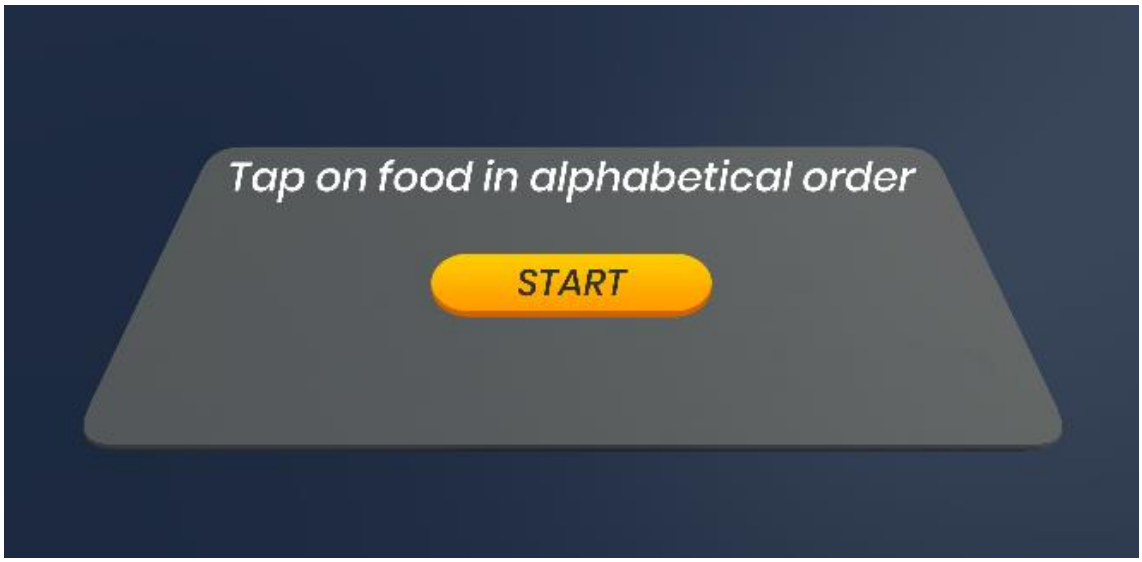

Рисунок 3.7 – Демонстрація стартового екрану після запуску рівня, присвяченого алгоритму сортування вибором

У даній сцені до головної камери прикріплено усього 8 скриптів – «FoodDetector.cs», «FoodSpawner.cs», «TimerBar.cs», «SelectionSortWinPanelLearning.cs», «MainSelectionSort.cs», «Main.cs», «SaveSystem.cs» та «SoundEffector.cs». Про їх призначення поговоримо згодом.

При натисненні кнопки «Start» викликається два методи класу «MainSelectionSort» – «MakeFoodItemsVisible()» і «StartGame()». Розглянемо ці два методи по порядку.

Перший з них приховує стартовий екран з підсказкою та кнопкою «Start», здійснює пошук ігрових об’єктів типу «FoodItem» (тих, до яких прикріплений скрипт «FoodItem.cs» – ми розглянемо його призначення

згодом) та збирає їх у список, після чого робить ці об’єкти видимими та інтерактивними. Цей метод виглядає наступним чином:

```
public void MakeFoodItemsVisible()
{
    startingPanel.SetActive(false);

    FoodItem[] foodItems =
    FindObjectsOfType(typeof(FoodItem)) as FoodItem[];
    foreach (FoodItem item in foodItems)
    {
        item.GetComponent<MeshRenderer>().enabled =
        true;
        item.GetComponent<BoxCollider>().enabled =
        true;
    }
}
```

Другий метод міняє значення змінної «canStartGame» класу «MainSelectionSort» з «false» на «true».

Після того, як стартовий екран закривається, користувач бачить екран, як показано на рис. 3.8. У даній ігровій сцені ми бачимо п’ять об’єктів, до яких прікріплено скрипт «FoodItem.cs» та «FoodAnimation.cs». У кожного з них є анімація обертання навколо своєї осі, яка прописана у скрипті «FoodAnimation.cs». Клас «FoodItem» містить змінні з назвою об’єкта, його кординатами у просторі по осі «Y», числовим кодом, а також посиланням на клас «MainSelectionSort».

Рисунок 3.8 – Демонстрація головного геймплею рівня «SelectionSort»

На рисунку 3.8 можна побачити п’ять різних елементів, але усього їх є десяти різновидів. З них п’ять елементів з’являються випадково після того, як користувач натичсне на кнопку «Start». Виведенням цих об’єктів займається клас «FoodSpawner». У своєму методі «Start()» він викликає три власних методи – «ShuffleZPositions()», «Pick5RandomFoodItems()» та «SpawnFoodItems()».

Перший з цих трьох методів міняє місцями елементи масиву з координатами по осі «Z» у випадковому порядку для того, щоб ігрові об’єкти у вигляді їжі з’являлись у різному порядку. Хоча й з десяти різних елементів обираються п’ять випадкових, без методу «ShuffleZPositions()» кожен з них би завжди з’являвся не лівіше на екрані, ніж попередній, що йде у списку об’єктів «FoodItems» класу «FoodSpawner». Даний метод виглядає наступним чином:

```
private void ShuffleZPositions()
{
    for (int i = 0; i < zFruitPositions.Length; i++)
    {
        float tmp = zFruitPositions[i];
        int r = Random.Range(i,
        zFruitPositions.Length);
```

```
            zFruitPositions[i] = zFruitPositions[r];
            zFruitPositions[r] = tmp;
        }
    }
```

Другий метод зі списку ігрових об’єктів «foodItems» обирає п’ять випадкових та заносить їх до списку ігрових об’єктів «randomFoodItems». Даний метод виглядає наступним чином:

```
    private void Pick5RandomFoodItems()
    {
        randomFoodItems = new List<GameObject>(5);

        for (int i = 0; i < 5; i++)
        {
            if (i < 1)
            {
                randomItem = foodItems[Random.Range(0,
                foodItems.Count - 1)];
            }
            else
            {
                do
                {
                    randomItem =
                    foodItems[Random.Range(0,
                    foodItems.Count - 1)];
                }
                while
                (randomFoodItems.Contains(randomItem));
```

```
            }
            randomFoodItems.Add(randomItem);
        }
    }
```

Третій метод «SpawnFoodItems()» інстанціює об’єкти зі списку «randomFoodItems()» на ігровій сцені з використанням точок по осі «Z», що містяться у масиві типу «zFoodPositions». Даний метод виглядає наступним чином:

```
    private void SpawnFoodItems()
    {
        for (int i = 0; i < randomFoodItems.Count; i++)
        {
            GameObject obj =
            Instantiate(randomFoodItems[i],
            new Vector3(fruitsXPosition,
            randomFoodItems[i].GetComponent<FoodItem>().yP
            osition, zFoodPositions[i]),
            randomFoodItems[i].transform.rotation);
            obj.GetComponent<MeshRenderer>().enabled =
            false;
            obj.GetComponent<BoxCollider>().enabled =
            false;
        }
    }
```

На скриншоті, що зображений на рис. 3.8 також можна побачити шкалу часу та кнопку виходу в меню. При натисненні кнопки меню спрацьовує метод «QuitLevel()» класу «Main», котрий закриває поточну ігрову сцену та відкриває меню рівнів. Шкала часу працює як таймер, тобто при вичерпанні

часу користувач програє, а якщо він встигне натиснути на всі ігрові об’єкти з «FoodItem» в алфавітному порядку до того, як час таймера вичерпається, користувач здобуває перемогу в цьому рівні.

За розрахунок часу, відведеного на натискання ігрових об’єктів зі списку «foodItems» відповідає клас «TimerBar». У своєму методі «Update» бібліотеки «UnityEngine», що виконується кожен кадр, цей скрипт перевіряє розраховує заповненість шкали часу та її колір.

Для розрахунку заповненості шкали відповідає метод «GetCurrentFill()» класу «TimerBar», в якому створюється змінна для поточного відсотку заповнення шкали часу, розрахована як поточне значення шкали, поділене на її максимальне значення. Розраховане значення використовується для зміни параметру «fillAmount» маски (шкали), що відповідає за відображення часу таймера. Метод «GetCurrentFill()» виконується в методі «Update()» кожен під час відтворення кожного кадру сцени та має наступний вигляд:

```
private void GetCurrentFill()
{
    float fillAmount = current / maximum;
    mask.fillAmount = fillAmount;
}
```

Колір шкали змінюється у залежності від поточного значення часу таймера та розраховується у методі «ChangeColor» класу «TimerBar» та виконується кожен під час відтворення кожного кадру сцени в методі «Update()». Даний метод виглядає наступним чином:

```
private void ChangeColor()
{
    if (current >= (maximum * 0.8f))
    {
        bar.GetComponent<Image>().color = green;
```

```
        }
            else if (current >= (maximum * 0.6f))
        {
            bar.GetComponent<Image>().color = lightGreen;
        }
        else if (current >= (maximum * 0.4f))
        {
            bar.GetComponent<Image>().color = yellow;
        }
        else if (current >= (maximum * 0.2f))
        {
            bar.GetComponent<Image>().color = orange;
        }
        else
        {
            bar.GetComponent<Image>().color = red;
        }
    }
```

Запуск таймеру відбувається при натисненні кнопки «Start» на стартовому екрані рівня. За цей запуск відповідає метод «SetTimer()» класу «TimerBar()». У методі «Update()» класу «MainSelectionSort» перевіряється, чи дорівнює значення його змінної «canStartGame» значенню «true». Після натиснення на кнопку «Start» це значення з початкового «false» змінюється на «true», та метод «SetTimer()» спрацьовує. Цей метод має наступний вигляд:

```
public void SetTimer()
{
    if (!stopped)
    {
        timerBarObject.SetActive(true);
```

```
            if (current > 0f)
            {
                current -= Time.deltaTime;
            }
        }
    }
```

Даний метод окрім запуску таймера також віднімає різницю в часі між відтворенням двох сусудніх кадрів від поточного значення залишеного часу.

При натисненні на кожен з ігрових об’єктів типу «FoodItem» спрацьовує метод «CheckIfFoodItemClicked()» класу «FoodDetector». Даний метод виконується в методі «Update()» свого класу. Його основна умова виконується лише у тому випадку, коли користувач натискає на екран. Якщо користувач натискає на ігровий об’єкт типу «FoodItem», то перевіряється, чи є код цього об’єкта «FoodItem.code» найменшим серед списку об’єктів цього типу.

Отримання найменшого коду серед списку «FoodSpawner.randomFoodItems» відбувається за допомогою виконання методу «FindFoodItemSmallestCode()» класу «FoodDetector». Найменший код елемента отримується кожен кадр на початку методу «CheckIfFoodItemClicked()».

Таким чином, якщо код елемента, на який натискає користувач, співпадає з найменшим кодом серед серед списку елементів «randomFoodItems», елемент зникає зі сцени. Якщо ці коди не співпадаються, від шкали часу віднімається одна секунда, а елемент типу «FoodItem» при цьому залишається на сцені.

Метод «CheckIfFoodItemClicked()» виглядає наступним чином:

```
private void CheckIfFoodItemClicked()
{
    if (Input.GetMouseButtonDown(0) &&
    foodSpawner.randomFoodItems.Count != 0 &&
    !mainSelectionSort.startingPanel.activeSelf)
```

```
{
    RaycastHit raycastHit;
    Ray ray =
    Camera.main.ScreenPointToRay(Input.mousePositi
    on);
    FindFoodItemSmallestCode();

    if (Physics.Raycast(ray, out raycastHit,
    100f))
    {
        if (raycastHit.transform != null &&
        raycastHit.transform.gameObject.GetCompone
        nt<FoodItem>() != null)
        {
            int clickeFoodItemCode =
            raycastHit.transform.gameObject.GetCo
            mponent<FoodItem>().code;
            if (clickeFoodItemCode ==
            smallestFoodItemCode)
            {
                DeleteFoodItemFromArray
                (clickeFoodItemCode);
                soundEffector.
                PlayFoodTapOkSound();
                Destroy
                (raycastHit.transform.gameObject)
                ;
                timerBar.AddSeconds(0.25f);
            }
```

```
                else
                {
                    soundEffector.
                    PlayFoodTapNotOkSound();
                    timerBar.AddSeconds(-1f);
                }
            }
        }
    }
```

Метод «FindFoodItemSmallestCode()» виглядає наступним чином:

```
private void FindFoodItemSmallestCode()
{
    smallestFoodItemCode =
    foodSpawner.randomFoodItems[0].transform.gameObject
    .GetComponent<FoodItem>().code;
    for (int i = 1; i <
    foodSpawner.randomFoodItems.Count; i++)
    {
        if (smallestFoodItemCode >
        foodSpawner.randomFoodItems[i].transform.gameO
        bject.GetComponent<FoodItem>().code)
        {
            smallestFoodItemCode =
            foodSpawner.randomFoodItems[i].transform.g
            ameObject.GetComponent<FoodItem>().code;
        }
    }
}
```

Якщо користувач натисне на усі елементи зі списку «randomFoodItems» до вичерпання часу таймера, можливі два сценарії: відкриття виграшного меню з пропозицією перезапуску рівня та виходу в меню рівнів або відкриття меню з роз'ясненням принципу роботи алгоритму швидкого пошуку та поясненням стосовного того, як саме ігрова ситуація пов'язана з цим алгоритмом (рис. 3.9). Меню другого випадку відкривається за допомогою запуску методу «WinLearning()» класу «MainSelectionSort». Перший випадок стається при вимкненому навчальному режимі для рівня, а другий – при ввімкненому.

Виграшне меню при вимкненому режимі навчання має вигляд, як на рис. 3.10. Дане меню відкривається за допомогою запуску методу «Win()» класу «MainSelectionSort».

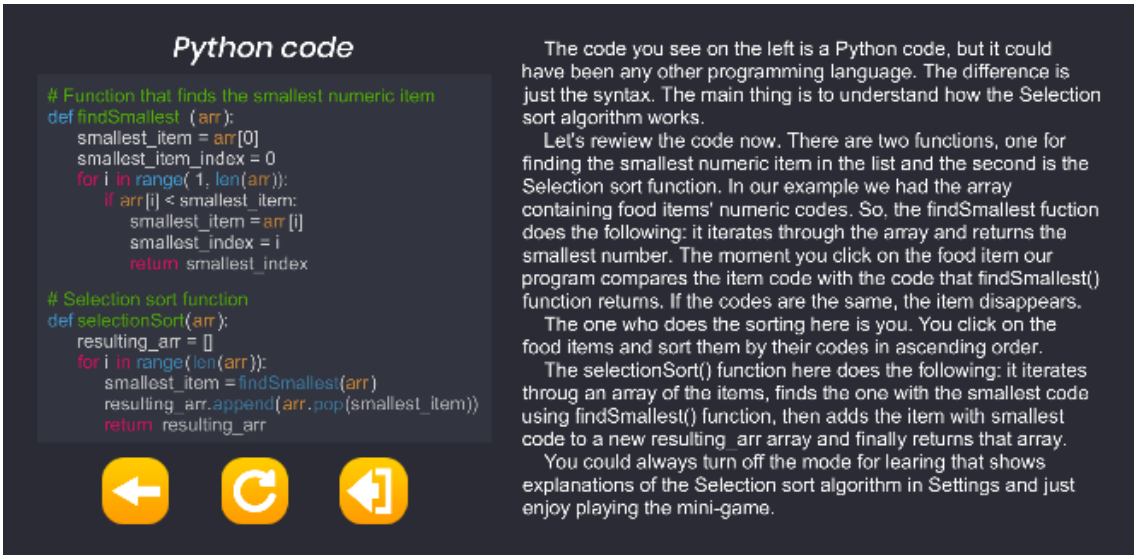

Рисунок 3.9 – Демонстрація виграшного меню при ввімкненому навчальному режимі для рівня «SelectionSort»

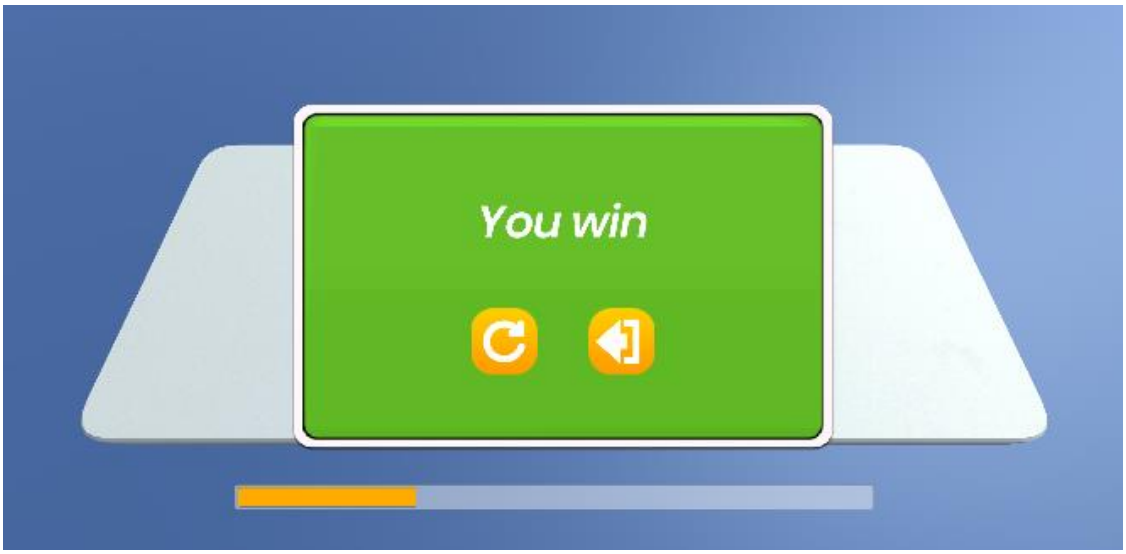

Рисунок 3.10 – Демострація запуску виграшного меню при вимкненому навчальному режимі для рівня «SelectionSort»

Якщо гравець не встигає натиснути на всі елементи типу «FoodItem» до закінчення таймера, активується програшне меню з пропозицією зіграти рівень повторно або вийти в меню рівнів, як показано на рис. 3.11. Це меню відкривається як для активного стану навчального режиму, так і для неактивного. Дане меню відкривається за допомогою запуску методу «Lose()» класу «MainSelectionSort».

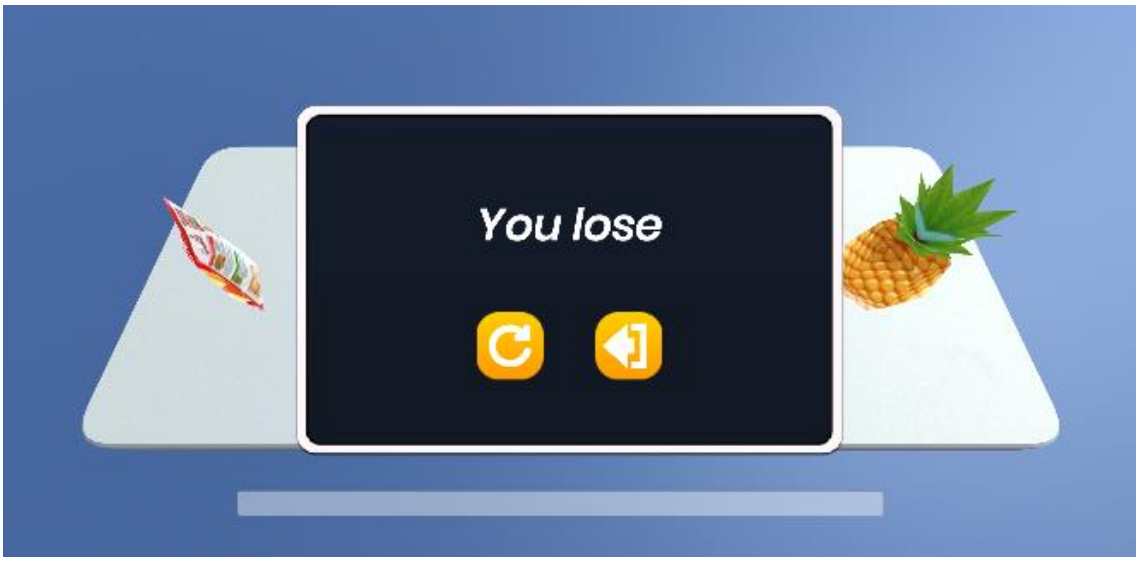

Рисунок 3.11 – Демонстрація програшного меню рівня «SelectionSort»

За відображення UI-елементів у виграшному меню під час активного навчального режиму відповідає скрипт «SelectionSortWinPanelLearning.cs». У даній сцені також присутні звуки та музика, але їх опис проводиться у розділі 3.8 цієї роботи.

