Міністерство освіти і науки України
Одеський національний технологічний університет
Кафедра комп'ютерної інженерії

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему *Розробка комп'ютерної гри жанру "головоломка"*
(назва кваліфікаційної роботи згідно наказу ОНТУ)

Здобувача *Беня Д. М.*
(прізвище, ініціали)

2 ск-2 курсу *КІ-543* групи

Керівники: *доц. Нєнов О.Л.*
(посада, прізвище та ініціали)
ст. викл. Слушна Н.В.
(посада, прізвище та ініціали)

Консультанти:
(посада, прізвище та ініціали)
Phd, ст. викл. Богданов О.О.
(посада, прізвище та ініціали)

Кваліфікаційна робота допускається до захисту

Рішення кафедри від *05.06* 2024 р., протокол № *8*

Завідувач кафедри комп. інженерії _____ Сергій АРТЕМЕНКО
(назва кафедри) (підпис) (Ім'я ПРІЗВИЩЕ)

Одеса - 2024 рік

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерної інженерії, програмування та кіберзахисту*
Кафедра *комп'ютерної інженерії*
Ступінь вищої освіти *бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*
Освітня програма *Розробка ігор та інтерактивних медіа у віртуальній реальності*

ЗАТВЕРДЖУЮ
Зав. кафедри *комп'ютерної інженерії*
Сергій АРТЕМЕНКО
« *30* » *серпня* 2023 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Беня Дмитра Миколайовича

1. Тема роботи *Розробка комп'ютерної гри жанру "головоломка"*

Затверджена наказом університету від « *30* » *серпня* 2023 р., наказ № *442-03*

2 Термін здачі здобувачем закінченої роботи *28 травня 2024 р.*

3. Вихідні дані роботи

Документація на середовище розробки ігрового рушія Unity

4. Перелік питань, які потрібно розробити

1. Вступ. 2. Передпроектний аналіз. 3. Проектування гри.
4. Реалізація гри. 5. Техніко-економічне обґрунтування.
6. Охорона праці. 7. Загальні висновки.

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

Слайд 1. Характеристика кваліфікаційної роботи. Слайд 2. Огляд подібних ігор на ринку. Слайд 3. Вимоги до програмного проекту. Слайд 4. Архітектура проекту. Слайд 6. Програмні алгоритми. Слайд 7. Елементи користувацького інтерфейсу. Слайд 8. Техніко-економічні показники. Слайд 9. Загальні висновки.

6. Консультанти по роботі, із зазначенням розділів роботи, що стосуються їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Економіка	*Phd, ст. викл. Богданов О.О.*		
Охорона праці	*к.т.н., доц. Нєнов О.Л.*		
Нормоконтроль	*ст. викл. Слушна Н.В.*		

7. Дата видачі завдання *07.05.2024*

Керівники *Олексій НЄНОВ*

Наталя СЛУШНА

Завдання прийняв до виконання *Дмитро БЕНЬ*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Дослідженні предметної області*	*26.10.2023*	
2.	*Дослідження існуючих аналогів*	*30.11.2023*	
3.	*Дослідження методів генерації лабіринту*	*28.01.2023*	
4.	*Проектування*	*15.02.2024*	
5.	*Розробка демонстраційної версії ПЗ*	*27.03.2024*	
6.	*Підготовка техніко-економічної частини*	*15.04.2024*	
7.	*Підготовка розділу охорони праці*	*15.04.2024*	
8.	*Оформлення пояснювальної записки*	*27.05.2024*	
9.	*Оформлення графічної частини та лістингу*	*27.05.2024*	

Здобувач-дипломник _____ *Дмитро БЕНЬ*

Керівники роботи _____ *Олексій НЄНОВ*

_____ *Наталя СЛУШНА*

Несу відповідальність за ідентичність електронного та друкованого варіантів кваліфікаційної роботи, даю згоду на обробку персональних даних та не заперечую проти розміщення кваліфікаційної роботи на офіційних web-ресурсах ОНТУ.

Підтверджую, що в кваліфікаційній роботі відсутні порушення норм академічної доброчесності.

Здобувач-дипломник *Дмитро БЕНЬ* _____

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці гри у жанрі «головоломка», а саме – лабіринт. Ігри у цьому жанрі допомагають людям проводити свій вільний час із користю для розвитку своїх розумових здібностей. За допомогою цієї гри людина зможе у будь-який час присвятити пару вільних хвилин розвитку своєї спостережливості та кмітливості.

В першому розділі висвітлені особливості жанру «головоломка», проведено дослідження об’єкту проектування, розглянуті варіанти існуючих рішень.

В другому розділі описані основні етапи проектування гри: визначення необхідних алгоритмів, основних об’єктів і прототипів екранів гри для 2D-простору.

У третьому розділі описано процес реалізації гри у рушії Unity, користувацький інтерфейс гри, деякі результати тестування.

Четвертий розділ обґрунтовує економічну вигоду від застосування одержаних результатів.

У п’ятому розділі розглянуто питання охорони праці.

Результатом роботи є проект і тестовий застосунок гри, зібраний для платформи Windows.

Ключові слова: *гра, головоломка, лабіринт, Unity, C#*.

ABSTRACT

The qualification work is devoted to the development of a game in the genre of "puzzle" – a maze. Games in this genre help people spend their free time with the benefit of developing their mental abilities. With the help of this game, a person will be able to devote a couple of free minutes at any time to the development of his observation and intelligence.

In the first chapter, the peculiarities of the "puzzle" genre were identified, the design object was researched, and options for existing solutions were considered.

The second chapter describes the main stages of game design: determination of necessary algorithms, main objects and prototypes of game screens for 2D space.

The third chapter describes the process of implementing the game in the Unity engine, the user interface of the game, and some test results.

The fourth section substantiates the economic benefit from the application of the obtained results.

The fifth chapter deals with the issue of labor protection.

The result of the work is a project and test application of the game, compiled for the Windows platform.

Keywords: *game, puzzle, maze, Unity, C#*.

ЗМІСТ

стор.

ВСТУП

За роки незалежності України ігрова галузь продемонструвала свою життєздатність та перспективність. Так, за останнє десятиліття кількість компаній-розробників ігор збільшилася з 20 до більше ста, а загальна аудиторія їхніх продуктів становить на сьогодні більше 800 мільйонів користувачів. Загальний дохід української ігрової індустрії перевищує 200 мільйонів доларів