3.6 Рівень, присвячений навчанню алгоритму бінарного пошуку

Під час запуску рівня «BinarySearch» в меню рівнів відкривається нова ігрова сцена «Unity» під назвою «BinarySearch». Перше, що бачить користувач – це екран з підсказкою того, що потрібно робити в цьому рівні, калькулятор з кнопками, а текст числа виконаних спроб, а також кнопка виходу в меню рівнів, як показано на рис. 3.12.

Усього в ігровій сцені до головної камери прикріплено вісім скриптів: «Dial.cs», «NumberProcessor.cs», «Main.cs», «MainBinarySearch.cs», «BinarySearchWinPanelLearning.cs», «SaveSystem.cs», «LevelQuitterBinarySearch.cs» та «SoundEffector.cs».

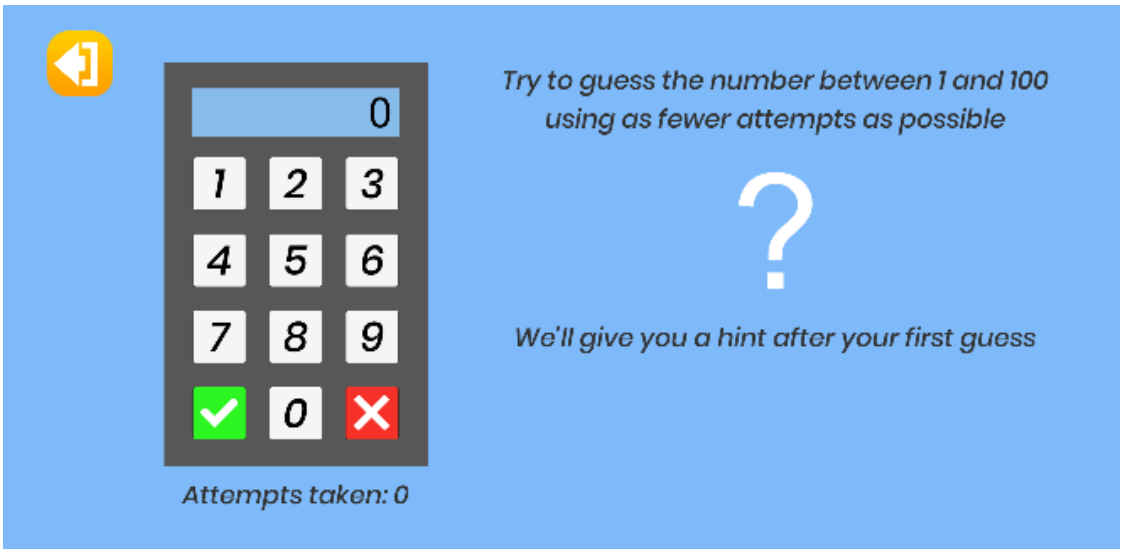

Рисунок 3.12 – Демонстрація стартового екрану рівня «BinarySearch»

Користувачу пропонується вгадати число від одного до ста за нескінченне число спроб. Попри нескінченне число спроб його головною задачею є використати якомога менше спроб.

Користувач може вводити число за допомогою циферблату калькулятора. За введення числа та відображення його на екрані калькулятора відповідає клас «Dial».

До події натиснення кожної цифри калькулятора прикріплений метод «AddNumber()», де замість «Number» стоїть число від нуля до дев’яти.

До події натиснення червоної кнопки калькулятора прив’язани метод «DeleteNumber()» цього ж класу. Цей метод стирає число з екрану калькулятора або замінює число на «0», якщо на момент натиснення червоної кнопки довжина числа на екрані калькулятора дорівнює одній цифрі.

До події натиснення зеленої кнопки калькулятора прив’язаний метод «GuessNumber()» цього ж самого класу. Даний метод відповідає за обробку спроби відгадування секретного чила користувачем. Якщо число, вказане користувачем, більше, ніж секретне число, на екран виводиться підсказка про те, що секретне число є меншим, ніж число користувача. Якщо число користувача є меншим, ніж секретне, виводиться підсказка про те, що секретне є меншим ніж те, що вказав користувач. Це дозволяє зменшити число спроб користувача на відгадування числа.

Якщо для даного рівня увімкнено навчальний режим, за ситуації, коли користувач відгадує секретне число (рис. 3.13), на екрані замість знака питання з’являється секретне число, а під ним виводиться текст.

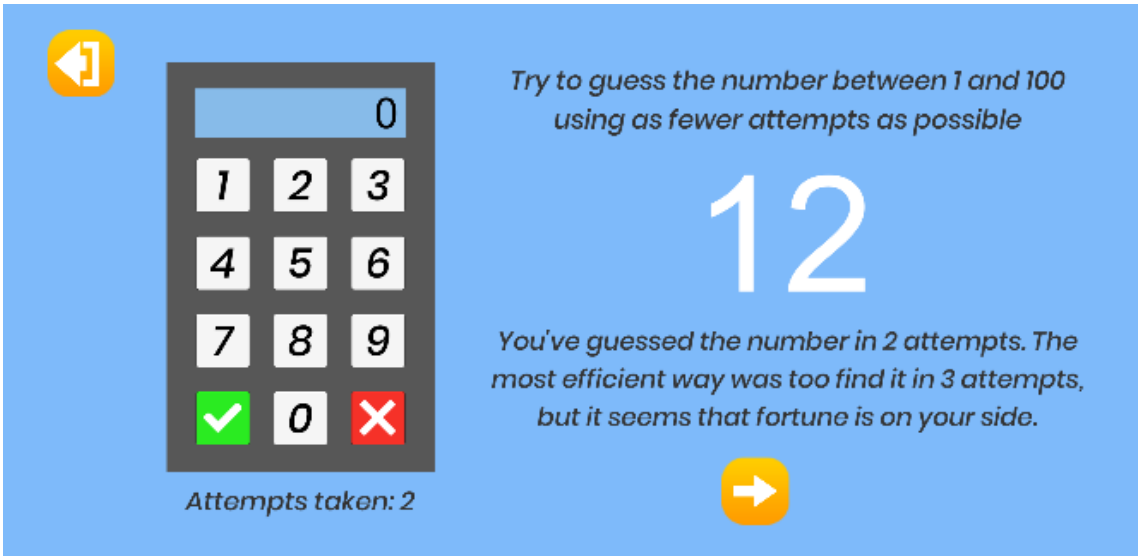

Рисунок 3.13 – Демонстрація виграшного екрану рівня «BinarySearch» при активному режимі для навчання

У цьому тексті говориться про те, за скільки спроб користувач відгадав число, а також про те, якою є найефективніша кількість спроб для вгадування числа. Це найефективніше число спроб розраховується за допомогою методу «GetAttemptsUsingBinarySearch()» класу «NumberProcessor». Дана кількість сроб знаходиться за допомогою використання алгоритму бінарного пошуку, принцип роботи якого було розглянуто у розділі 1.2.2 цієї роботи. Метод «GetAttemptsUsingBinarySearch()» виглядає наступним чином:

```
private int GetAttemtsUsingBinarySearch(int
numberToGuess)
{
    // This function returns the most efficient number
    of attempts
    // in which you can guess the number using Binary
    search

    int attemptCount = 1;
    int lowest = min;
```

```
        int highest = max - 1;
        int mid = highest / 2;

        while (numberToGuess != mid)
        {
                mid = (lowest + highest) / 2;
                if (numberToGuess > mid)
                {
                     lowest = mid + 1;
                     attemptCount++;
                }
                else if (numberToGuess < mid)
                {
                     highest = mid - 1;
                     attemptCount++;
                }
                else
                {

                }
        }
        return attemptCount;
    }
```

Під текстом з найефективнішою кількістю спроб для відгадування секретного числа також з’являється кнопка для переходу в меню роз’яснення щодо того, як саме пов’язана ігрова ситуація з алгоритмом бінарного пошуку та пояснення принципу роботи цього алгоритму.

Якщо навчальний режим вимкнено, то при відгадуванні секретного числа текст під показаним секретним числом залишається таким самим, але з’являються кнопки виходу з рівня та перезапуску рівня (рис. 3.14), до події натиснення яких прив’язані методи «QuitLevel()» та «ReloadLevel()» класу «Main» відповідно.

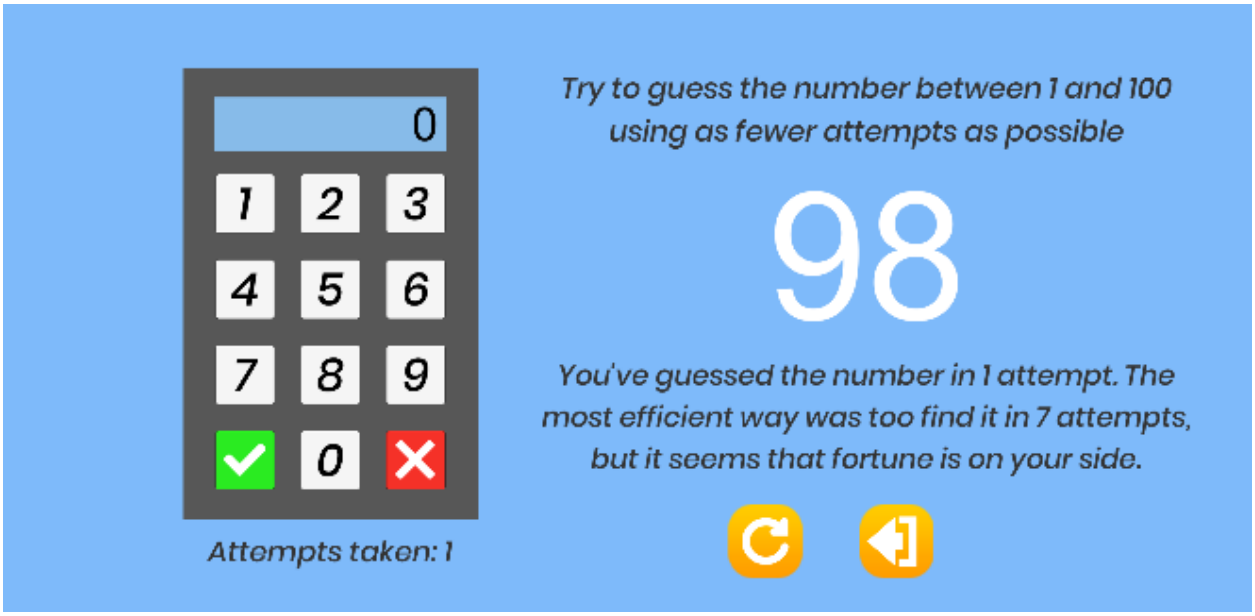

Рисунок 3.14 – Демонстрація виграшного екрану рівня «BinarySearch» при вимкненому режимі для навчання

При натисненні кнопки виходу в меню рівнівнів з’являється меню, в якому є кнопки про підтвердження виходу з рівня (рис. 3.15). За відображення даного меню відповідає скрипт «LevelQuitterBinarySearch.cs», який прикріплений до головної камери в ігровій сцені.

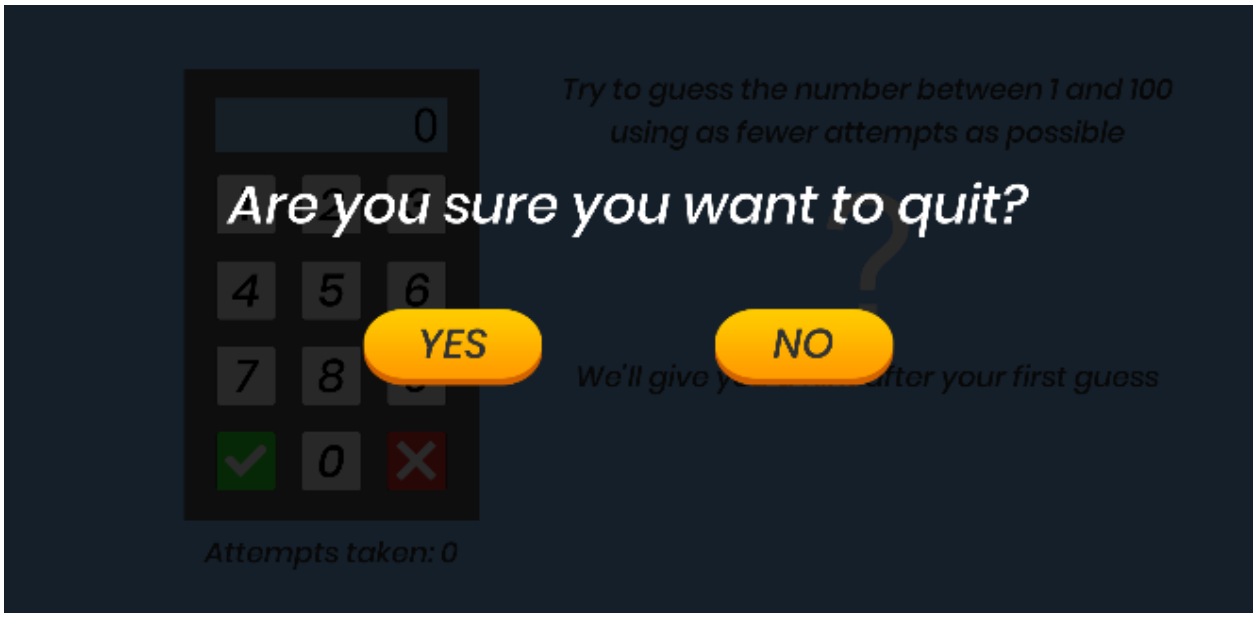

Рисунок 3.15 – Демонстрація меню підтвердження виходу з рівня

До подій натиснення на кнопки «Yes» та «No» у меню підтвердження виходу з рівня прив’язані методи «QuitLevel()» класу «Main» та «StayOnLevel()» класу «LevelQuitterBinarySearch()» відповідно. Метод «StayOnLevel()» прив’язаний до кнопки «No» у меню підтвердження виходу з рівня у всіх наступних рівнях, що розглядаються у роботі далі. Головною відмінністю від поточного методу з класу «LevelQuitterBinarySearch()» є тільки те, що у наступних рівнях метод «StayOnLevel()» прописано окремо в класі «LevelQuitterRecursionAndDijkstras».

3.6 Рівень, присвячений навчанню алгоритму рекурсії

Під час запуску рівня «Recursion» в меню рівнів відкривається нова ігрова сцена «Unity» під назвою «Recursion». Перше, що бачить користувач – це екран з підсказкою того, що потрібно робити в цьому рівні та зворотній відлік від трьох до нуля (рис. 3.16), після закінчення якого користувач може взаємодіяти з грою. За відлік у стартовому меня відповідає скрипт «StartingCountdown.cs».

Рисунок 3.16 – Демонстрація стартового екрану рівня «Recursion»

Після закінчення відліку стартовий екран деактивується, та гравець може натиснути на екран для передачі бомби монстрові. Головною задачею гравця є запобігання ситуації, коли вибухівка підривається в його руках, тому він повинен перекидати її монстрові в руки, як показано на рис. 3.17.

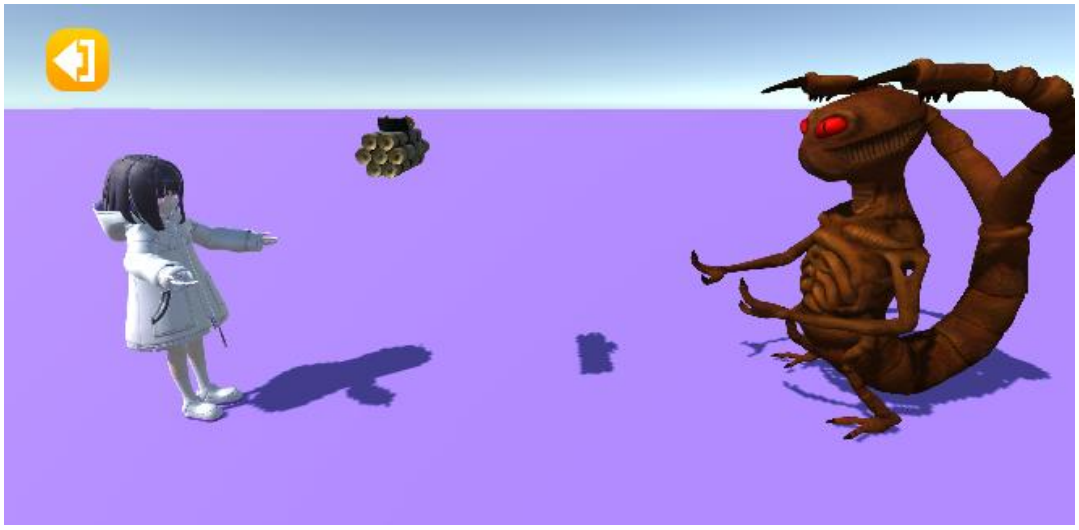

Рисунок 3.17 – Демонстрація поведіки ігрових об’єктів при натисненні на екран

Усього на сцені присутньо десять скриптів. Шість із них прікріплені до головної камери. До цього списку входять наступні скрипти: «MainRecursion.cs», «StartingCountdown.cs», «Main.cs», «SaveSystem.cs», «LevelQuitterRecursionAndDijkstras.cs» та «SoundEffector.cs». Окремі скрипти прикріплено до об’єкту-дівчини, об’єкту-монстра, вибухівки, а також «зони знаходження вибухівки у повітрі». До об’єкту-дівчини прикріплено скрипт «Girl.cs», до об’єкту-монстра – «Monster.cs», до об’єкту-вибухівки – «Bomb.cs», а до «зони знаходження вибухівки у повітрі» – «FlyingArea.cs».

Під «зони знаходження вибухівки у повітрі» мається на увазі ігровий об’єкт, до якого прикріплено box-колайдер з увімкненим параметром «Trigger», що означає можливість проходження об’єктів через нього. У редакторі «Unity» границі колайдера об’єкту з класом «FlyingArea» можна побачити. Вони виглядають так, як показано на рис. 3.18.

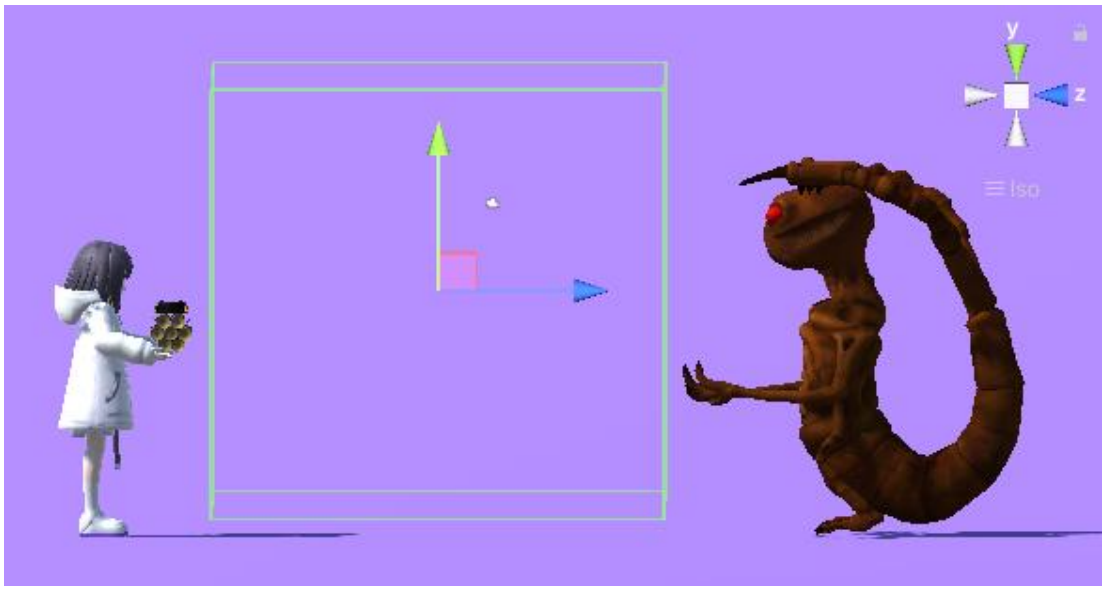

Рисунок 3.18 – Демонстрація виду границь колайдера ігрового об’єкта з класом «FlyingArea» у редакторі «Unity»

На своєму екрані гравець також бачить кнопку виходу з рівня, після натиснення на яку активується меню з підтвердженням виходу з рівня, принцип роботи якого аналогічний, розглянутому в попередньому розділі. Основною відмінністю є лише те, що він також виступає у якості паузи в грі, тобто при активації меню усі таймери та анімації призупинюються.

У обох ігрових об’єктів – як у монстра, так і у дівчини – є свої анімації перекидання вибухівки. Анімації кожного з них відтворюються в методах «Update()» класів «Monster» та «Girl» відповідно.

Для того, щоб користувач міг приблизно розуміти, в який момент вибухне бомба, звук годинника її таймера тим більше пришвидшується, чим менше залишається часу до вибуху. Принцип відтворення всіх аудіо-даних, що присутні в грі, розглядається в розділі 3.8.

Скрипт «SaveSystem.cs» у сценах з рівнями потрібен тільки для завантаження даних за допомогою системи збереження, розглянутої в розділі 3.4 цієї роботи.

Скрипт «Main.cs» у цьому рівні потрібен для виходу з рівня та його перезапуску при натисненні на кнопки у програшному або виграшному меню, які при вимкненому режимі навчання виглядають та функціонують аналогічно до таких самих меню в рівні, присвяченому навчанню алгоритму сортування вибором.

Класи «Girl» та «Monster» мають метод «ThrowBomb()», у першого з яких він спрацьовує при натисненні користувачем на екран, а у другого – через 15 мілісекунд після отримання до руку вибухівки об’єктом-монстром. Кожен з цих методів переміщає ігровий об’єкт-вибухівки з однієї точки в іншу по дузі. Таким чином, вибухівка припиняє свій політ при досягненні однієї з точок. Такими точками виступають координати у просторі рук дівчини та монстра.

У об’єкта-дівчини та об’єкта-монстра також є ігрові об’єкти-точки, позиція яких повторюється для позиції у просторі для вибухівки, поки вона знаходиться в руках монстра або дівчини. Так як у обох об’єктів – монстра та дівчини – ці точки є анімованими, та піднімаються вверх разом з руками при віддтворенні анімації кидання, то вибухівка потрапляє до області детектування колайдера об’єкту з класом «FlyingArea». Ця подія фіксується та записується у змінній «inFlyingArea» класу «Bomb» задля того, щоб уникнути ситуації, коли бомба вибухне у повітрі. Така ситуація не є бажаною, адже нам потрібно, щоб користувач отримував перемогу або поразку. Перемого відповідає ситуації, коли бомба вибухає в руках монстра, а поразка – в руках дівчини.

Вибух бомби (рис. 3.19) реалізовано за допомогою системи частинок, анімація якої запускається під час закінчення таймера бомби та її перебування в руках одного з ігрових об’єктів.

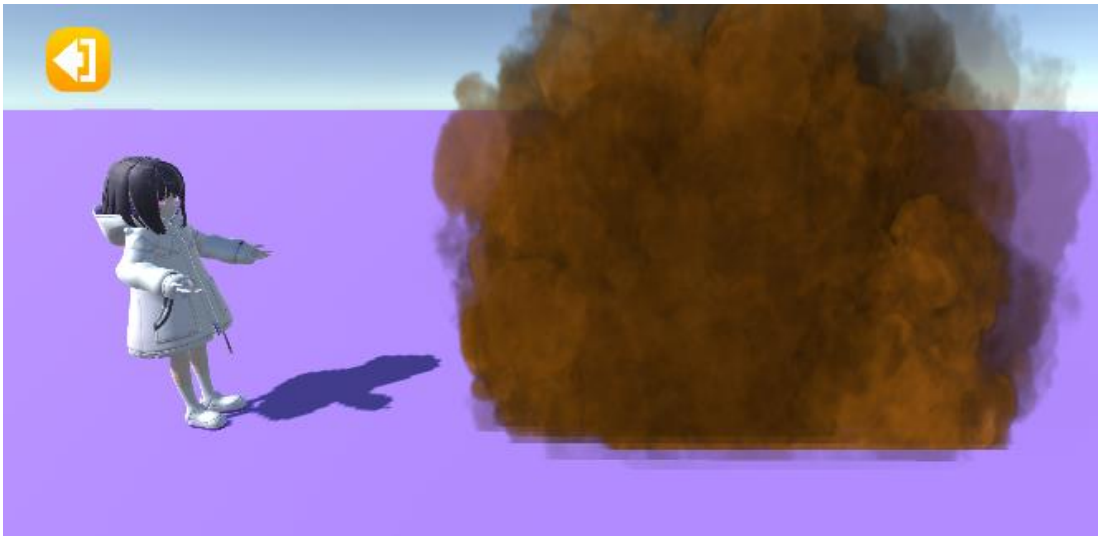

Рисунок 3.19 – Вибух бомби в руках монстра, реалізований за домогою системи частинок «Unity»

Після вибуху бомби в руках монстра та увімкненому режимі навчання з’являється екран з роз’ясненням щодо того, як ігрова ситуація пов’язана з алгоритмом рекурсії, поясненням принципу роботи цього алгоритму, а також кнопками перезапуску рівня та виходу в меню рівнів (рис. 3.20).

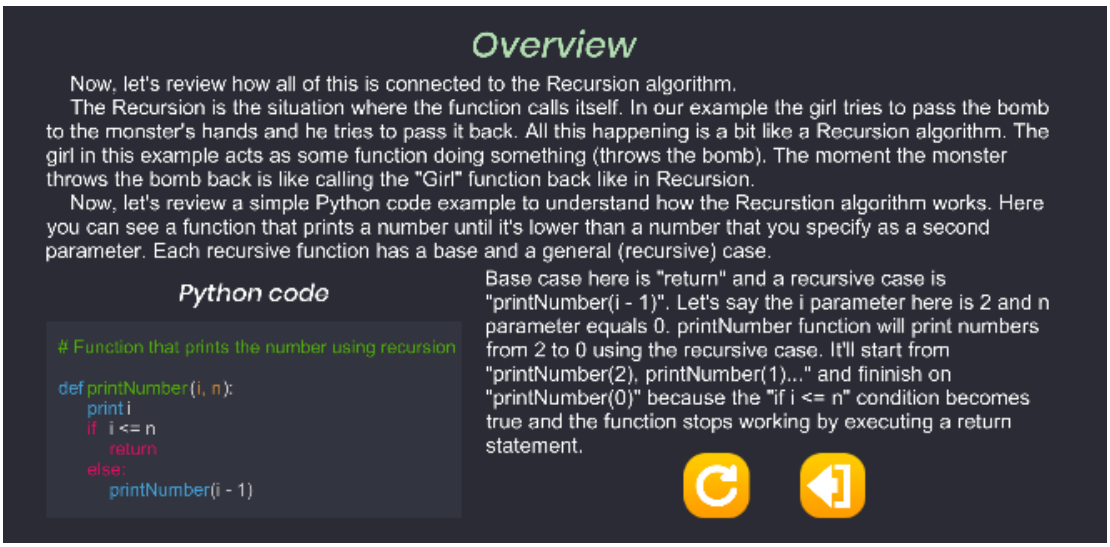

Рисунок 3.20 – Демонстрація екрану роз’яснень щодо того, як ігрова ситуація пов’язана з алгоритмом рекурсії, та пояснень щодо принципу роботи алгоритму

3.7 Рівень, присвячений навчанню алгоритму Дейкстри, або алгоритму пошуку найкоротшого шляху

Під час запуску рівня «Dijkstra’s alg.» в меню рівнів відкривається нова ігрова сцена «Unity» під назвою «DijkstrasAlgorithm». Перше, що бачить користувач – це екран з підсказкою того, що потрібно робити в цьому рівні та зворотній відлік від трьох до нуля (рис. 3.21), після закінчення якого користувач може взаємодіяти з грою. За відлік у стартовому меня відповідає скрипт «StartingCountdown.cs».

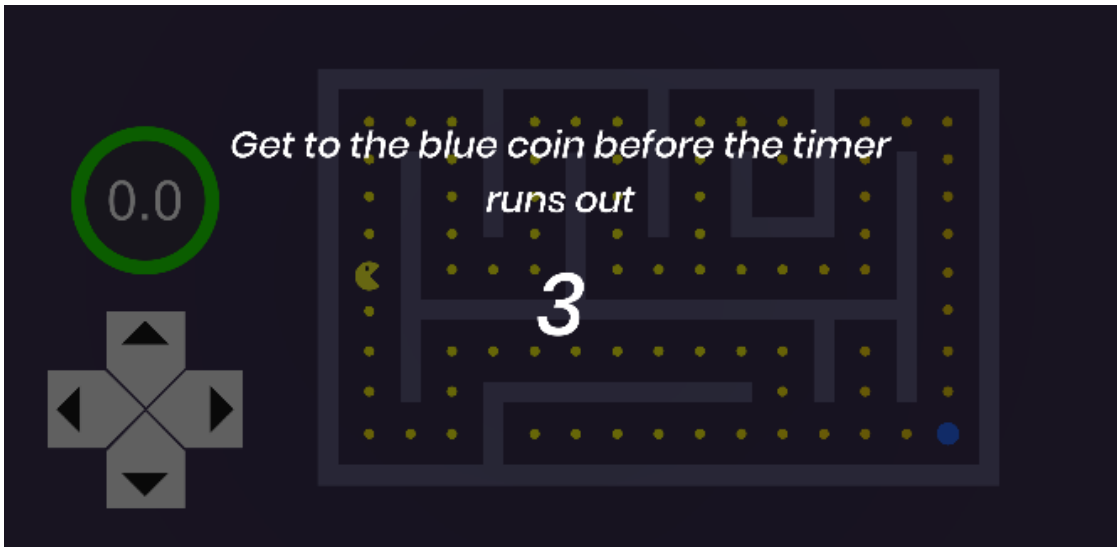

Рисунок 3.21 – Демонстрація стартового екрану рівня «Dijkstra’s alg.»

У даній ігровій сцені присутньо усього двадцять скриптів: «FindShortestPath.cs», «MainDijkstras.cs», «StartingCountdown.cs», «Main.cs», «SaveSystem.cs», «LevelQuitterRecursionAndDijkstras.cs», «SoundEffector.cs», «WallElement.cs», «TurnGuide.cs», «GraphEdge.cs», «GraphNode.cs», «WallDetector.cs», «Coin.cs», «Packman.cs», «PackmanPart.cs», «ButtonUp.cs», «ButtonRight.cs», «ButtonDown.cs», «ButtonLeft.cs» та «DijkstrasWinPanelLearning.cs».

Скрипти «FindShortestPath.cs», «GraphEdge.cs», «GraphNode.cs» відповідаються за знаходження найкоротшого шляху від стартового місцезнаходження гравця до синьої монети. Цей розрахунок відбувається на основі побудови шляхів лабіринту у вигляді графу та знаходження найкоротшого шляху від одного вузла графу до іншого за допомогою алгоритму Дейкстри. Місця в лабіринті, де гравець може змінити напрямок руху виступають у якості вузлів графу, а відрізки, що з’єднують ці вузли – у якості ребер графу. У редакторі «Unity» ми створюємо ребра та вузли графу у якості ігрових об’єктів та розміщуємо їх на шляху проходження лабіринту, як показано на рис. 3.22. Користувач при цьому не бачить цих ігрових об’єктів за рахунок того, що ми деактивуємо компонети цих об’єктів, котрі відповідають за рендеринг їх зображення (рис. 3.23).

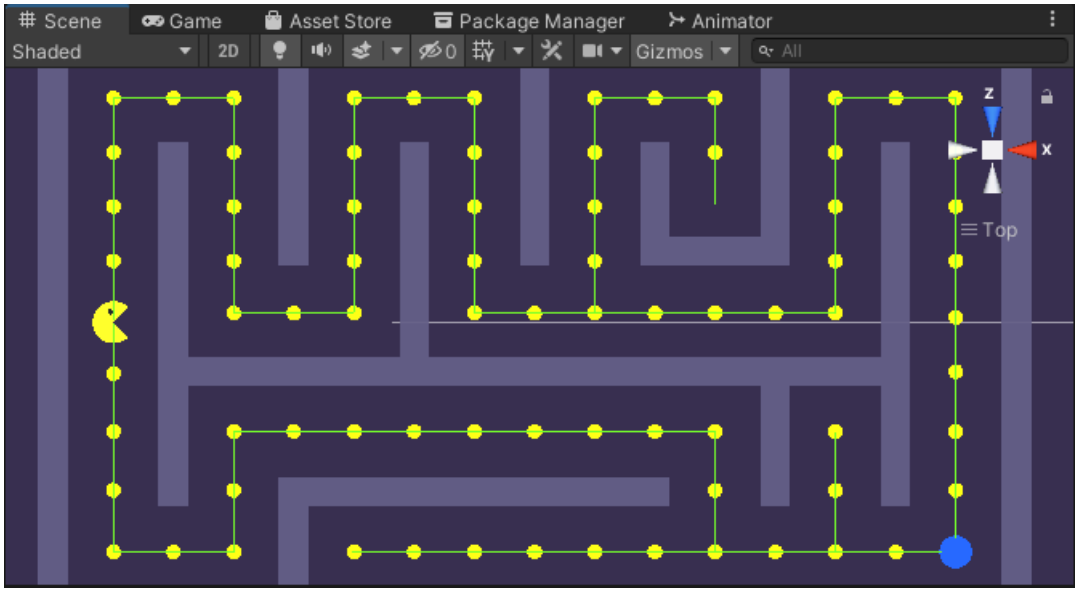

Рисунок 3.22 – Прокладання вузлів та ребер графу вздовж лабіринту в рівні «Dijkstra’s alg.»

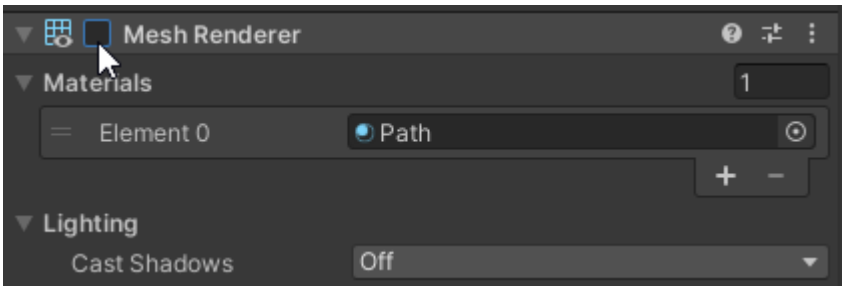

Рисунок 3.23 – Деактивація компоненту «Mesh Renderer» для ігрових об’єктів, що виступають в якості ребер графу

У кожного ребка графу є довжина, яка зберігається у скрипті «GraphEdge.cs» та використовується для розрахунку найкоротшої відстані від стартової точки до синьої монети. Кожне ребро-ігровий об’єкт має свою довжину та вона є різною у значеннях змінної «length» класу «GraphEdge» для кожного з них. Для того, щоб знайти найкоротшу відстань, спочатку в скрипті «GraphNode.cs» в методі «Awake()» за допомогою методу «GetNeigbourCosts()» заповнюється словник «costs» класу «FindShortestPath» зі значеннями вартостей вузлів графа, тобто найкоротших шляхів від стартового вузла. Вартість у вигляді числа на початку буде тільки у вузлів, що

є сусідніми відносно стартового, а в усіх інших – у вигляді нескінченності, адже ми не можемо знати найкоротші шляхи до їх розрахунку. Так як скрипт «GraphNode.cs» прикріплено до кожного об’єкта-вузла графу, то вартості сусідів кожного вузла розраховуються окремо.

Після цього за допомогою виклику методу «GetParent()» також у методі «Awake» для кожного об’єкта вузла окремо до словнику «parents» класу «FindShortestPath» заносяться батьківські вузли для кожного з вузлів, як в алгоритмі Дейкстри.

Далі для кожного вузла у тому ж самому методі «Awake()» за допомогою методу «GetGraphItem()» заповнюється словник з одним елементом, щоб потім отримати словник з вузлами графу, сусідами кожного вузла та їх вартостями у методі «GetGraph()» класу «FindShortestPath».

Маючи словники з вартостями вузлів графа, батьківськими вузлами кожного з них, а також графа, отриманого за допомогою методу «GetGraph()», у тілі цикла «while» в методі «Awake()» класу «FindShortestPath» розраховується найокоротший шлях від стартової точки до синьої монети за допомогою алгоритму Дейкстри.

Отримане після розрахунку значення довжини найкоротшого шляху використовується для визначення точного часу, за який гравець може дістатись від старту до синьої монети, якщо буде йти найкоротшим шляхом. Цей час розраховується з урахуванням швидкості переміщення гравця, а також затримки по декілька секунд в залежності від довжини найкоротшого шляху, але не більше, ніж чотири секунди. Дані розрахунки виконує скрипт «TimerBar.cs», прикріплений до головної камери.

Скрипти «WallElement.cs» та «WallDetector.cs» відповідають за детектування стін гравцем, аби той міг пересуватись строго на одній відстані між двома паралельними стінами лабіринту.

У класі «WallDetector» прописане детектування стін лабіринту за допомогою випускання променів з лівої, верхньої, правої та нижньої сторони гравця. Якщо промінь стикається з ігровим об’єктом, до котрого прикріплено

скрипт «WallElement.cs», напрямок, якого було виявлено стіну лабіринту, фіксується та рух гравця прописнаний в «Packman.cs» припиняється у зафіксованому напрямку.

Скрипт «TurnGuide.cs» прикріпляється до невидимих для користувача стін, які розташовуються в нетипових ситуаціях для детектування стін за допомогою чотирьох променів, наприклад на перехрестях лабіринту, або ситуаціях, коли промінь виявляє лише одну стіну. Ці невидимі стіни детектуються об’єктом-гравця та дають йому змогу змінювати напрямок в нетипових ситуаціях, як показано на рис. 3.24.

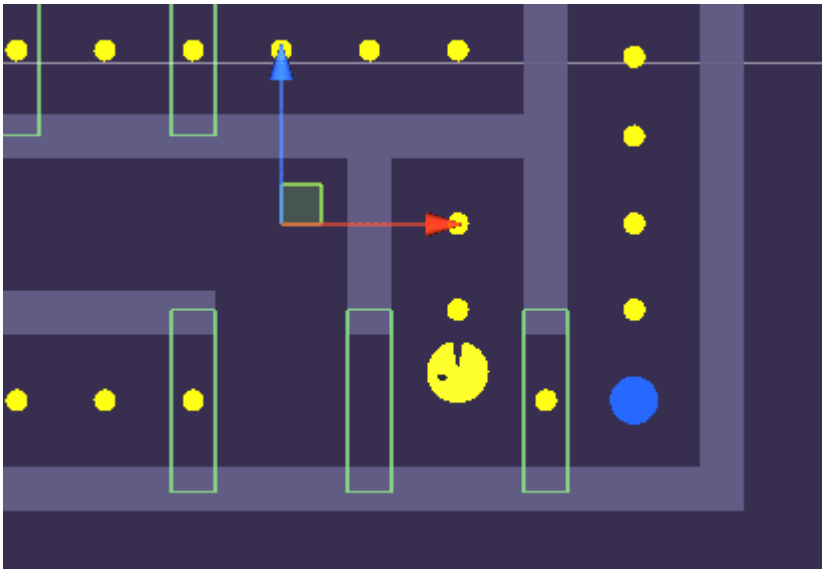

Рисунок 3.24 – Демонстрація вигляду ігрових об’єктів типу «TurnGuide» та їх колайдерів у редакторі «Unity»

До кнопок користувацього інтерфейсу прікріплені скипти «ButtonUp.cs», «ButtonRight.cs», «ButtonDown.cs» «ButtonLeft.cs», котрі повідомляють клас «Packman» про необхідність зміни напрямку руху.

Клас «Coin» прикріпляється до кожної монети та виконує функції видалення монет зі сцени при зіткненні з гравцем. У цього класу є змінна «isBlue», щоб вказати, чи є монета блакитною. Це потрібно для того, щоб при досягненні

гравцем блакитної монети викликати методи виграшу класу «MainDijkstras» для увімкненого режиму навчання або для вимкненого.

При вираші та увімкненому навчальному режимі для даного рівня активується екран з роз’ясненням щодо того, як саме ігрова ситуація пов’язана з алгоритмом Дейкстри, а також поясненням принципу його роботи (рис. 3.25).

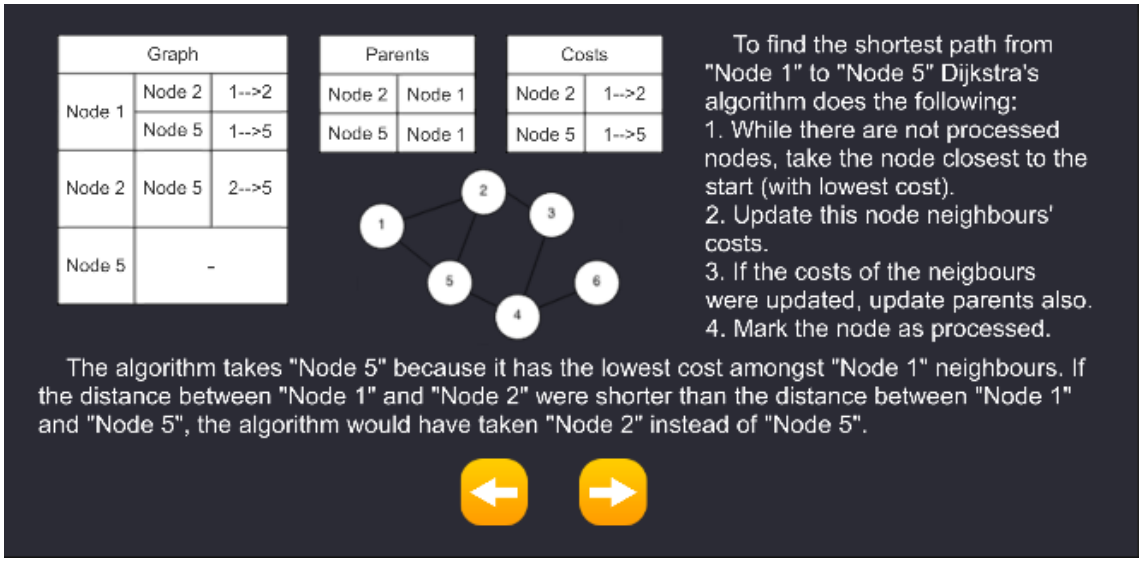

Рисунок 3.25 – Демонстрація однієї зі сторінок з роз’ясненнями після виграшу з активним навчальним режимом для рівня «Dijkstra’s alg.»

3.8 Відтворення аудіо-даних

За відтворення аудіо-даних у додатку відповідає класу «SoundEffector», котрий присутній в ігровій сцені кожного з рівнів. Деякі звуки в рівнях відтворюються за допомогою саме цього класу, а деякі окремо в скриптах. Фонова музика відтворюється в усіх рівнях, окрім «BinarySearch». Відтворення фонової музики виконується за допомогою використання компонентів «AudioSource», які прикріплюються до ігрових об’єктів сцени з іменем «MusicSource» (рис. 3.26 та рис. 3.27), відбувається в скриптах «MainSelectionSort.cs», «MainRecursion.cs» та «MainDijkstras.cs» для рівнів «SelectionSort», «Recursion», «Dijkstra’s alg.» відповідно.

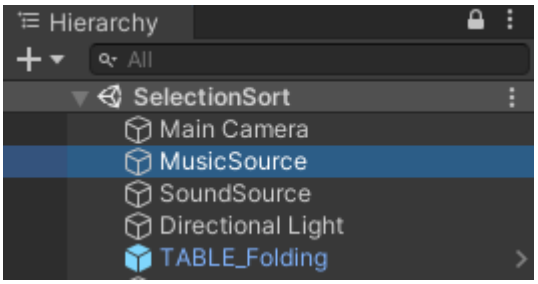

Рисунок 3.27 – Демонстрація ігрового об’єкту з назвою «MusicSource» в ієрархії об’єктів рівня «SelectionSort» у редакторі «Unity»

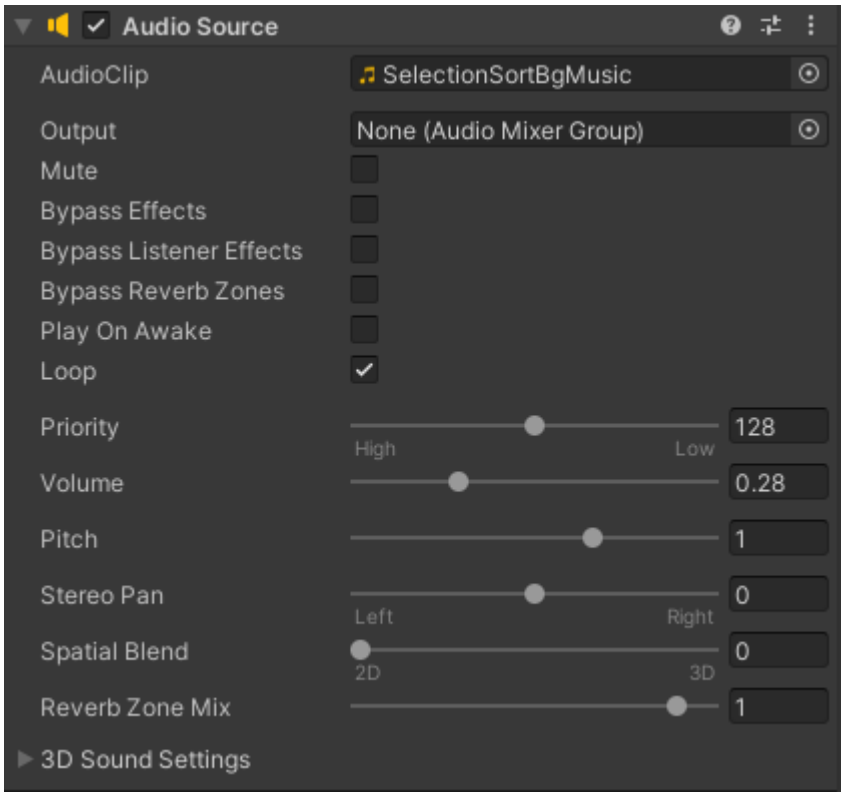

Рисунок 3.28 – Вигляд компонента «AudioSource», прикріпленого до ігрового об’єкта, в інспекторі редактора «Unity»

Не за допомогою класу «SoundEffector» відбувається лише відтворення звуку годинника вибухівки в рівні «Recursion».

У рівні «SelectionSort» є звуки правильного та неправильного натиснення на ігровий об’єкт типу «FoodItem».

У рівні «BinarySearch» відтворюються звуки натиснення на кнопки калькулятора та події правильно відгаданого числа.

У рівні «Recursion», окрім згаданого раніше звуку, відтворюється звук кидання вибухівки, а також звуку вибуху.

У рівні «Dijkstra’s alg.» є лише звуки монет.

ВИСНОВКИ

У результаті проведення аналізу наявних едьютейнмент-додатків для навчання було виявлено, що в усіх з них інформаційна складова переважає над розважальною, тому було вирішено створити власний едьютейнмент-додаток для навчання програмування.

Після цього було сформовано вимоги до проекту, а також обрано технологічні засоби для досягнення мети. У якості програмного забезпечення для розробки додатку було обрано безкоштовну версію «Unity».

Також було сформовано алгоритм алгоритм виконання дій користувачем, що відображає порядок дій користувача для процесу навчання через едьютейнмент-додаток.

Для додатку було розроблено, головне меню, меню налаштувань, меню додаткової інформації, меню рівнів та чотири окемих рівні для демострації роботи алгоритмів програмування через ігрові ситуації. Для кожного рівня також було розроблено меню з роз’ясненням щодо того, як ігрова ситуація пов’язана з конкретним алгоритмом, пояснення принципу його роботи, а також його кодом на мові програмування «Python».

У результаті розробки додатку було задовільнено усі вимоги, викладені на етапі постановки завдання.

СПИСОК ЛІТЕРАТУРИ

1. Н. В. Громова, К. І. Ковальчук, Н. Ю. Кулікова, Едьютейнмент як засіб формування цілісного уявлення про науково-природничу картину світу, Державнй науково-методичний центр змісту культурно-мистецької освіти, Київ, 2020. – 58 с.

2. Unity vs Unreal: Which Game Engine Should You Choose. [Електронний ресурс] – Режим доступу: https://hackr.io/blog/unity-vs-unreal-engine

3. Unreal engine vs. Unity 3D. The 2022 overview. [Електронний ресурс] – Режим доступу: https://pinglestudio.com/blog/full-cycle-development/unreal-engine-vs-unity-3d

4. Aditya Y. Bhargava, Grokking algorithms, Manning Publications Co., 2016.

5. K. Narasimha, Data structures and algorithms made easy: data sturctures and algorithmic puzzles, CareerMonk Publications., 2016.

6. T. H. Cormen, Introduction to algorithms, third edition, The MIT Press, 2009.

7. R. Sedgewick, Algorithms, Addison Wesley; 4th revised edition, 2011.

8. T. H. Cormen, Algorithms Unlocked, The MIT Press, 2013.

9. T. Roughgarden, Algorithms Illuminated: Part 1: The basics, Soundlikeyourself Publishing, 2017.

10. S. Skiena, The algorithm design manual, Springer, 2nd edition, 2008.

11. J. Wengrow, A common-sense guide to data structures and algorithms, O’Reilly Media, 2nd edition, 2020.

12. F. Dedov, The bible of algorithms and data structures, Independently published, 2020.

13. H. Jain, Problems solving in data structures & algorithms using C#, Hermant Jain, 2022.

14. P. Brass, Advanced data structures, Cambridge University Press, 2019.

15. R. Karamagi, Data structures and algorithms in Python, Independently published, 2020.

16. W. Anggoro, C# data sturctures and algorithms, Packt Publishing, 2018.

17. A. V. Aho, J. E. Hopcroft, Data structures and algorithms, Addison Wesley, 1982.

18. J. Kubica, Data structures the fun way, No Starch Press, 2022.

19. Frontiers | Psychosocial Impacts of Mobile Game on K12 Students and Trend Exploration for Future Educational Mobile Games. [Електронний ресурс] – Режим доступу: https://www.frontiersin.org/articles/10.3389/feduc.2022.843090/full

20. E. W. Dijkstra, A note on two problems in connection with graphs, Numerische Mathematik, Vol. 1, pp. 269-271, 1959.

ДОДАТКИ

Додаток А. Лістинг програмного коду

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

[System.Serializable]
public class GameData
{
    public bool selectionSortLearningMode, binarySearchLearningMode,
    recursionLearningMode, dijkstrasLearningMode;
}



using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LevelLoader : MonoBehaviour
{
    public Main main;

    public void LoadSelectionSortLevel()
    {
        SaveSystem.instance.SaveGame(main.gameData);
        SceneManager.LoadScene("SelectionSort");
    }

    public void LoadBinarySearchLevel()
    {
        SaveSystem.instance.SaveGame(main.gameData);
        SceneManager.LoadScene("BinarySearch");
    }

    public void LoadRecursionLevel()
    {
        SaveSystem.instance.SaveGame(main.gameData);
        SceneManager.LoadScene("Recursion");
    }

    public void LoadDijkstrasAlgorithmLevel()
    {
        SaveSystem.instance.SaveGame(main.gameData);
        SceneManager.LoadScene("DijkstrasAlgorithm");
    }
}
```

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

public class LevelQuitterRecursionAndDijkstras : MonoBehaviour
{
    public GameObject quitLevelPanel;
    public Button menuButton;
    public bool inLevelQuitMenu = false;

    public void PromptForQuit()
    {
        Time.timeScale = 0f;
        inLevelQuitMenu = true;
        menuButton.gameObject.SetActive(false);
        quitLevelPanel.SetActive(true);
    }

    public void StayOnLevel()
    {
        quitLevelPanel.SetActive(false);
        inLevelQuitMenu = false;
        menuButton.gameObject.SetActive(true);
        Time.timeScale = 1f;
    }
}
```

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

public class Main : MonoBehaviour
{
    [HideInInspector] public GameData gameData = new GameData();

    private void Start()
    {
        Time.timeScale = 1f;

        if (File.Exists(SaveSystem.instance.filePath))
        {
            gameData = SaveSystem.instance.LoadGame();
        }
    }
```

```
    public void ReloadLevel()
    {
        SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);
    }

    public void QuitLevel()
    {
        SceneManager.LoadScene("MainMenu");
        StateMonitor.QuittedFromLevel = true;
    }
}



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

public class MainMenu : MonoBehaviour
{
    public GameObject MainMenuPanel, LevelsPanel, SettingsPanel, AboutPanel;
    public Toggle selectionSortLearningToggle, binarySearchLearningToggle,
recursionLearningToggle,
    dijkstrasLearningToggle;
    public Main main;

    private void Start()
    {
        if (File.Exists(SaveSystem.instance.filePath))
        {
            selectionSortLearningToggle.isOn =
main.gameData.selectionSortLearningMode;
            binarySearchLearningToggle.isOn =
main.gameData.binarySearchLearningMode;
            recursionLearningToggle.isOn = main.gameData.recursionLearningMode;
            dijkstrasLearningToggle.isOn = main.gameData.dijkstrasLearningMode;
        }
    }

    private void Update()
    {
        if (StateMonitor.QuittedFromLevel && MainMenuPanel != null)
        {
            OpenLevelsMenu();
            StateMonitor.QuittedFromLevel = false;
        }
```

```
    ChangeSelectionSortMode();
    ChangeBinarySearchMode();
    ChangeRecursionMode();
    ChangeDijkstrasMode();
}

public void OpenLevelsMenu()
{
    MainMenuPanel.SetActive(false);
    LevelsPanel.SetActive(true);
}

public void OpenSettings()
{
    MainMenuPanel.SetActive(false);
    SettingsPanel.SetActive(true);
}

public void OpenAboutMenu()
{
    MainMenuPanel.SetActive(false);
    AboutPanel.SetActive(true);
}

public void OpenMainMenu()
{
    LevelsPanel.SetActive(false);
    SettingsPanel.SetActive(false);
    AboutPanel.SetActive(false);
    MainMenuPanel.SetActive(true);
}

public void ChangeSelectionSortMode()
{
    if (selectionSortLearningToggle.isOn)
    {
        main.gameData.selectionSortLearningMode = true;
    }
    else
    {
        main.gameData.selectionSortLearningMode = false;
    }
}

public void ChangeBinarySearchMode()
{
    if (binarySearchLearningToggle.isOn)
    {
        main.gameData.binarySearchLearningMode = true;
    }
```

```
        else
        {
            main.gameData.binarySearchLearningMode = false;
        }
    }

    public void ChangeRecursionMode()
    {
        if (recursionLearningToggle.isOn)
        {
            main.gameData.recursionLearningMode = true;
        }
        else
        {
            main.gameData.recursionLearningMode = false;
        }
    }

    public void ChangeDijkstrasMode()
    {
        if (dijkstrasLearningToggle.isOn)
        {
            main.gameData.dijkstrasLearningMode = true;
        }
        else
        {
            main.gameData.dijkstrasLearningMode = false;
        }
    }
}



using UnityEngine;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

public class SaveSystem : MonoBehaviour
{
    static public SaveSystem instance;
    public string filePath;

    private void Awake()
    {
        if (!instance)
        {
            instance = this;
        }
        else
        {
```

```
            Destroy(gameObject);
        }

        filePath = Application.persistentDataPath + "/save.data";
    }

    public void SaveGame(GameData saveData)
    {
        FileStream dataStream = new FileStream(filePath, FileMode.Create);

        BinaryFormatter converter = new BinaryFormatter();
        converter.Serialize(dataStream, saveData);
        dataStream.Close();
    }

    public GameData LoadGame()
    {
        if (File.Exists(filePath))
        {
            FileStream dataStream = new FileStream(filePath, FileMode.Open);

            BinaryFormatter converter = new BinaryFormatter();
            GameData saveData = converter.Deserialize(dataStream) as GameData;

            dataStream.Close();
            return saveData;
        }
        else
        {
            Debug.LogError("Save file not found in " + filePath);
            return null;
        }
    }
}


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

public class SoundEffector : MonoBehaviour
{
    public AudioSource soundSource;
    public AudioClip foodTapOkSound, foodTapNotOkSound, dialButtonSound,
numberGuessedSound,
    bombThrowSound, bombBlowSound, coinPickUpSound;

    private bool numberGuessedSoundHasPlayed = false;
    public bool bombThrowSoundHasPlayed = false;
    private bool bombBlowSoundHasPlayed = false;
```

```
    private bool packmanSoundHasPlayed = false;

    public void PlayFoodTapOkSound()
    {
        soundSource.PlayOneShot(foodTapOkSound);
    }

    public void PlayFoodTapNotOkSound()
    {
        soundSource.PlayOneShot(foodTapNotOkSound);
    }

    public void PlayDialButtonSound()
    {
        soundSource.PlayOneShot(dialButtonSound);
    }

    public void PlayNumberGuessedSound()
    {
        if (!numberGuessedSoundHasPlayed)
        {
            soundSource.PlayOneShot(numberGuessedSound);
            numberGuessedSoundHasPlayed = true;
        }
    }

    public void PlayBombThrowSound()
    {
        if (!bombThrowSoundHasPlayed)
        {
            soundSource.PlayOneShot(bombThrowSound);
            bombThrowSoundHasPlayed = true;
        }
    }

    public void PlayBombBlowSound()
    {
        if (!bombBlowSoundHasPlayed)
        {
            soundSource.PlayOneShot(bombBlowSound);
            bombBlowSoundHasPlayed = true;
        }
    }

    public void PlayCoinPickUpSound()
    {
        soundSource.PlayOneShot(coinPickUpSound);
    }
}
```

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

public static class StateMonitor
{
    private static bool quittedFromLevel;

    public static bool QuittedFromLevel
    {
        get => quittedFromLevel;
        set
        {
            quittedFromLevel = value;
        }
    }
}
```

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

public class FoodAnimation : MonoBehaviour
{
    public float rotationSpeed;
    public MainSelectionSort mainSelectionSort;

    private void Start()
    {
        mainSelectionSort = Camera.main.GetComponent<MainSelectionSort>();
    }

    private void Update()
    {
        RotateFood();
    }

    private void RotateFood()
    {
        if (!mainSelectionSort.gameLost)
        {
            transform.Rotate(0f, rotationSpeed, 0f, Space.Self);
        }
    }
}
```

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

public class FoodDetector : MonoBehaviour
{
    public FoodSpawner foodSpawner;
    private int smallestFoodItemCode;
    public TimerBar timerBar;
    public MainSelectionSort mainSelectionSort;
    public SoundEffector soundEffector;

    private void Start()
    {
        mainSelectionSort = Camera.main.GetComponent<MainSelectionSort>();
    }

    private void Update()
    {
        CheckIfFoodItemClicked();
    }

    private void CheckIfFoodItemClicked()
    {
        if (Input.GetMouseButtonDown(0) && foodSpawner.randomFoodItems.Count != 0
&&
        !mainSelectionSort.startingPanel.activeSelf)
        {
            RaycastHit raycastHit;
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            FindFoodItemSmallestCode();

            if (Physics.Raycast(ray, out raycastHit, 100f))
            {
                if (raycastHit.transform != null &&
raycastHit.transform.gameObject.GetComponent<FoodItem>() != null)
                {
                    int clickeFoodItemCode =
raycastHit.transform.gameObject.GetComponent<FoodItem>().code;
                    if (clickeFoodItemCode == smallestFoodItemCode)
                    {
                        DeleteFoodItemFromArray(clickeFoodItemCode);
                        soundEffector.PlayFoodTapOkSound();
                        Destroy(raycastHit.transform.gameObject);
                        timerBar.AddSeconds(0.25f);
                    }
                    else
                    {
                        soundEffector.PlayFoodTapNotOkSound();
                        timerBar.AddSeconds(-1f);
```

```
                    }
                }
            }
        }
    }

    private void FindFoodItemSmallestCode()
    {
        smallestFoodItemCode =
foodSpawner.randomFoodItems[0].transform.gameObject.GetComponent<FoodItem>().code
;
        for (int i = 1; i < foodSpawner.randomFoodItems.Count; i++)
        {
            if (smallestFoodItemCode >
foodSpawner.randomFoodItems[i].transform.gameObject.GetComponent<FoodItem>().code
)
            {
                smallestFoodItemCode =
foodSpawner.randomFoodItems[i].transform.gameObject.GetComponent<FoodItem>().code
;
            }
        }
    }

    private void DeleteFoodItemFromArray(int foodItemCode)
    {
        for (int i = 0; i < foodSpawner.randomFoodItems.Count; i++)
        {
            if (foodItemCode ==
foodSpawner.randomFoodItems[i].transform.gameObject.GetComponent<FoodItem>().code
)
            {
                foodSpawner.randomFoodItems.Remove(foodSpawner.randomFoodItems[i]
);
            }
        }
    }
}



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

public class FoodItem : MonoBehaviour
{
    public string itemName;
    public float yPosition;
    public int code;
```

```
    public MainSelectionSort mainSelectionSort;

    private void Start()
    {
        mainSelectionSort = Camera.main.GetComponent<MainSelectionSort>();
    }

    private void Update()
    {
        if (mainSelectionSort.gameLost)
        {
            gameObject.GetComponent<BoxCollider>().enabled = false;
        }
        else
        {
            gameObject.GetComponent<BoxCollider>().enabled = true;
        }
    }
}



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

public class FoodSpawner : MonoBehaviour
{
    public List<GameObject> foodItems;
    private GameObject randomItem;
    public List<GameObject> randomFoodItems;
    public float[] zFoodPositions;
    public float fruitsXPosition;

    private void Start()
    {
        ShuffleZPositions();
        Pick5RandomFoodItems();
        SpawnFoodItems();
    }

    private void Pick5RandomFoodItems()
    {
        randomFoodItems = new List<GameObject>(5);

        for (int i = 0; i < 5; i++)
        {
            if (i < 1)
            {
```

```
                randomItem = foodItems[Random.Range(0, foodItems.Count - 1)];
            }
            else
            {
                do
                {
                    randomItem = foodItems[Random.Range(0, foodItems.Count - 1)];
                }
                while (randomFoodItems.Contains(randomItem));
            }
            randomFoodItems.Add(randomItem);
        }
    }


    private void ShuffleZPositions()
    {
        for (int i = 0; i < zFoodPositions.Length; i++)
        {
            float tmp = zFoodPositions[i];
            int r = Random.Range(i, zFoodPositions.Length);
            zFoodPositions[i] = zFoodPositions[r];
            zFoodPositions[r] = tmp;
        }
    }

    private void SpawnFoodItems()
    {
        for (int i = 0; i < randomFoodItems.Count; i++)
        {
            GameObject obj = Instantiate(randomFoodItems[i],
            new Vector3(fruitsXPosition,
randomFoodItems[i].GetComponent<FoodItem>().yPosition, zFoodPositions[i]),
            randomFoodItems[i].transform.rotation);
            obj.GetComponent<MeshRenderer>().enabled = false;
            obj.GetComponent<BoxCollider>().enabled = false;
        }
    }
}
```

Додаток Б. Ієраріхія елементів та скрипти у інспекторі в редакторі «Unity»

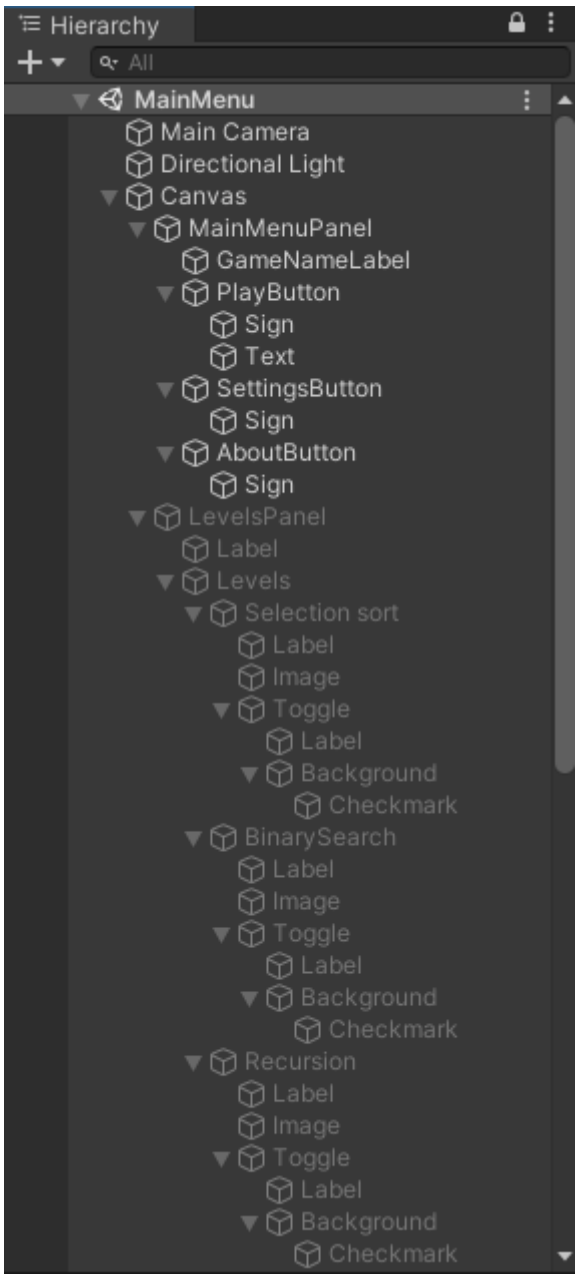

Рисунок Б.1 – Ієрархія елементів в ігровій сцені «MainMenu»

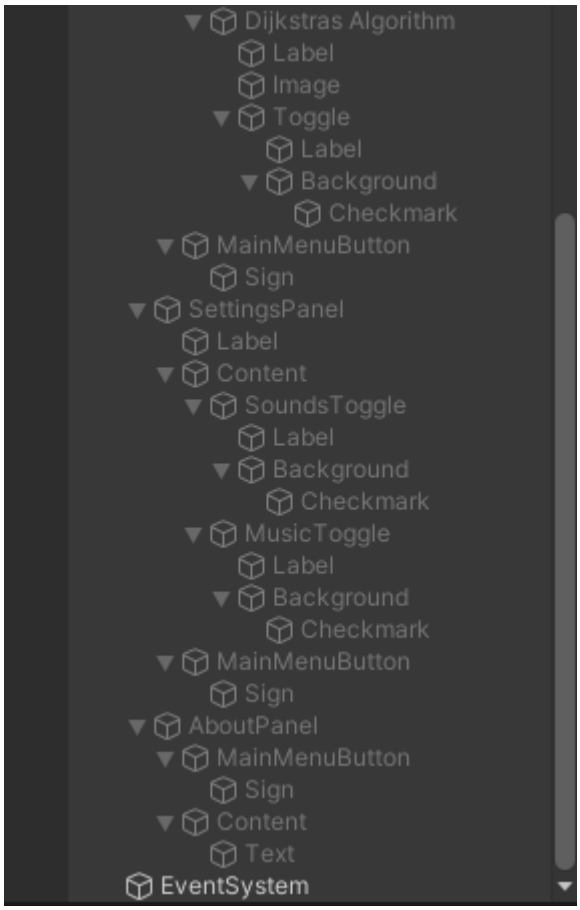

Рисунок Б.2 – Ієрархія елементів в ігровій сцені «MainMenu» (продовження)

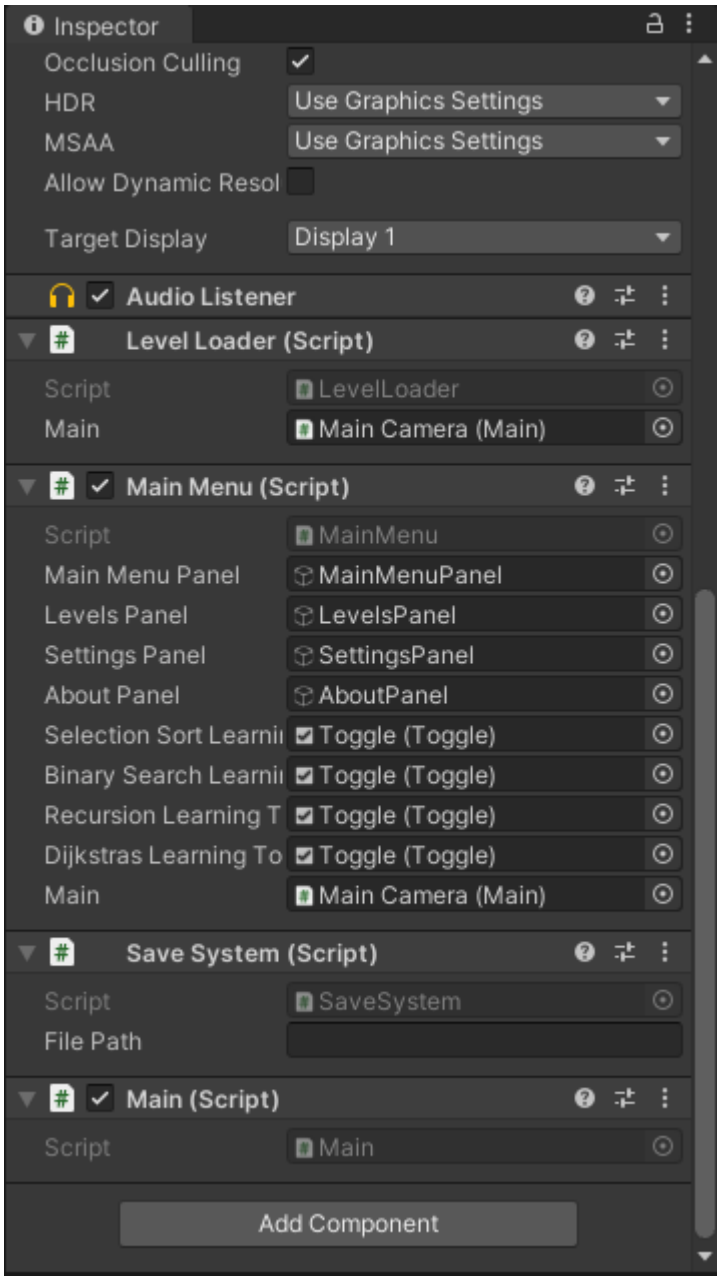

Рисунок Б.3 – Інспектор компонентів головної камери в сцені «MainMenu»

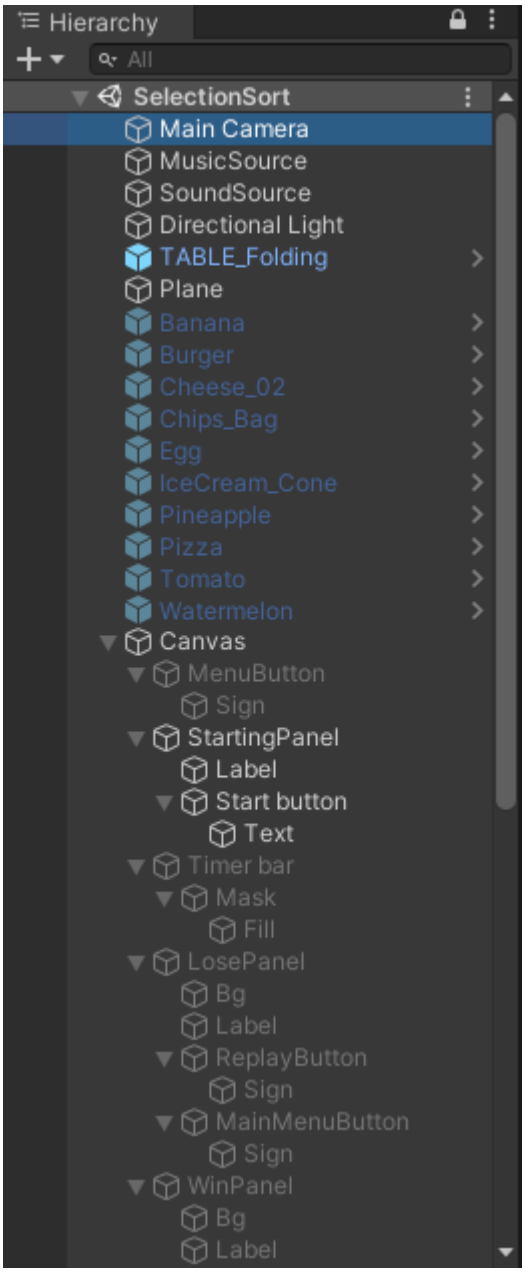

Рисунок Б.4 – Ієрархія елементів в ігровій сцені «SelectionSort»

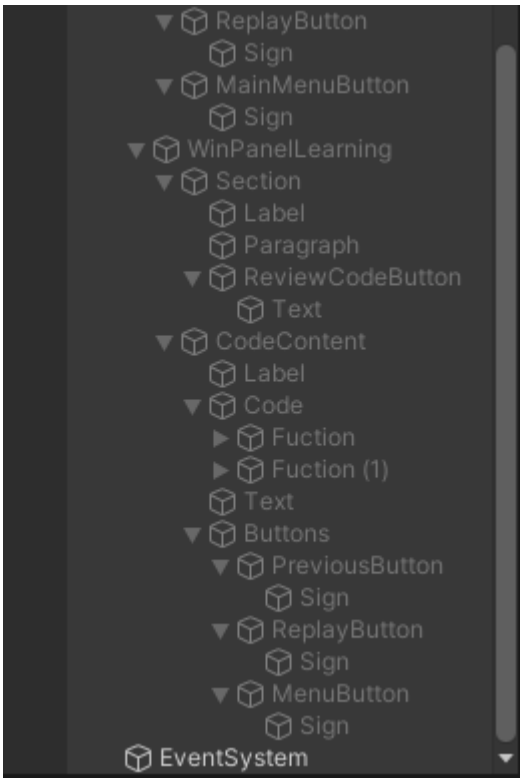

Рисунок Б.5 – Ієрархія елементів в ігровій сцені «SelectionSort» (продовження)

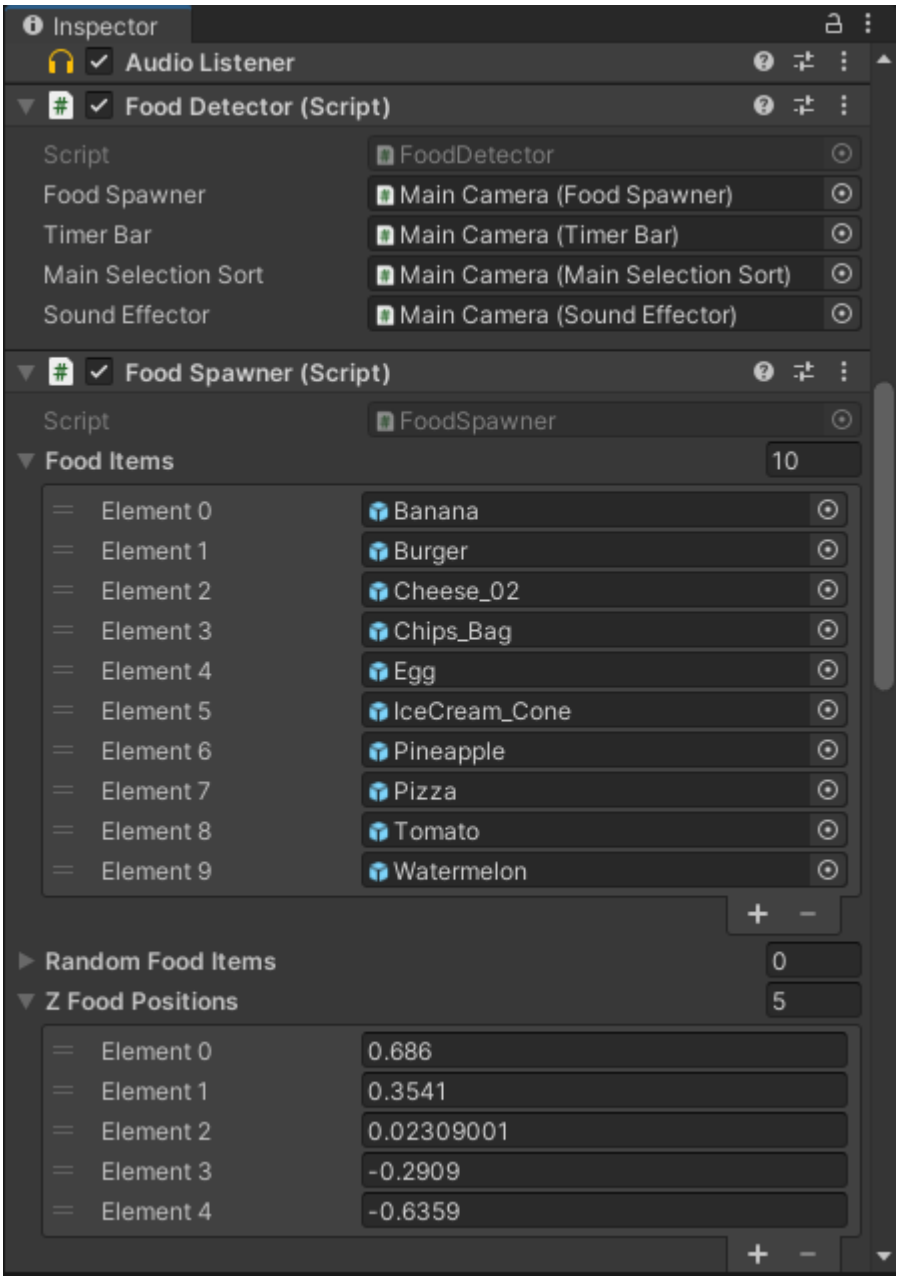

Рисунок Б.6 – Інспектор компонентів головної камери в сцені «SelectionSort»

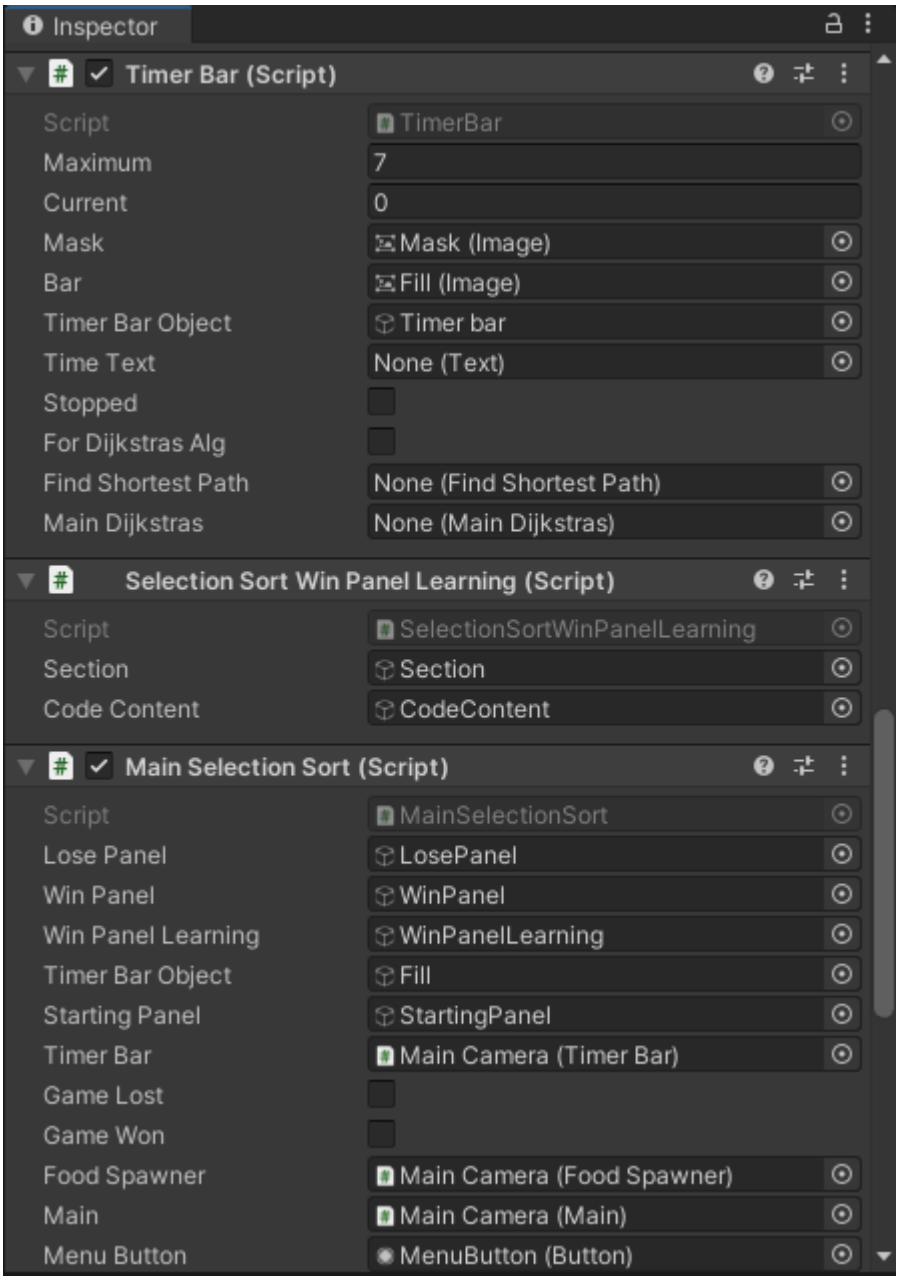

Рисунок Б.7 – Інспектор компонентів головної камери в сцені «SelectionSort» (продовження)

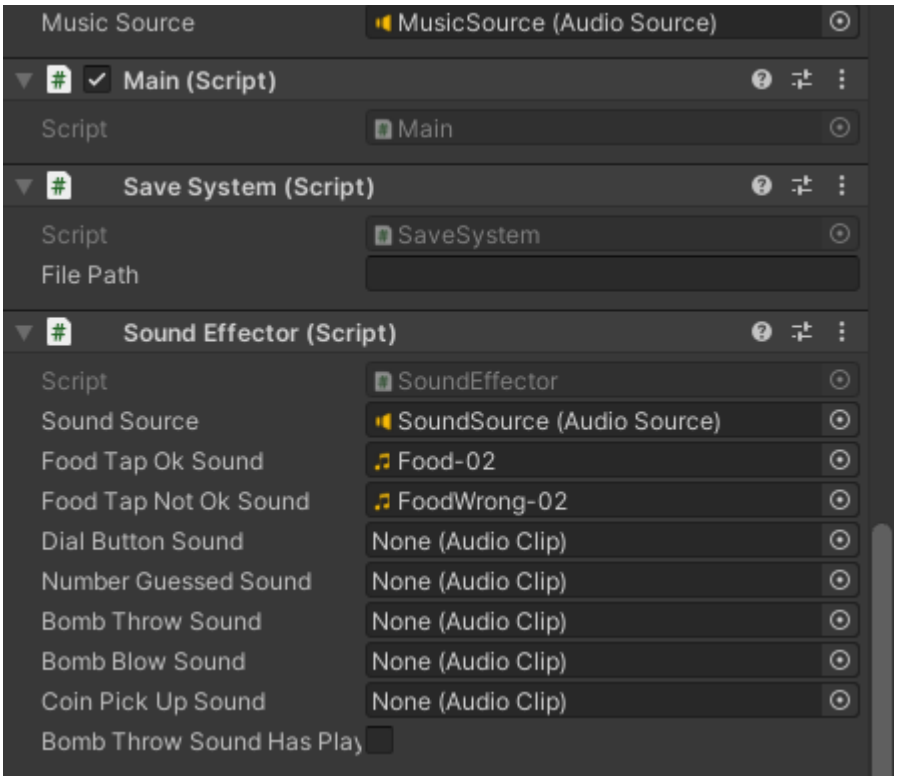

Рисунок Б.8 – Інспектор компонентів головної камери в сцені «SelectionSort» (продовження 2)

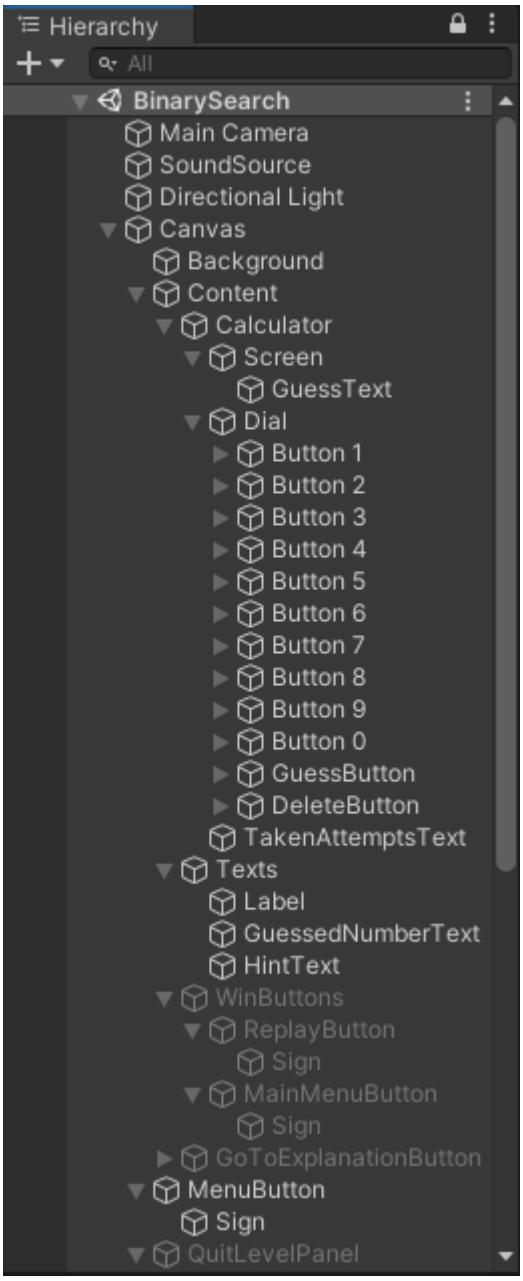

Рисунок Б.9 – Ієрархія елементів в ігровій сцені «BinarySearch»

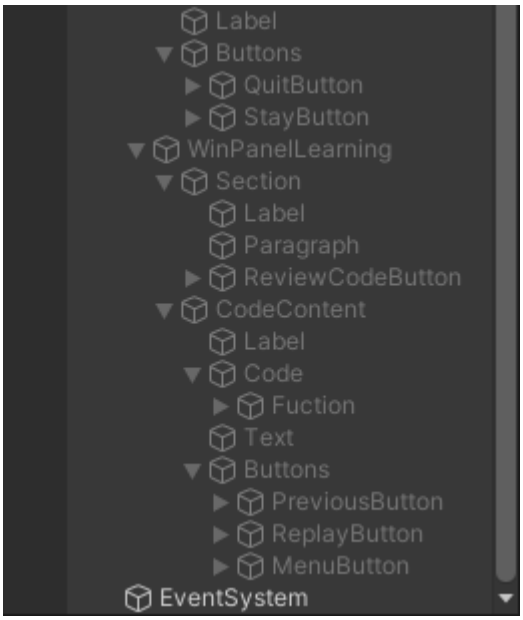

Рисунок Б.10 – Ієрархія елементів в ігровій сцені «BinarySearch» (продовження)

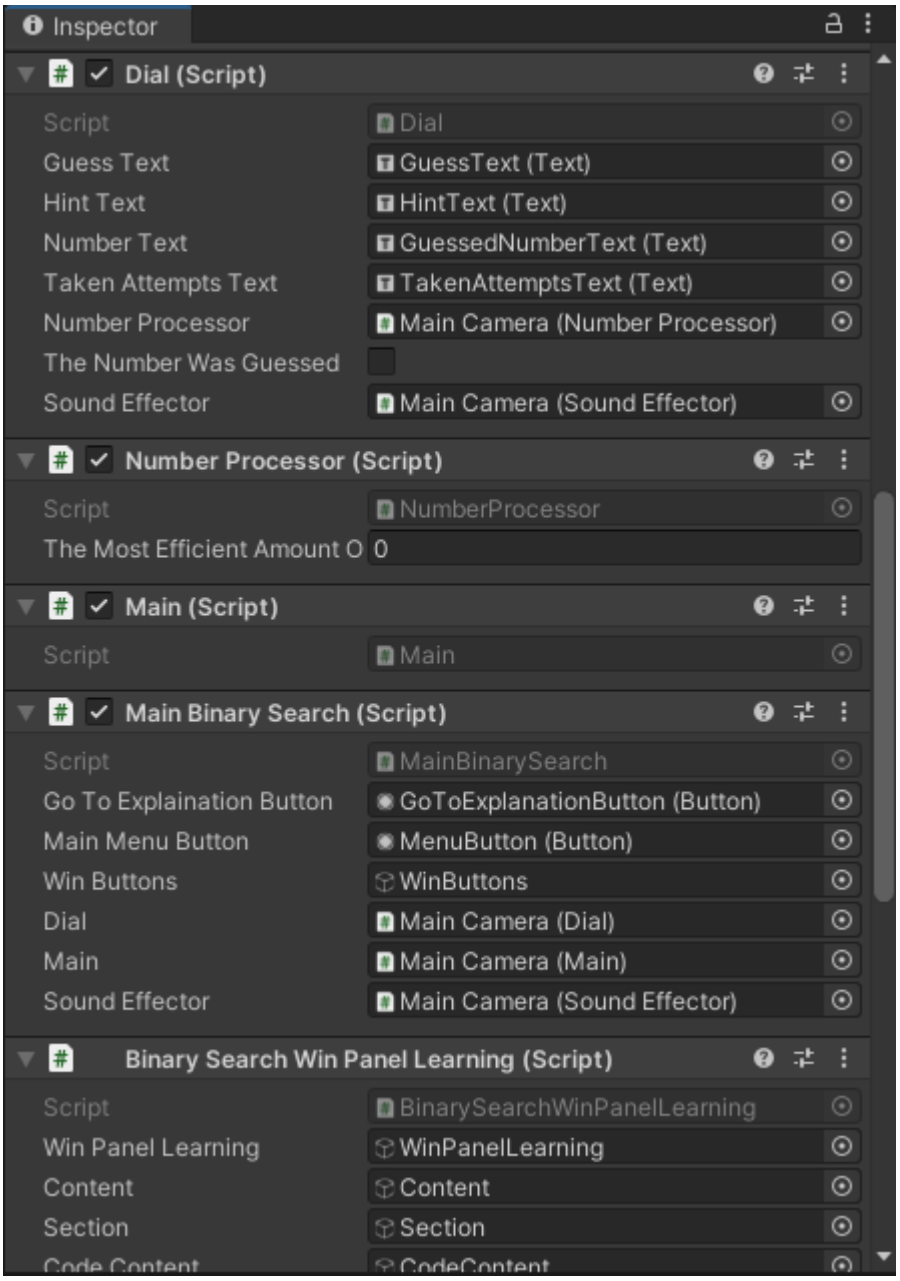

Рисунок Б.11 – Інспектор компонентів головної камери в сцені «BinarySearch»

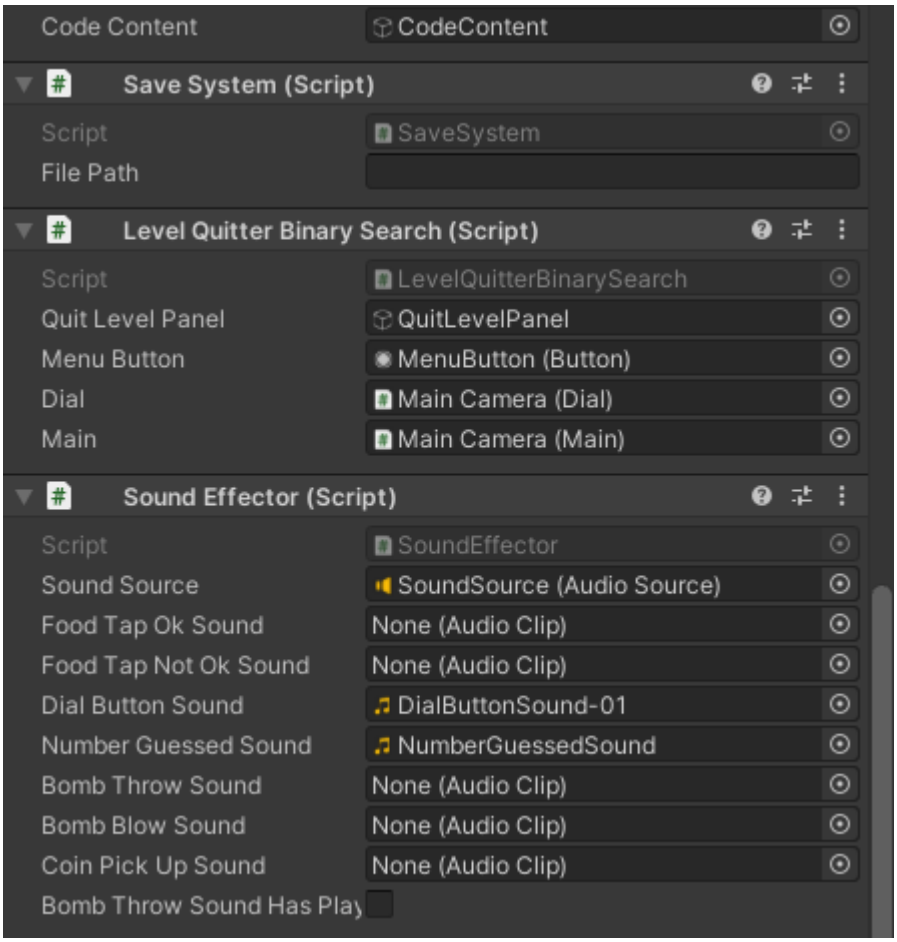

Рисунок Б.12 – Інспектор компонентів головної камери в сцені «BinarySearch» (продовження)

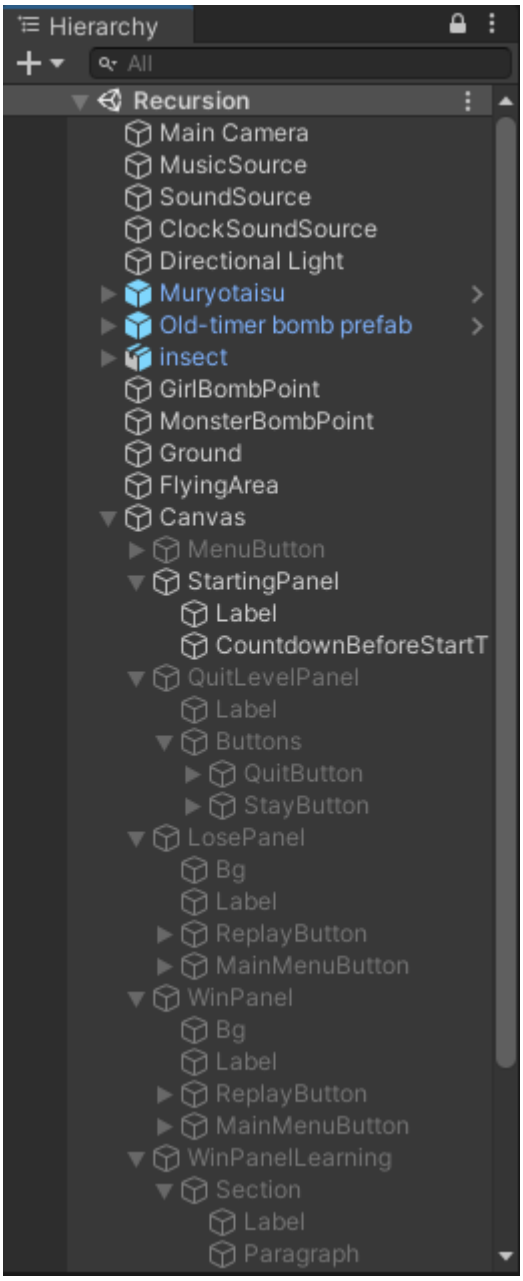

Рисунок Б.13 – Ієрархія елементів в ігровій сцені «Recursion»

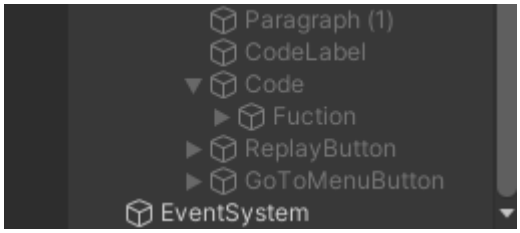

Рисунок Б.14 – Ієрархія елементів в ігровій сцені «Recursion» (продовження)

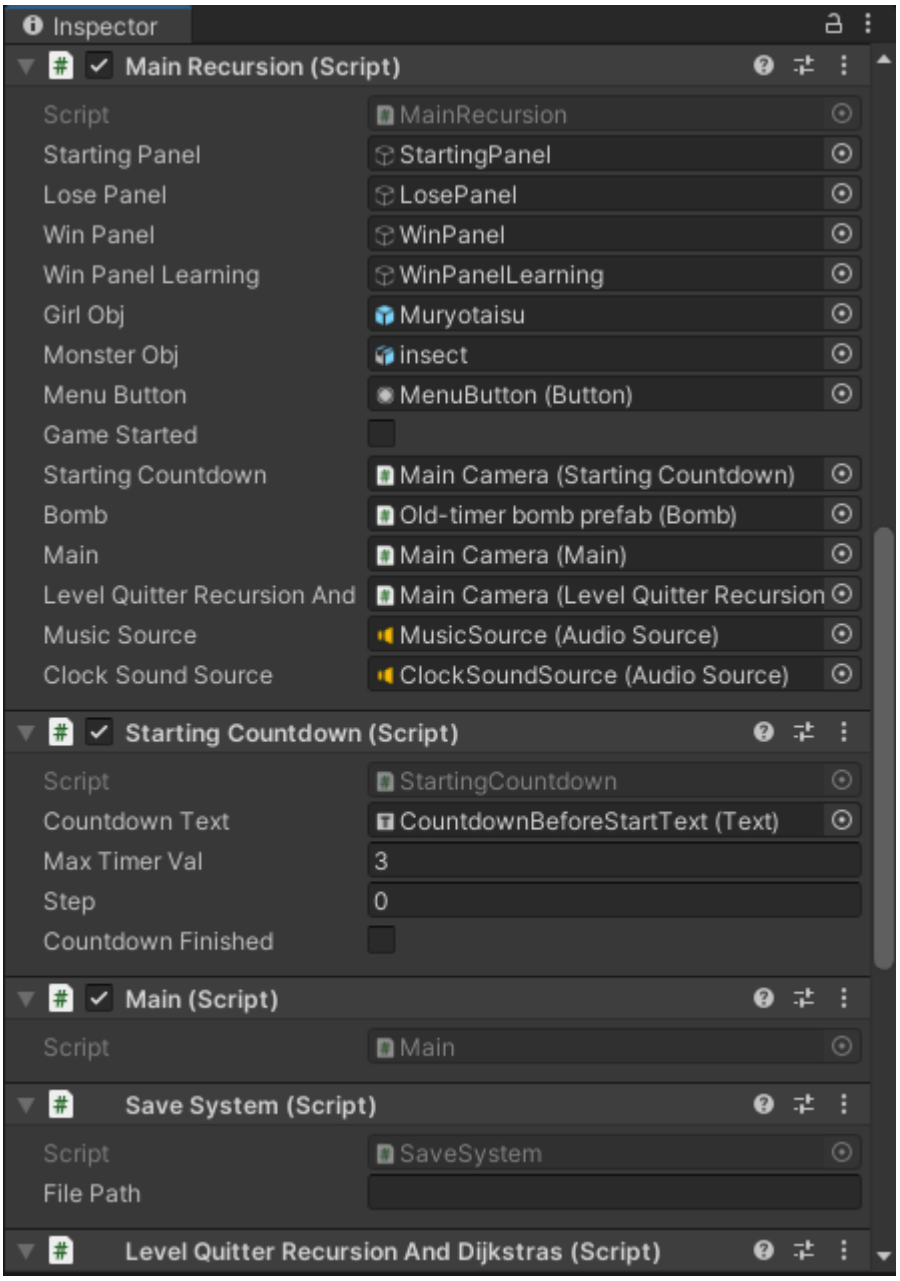

Рисунок Б.15 – Інспектор компонентів головної камери в сцені «Recursion»

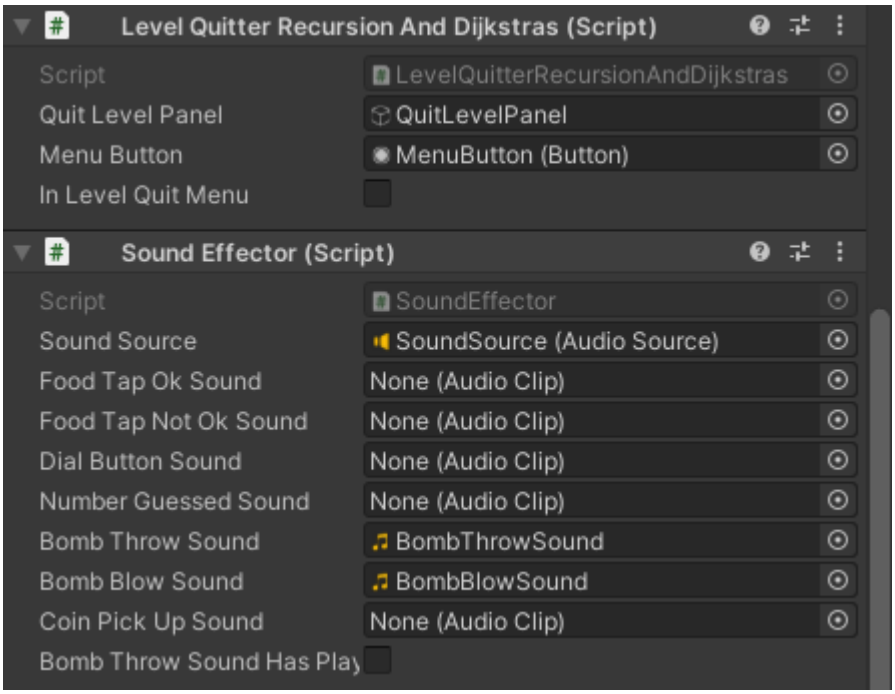

Рисунок Б.16 – Інспектор компонентів головної камери в сцені «Recursion» (продовження)

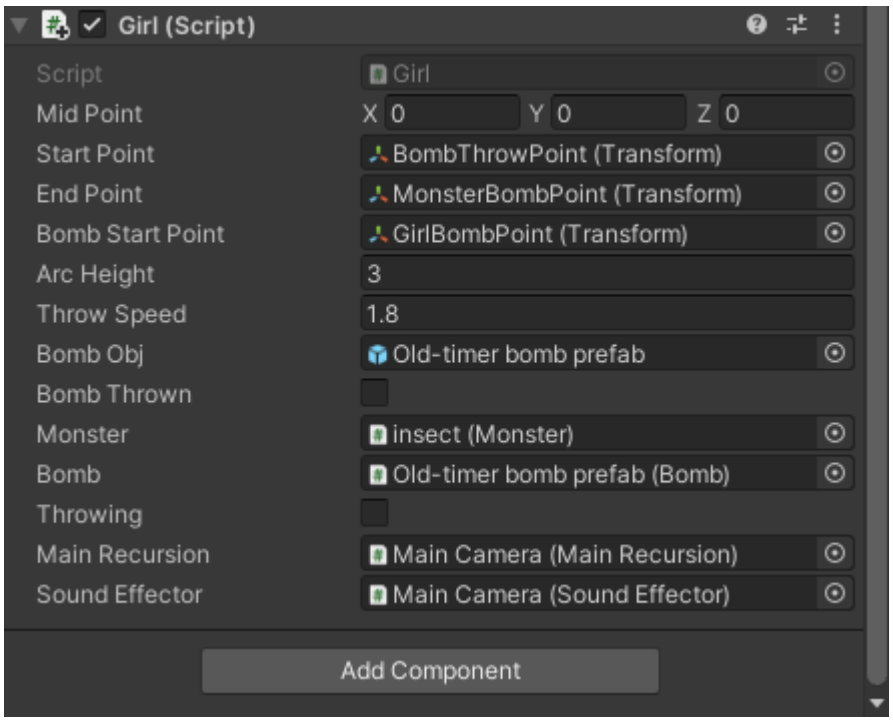

Рисунок Б.17 – Інспектор компонентів «Muryotaisu» в сцені «Recursion»

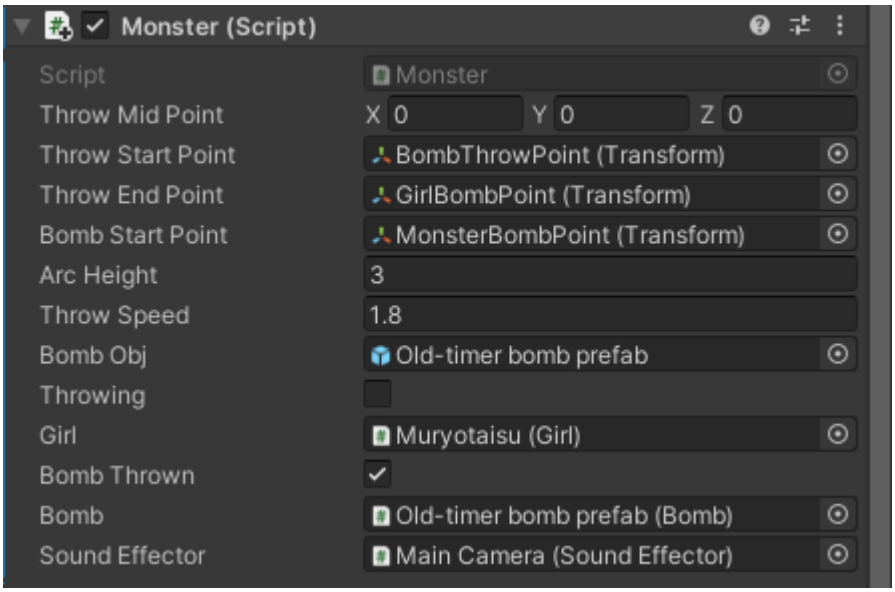

Рисунок Б.18 – Інспектор компонентів «insect» в сцені «Recursion»

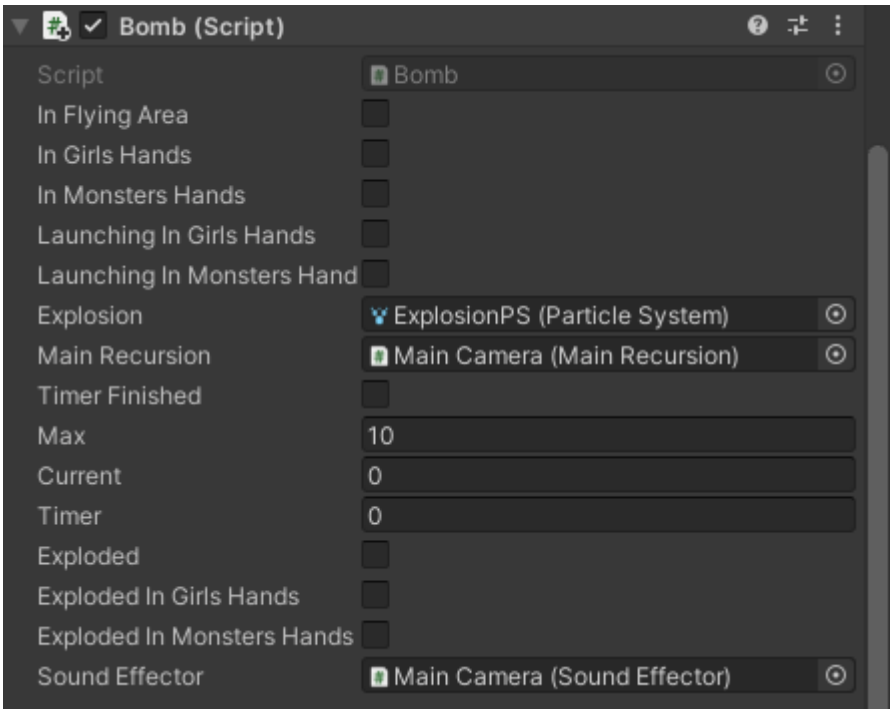

Рисунок Б.19 – Інспектор компонентів «Old-timer bomb prefab» в сцені «Recursion»

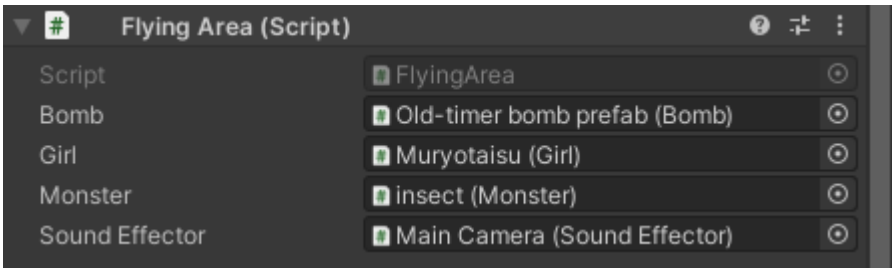

Рисунок Б.20 – Інспектор компонентів «FlyingArea» в сцені «Recursion»

Рисунок Б.21 – Ієрархія елементів в ігровій сцені «DijkstrasAlgorithm»

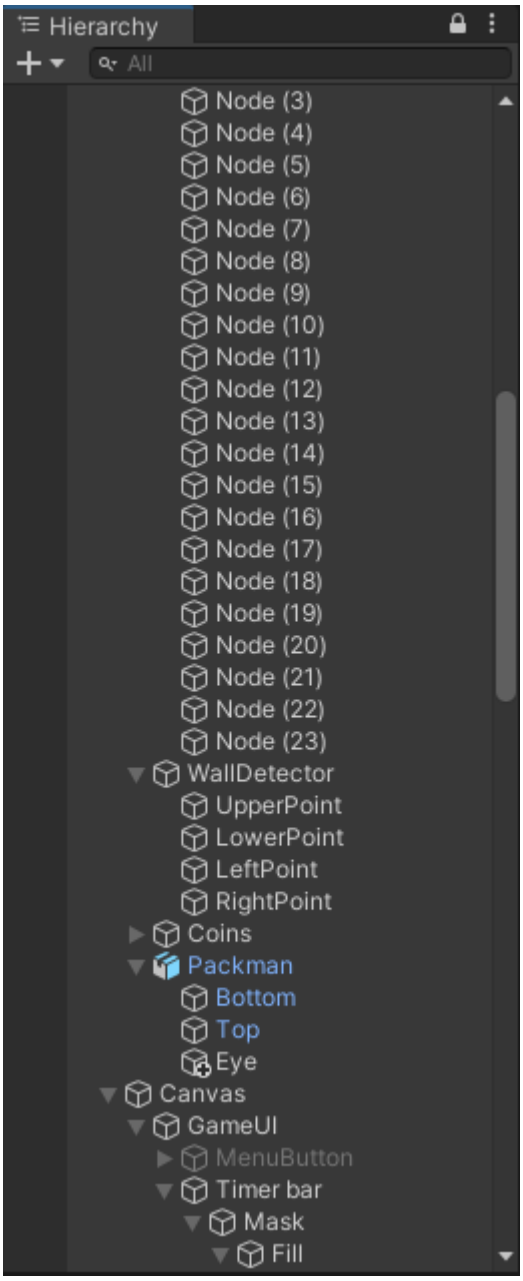

Рисунок Б.22 – Ієрархія елементів в ігровій сцені «DijkstrasAlgorithm» (продовження)

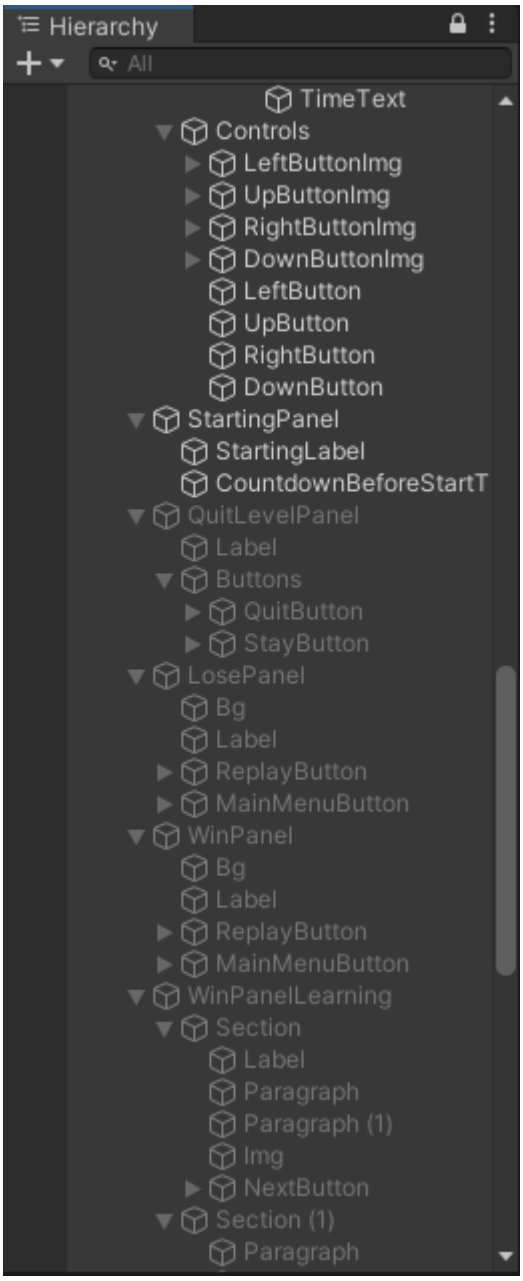

Рисунок Б.23 – Ієрархія елементів в ігровій сцені «DijkstrasAlgorithm» (продовження 2)

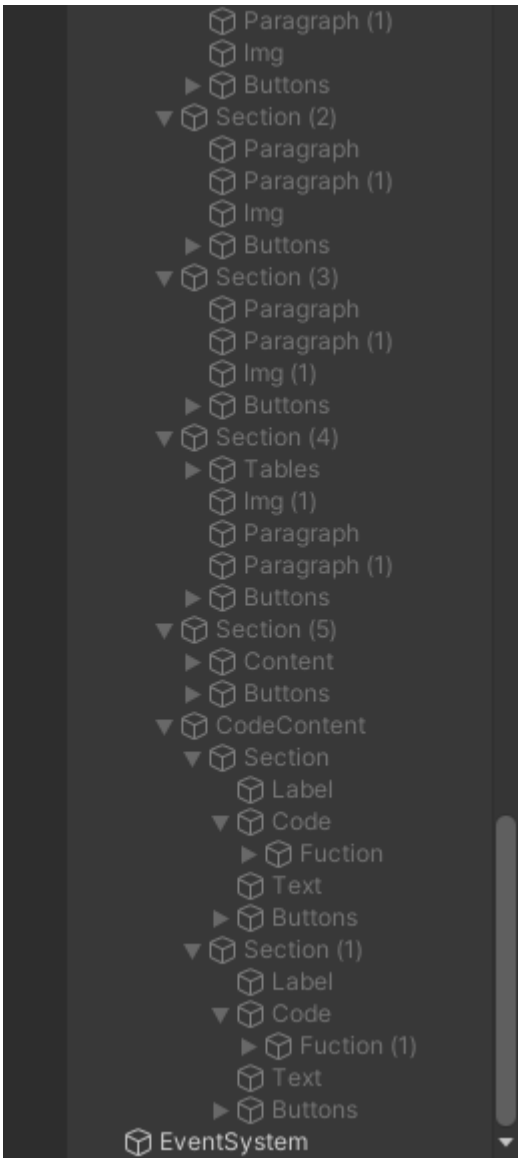

Рисунок Б.24 – Ієрархія елементів в ігровій сцені «DijkstrasAlgorithm» (продовження 3)

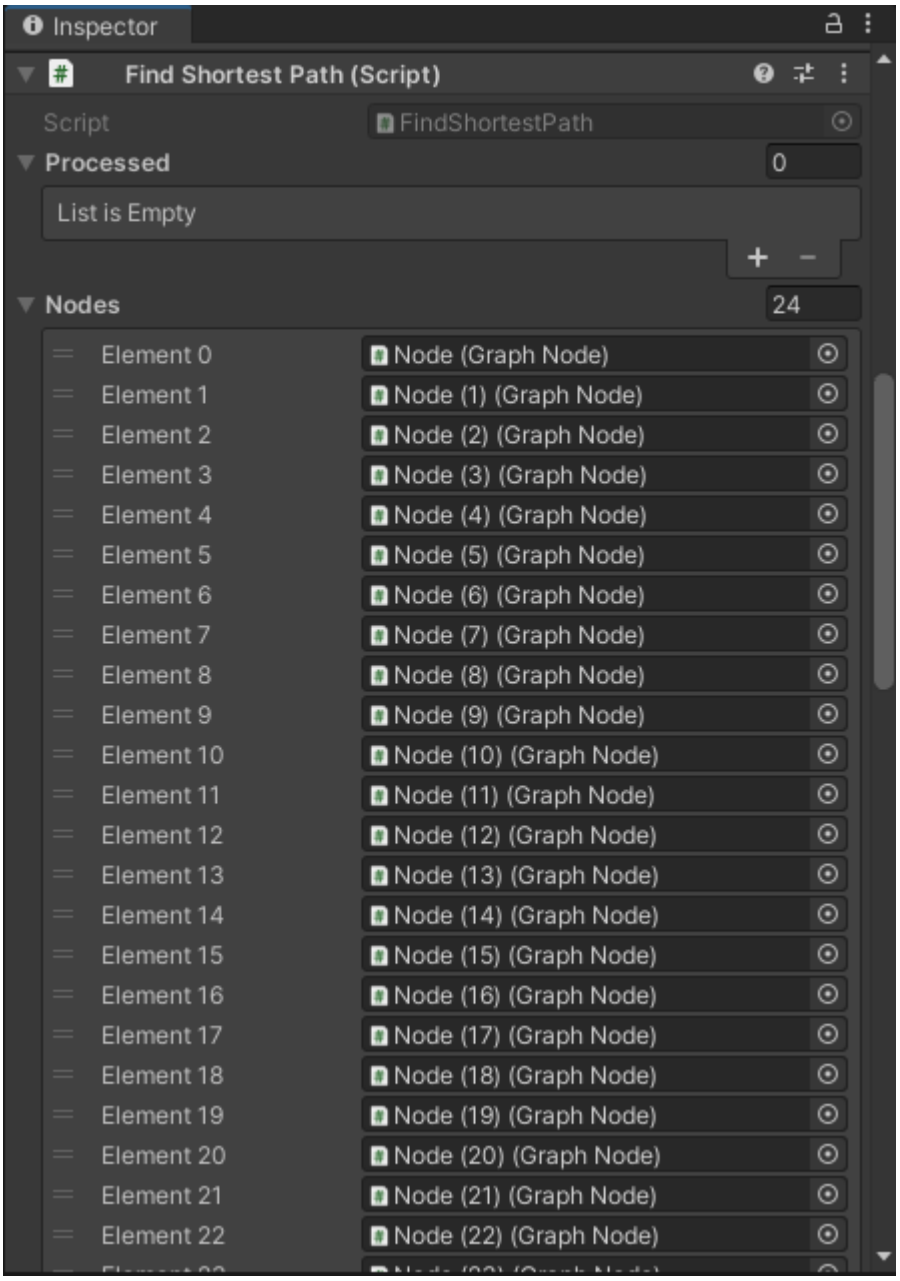

Рисунок Б.25 – Інспектор компонентів головної камери в сцені «DijkstrasAlgorithm»

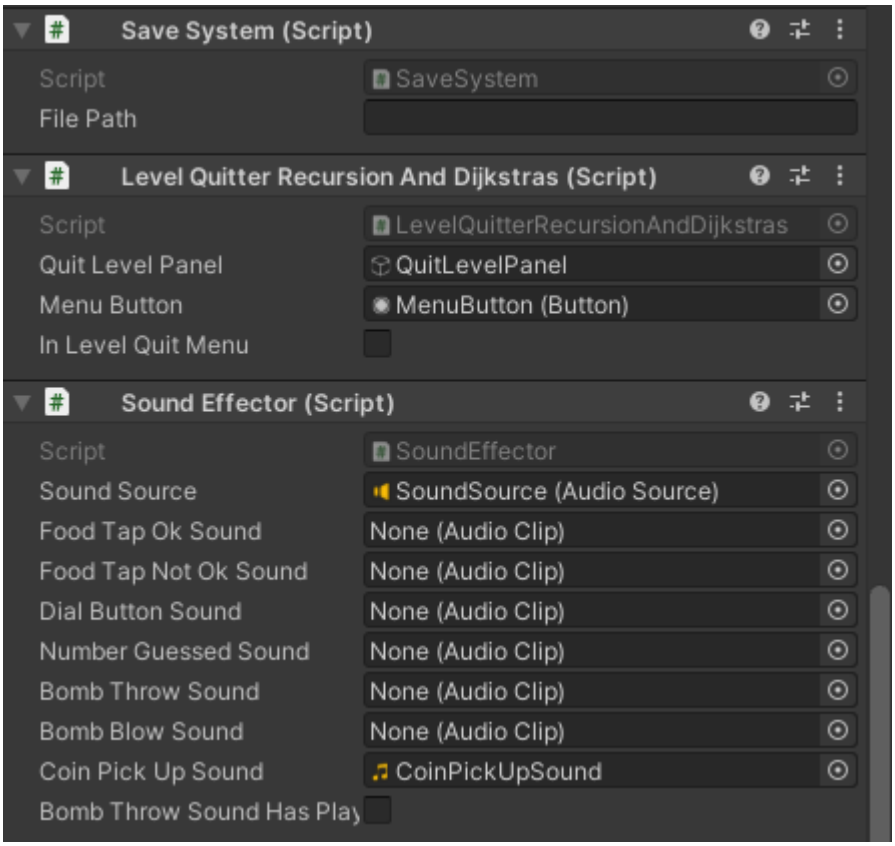

Рисунок Б.26 – Інспектор компонентів головної камери в сцені «DijkstrasAlgorithm» (продовження)

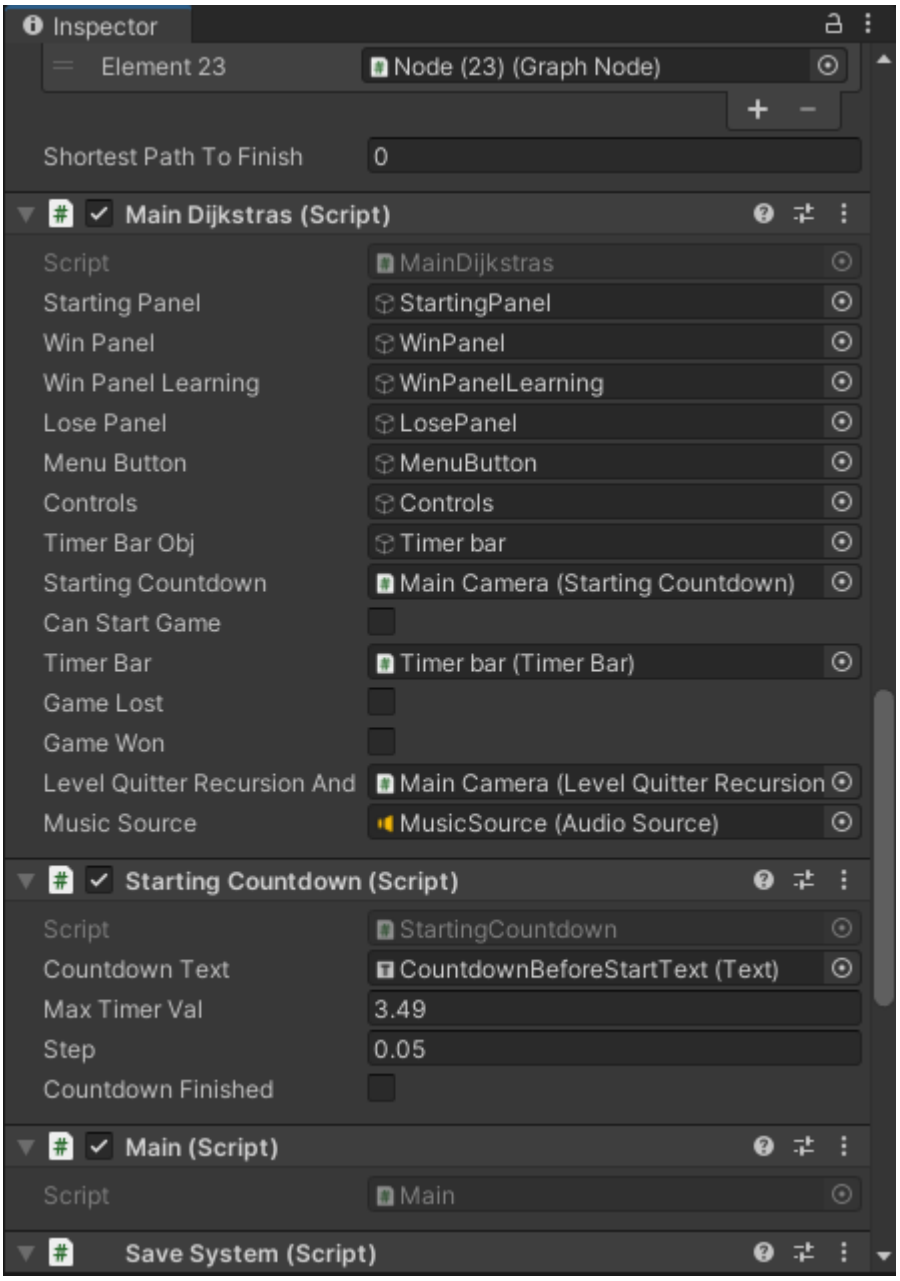

Рисунок Б.27 – Інспектор компонентів головної камери в сцені «DijkstrasAlgorithm» (продовження 2)

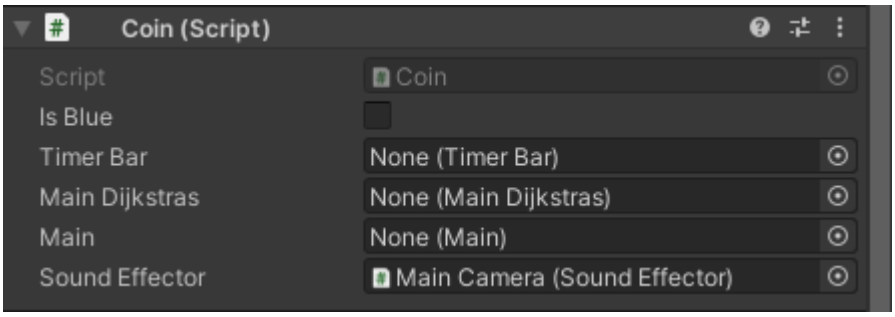

Рисунок Б.28 – Інспектор компонентів одного з об’єктів типу «Coin» в сцені «DijkstrasAlgorithm»

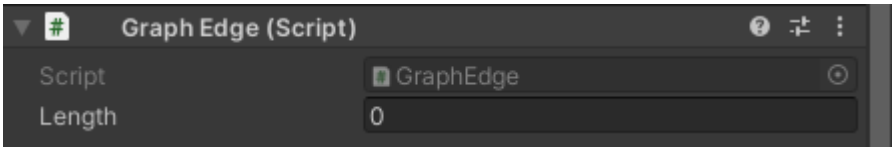

Рисунок Б.29 – Інспектор компонентів одного з об’єктів типу «GraphEdge» в сцені «DijkstrasAlgorithm»

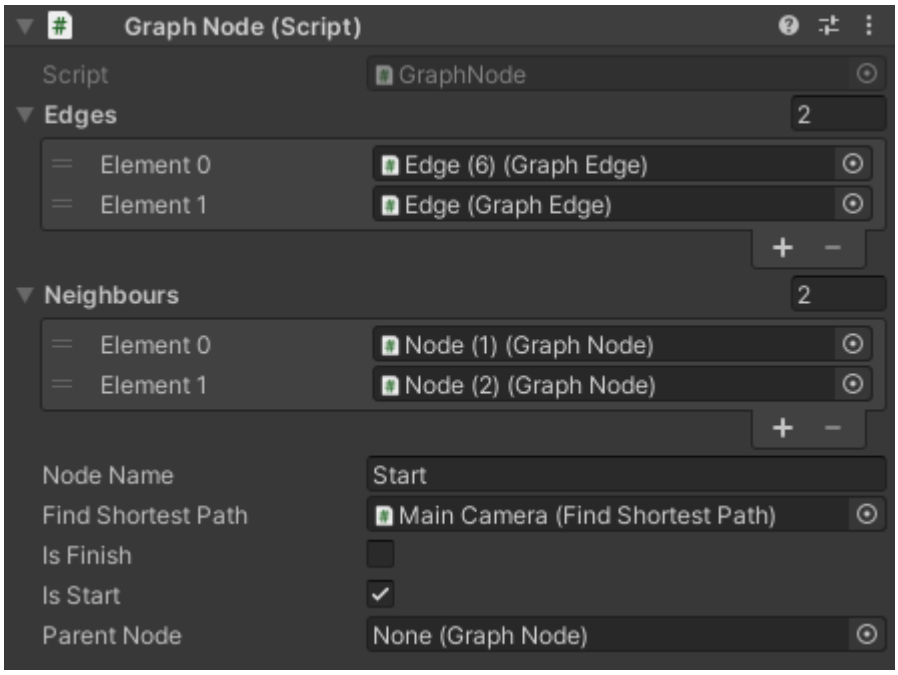

Рисунок Б.30 – Інспектор компонентів одного з об’єктів типу «GraphNode» в сцені «DijkstrasAlgorithm»

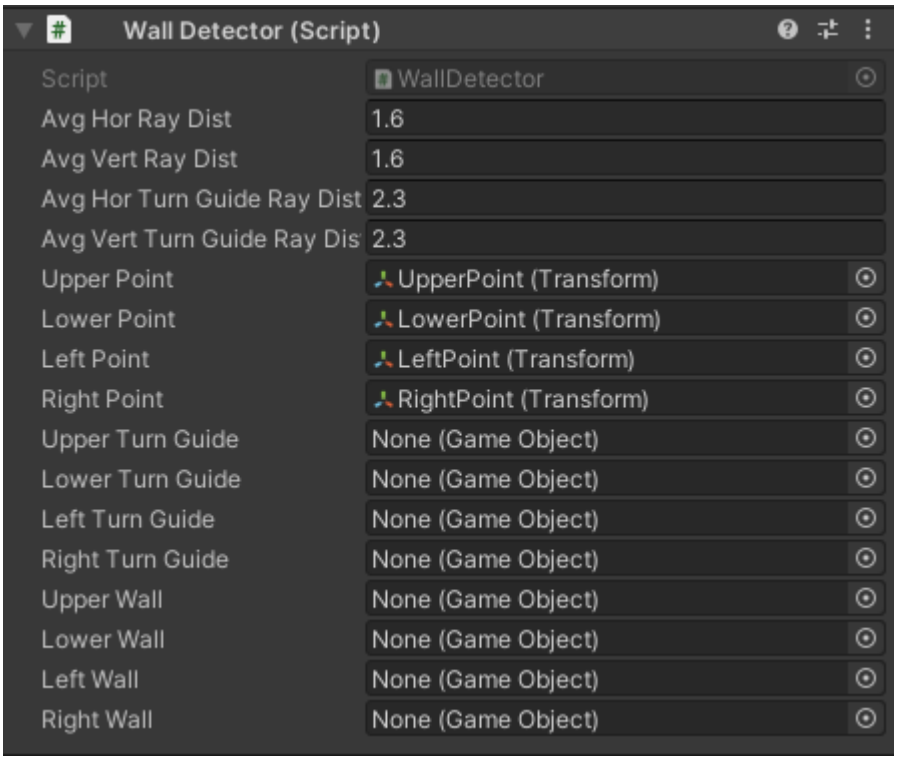

Рисунок Б.31 – Інспектор компонентів об’єкта «WallDetector» в сцені «DijkstrasAlgorithm»

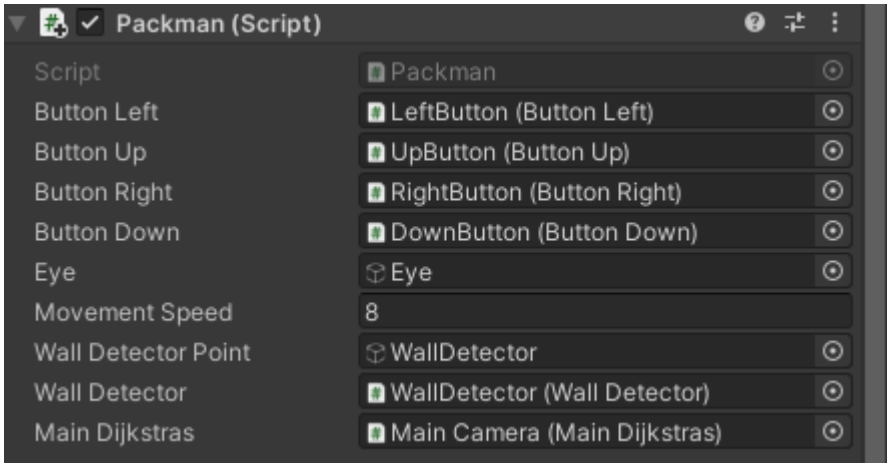

Рисунок Б.32 – Інспектор компонентів об’єкта «Packman» в сцені «DijkstrasAlgorithm»

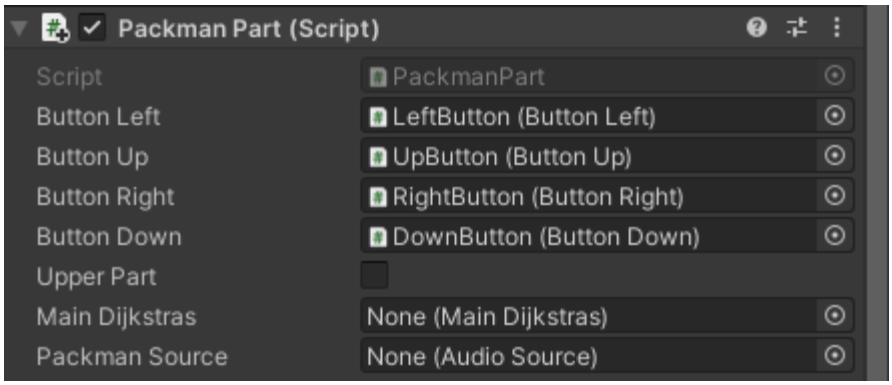

Рисунок Б.33 – Інспектор компонентів об’єкта «Bottom» в сцені «DijkstrasAlgorithm»

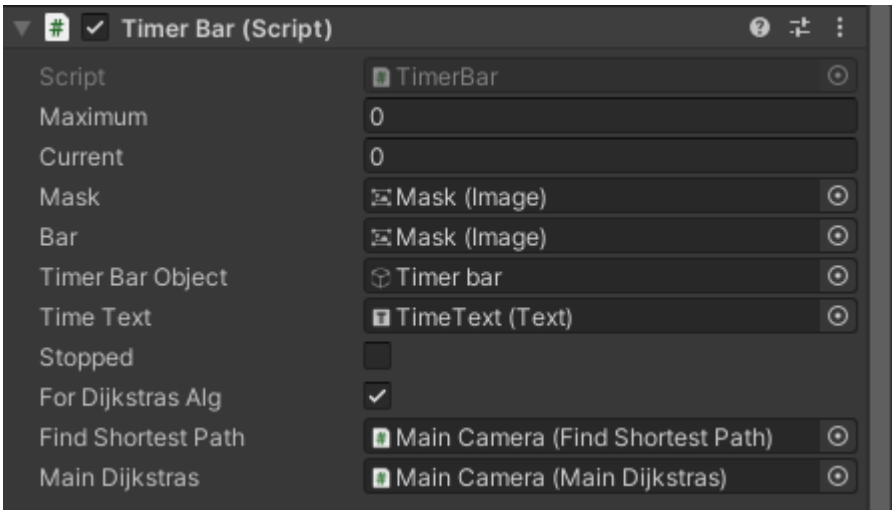

Рисунок Б.34 – Інспектор компонентів об’єкта «TimerBar» в сцені «DijkstrasAlgorithm»

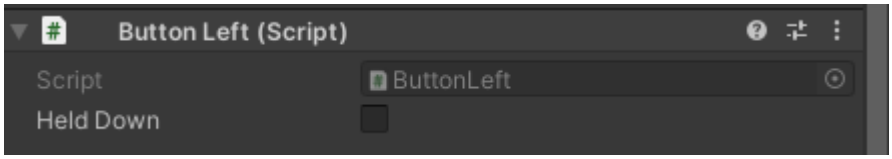

Рисунок Б.35 – Інспектор компонентів об’єкта «LeftButton» в сцені «DijkstrasAlgorithm»

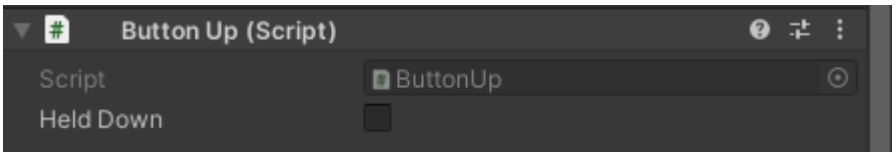

Рисунок Б.36 – Інспектор компонентів об’єкта «UpButton» в сцені «DijkstrasAlgorithm»

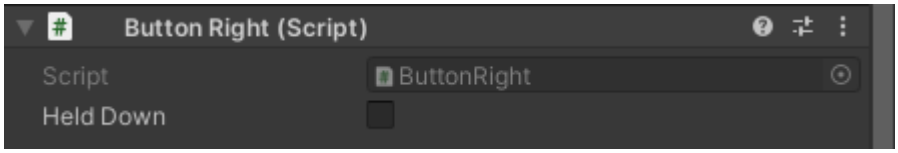

Рисунок Б.36 – Інспектор компонентів об’єкта «RightButton» в сцені «DijkstrasAlgorithm»

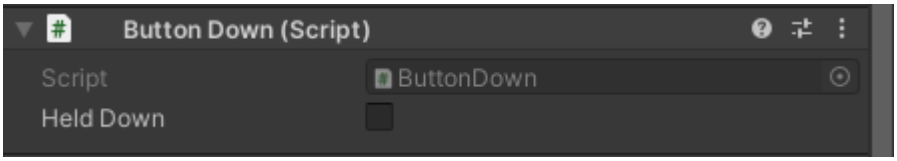

Рисунок Б.37 – Інспектор компонентів об’єкта «DownButton» в сцені «DijkstrasAlgorithm»

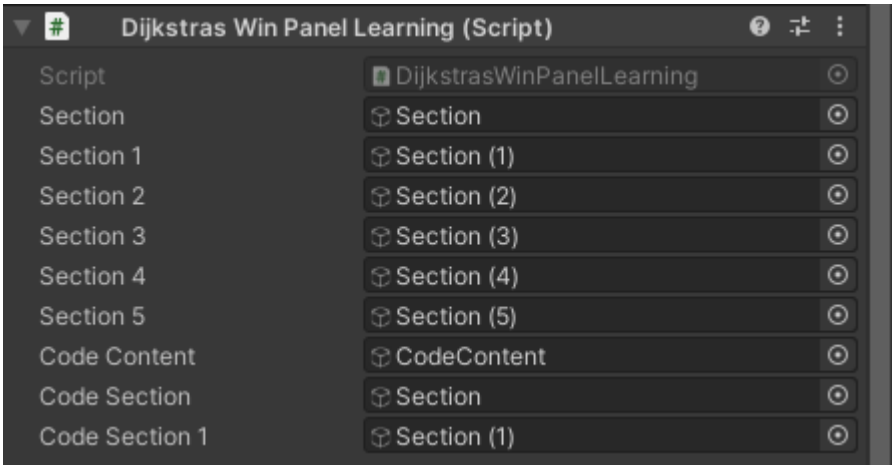

Рисунок Б.37 – Інспектор компонентів об’єкта «WinPanelLearning» в сцені «DijkstrasAlgorithm»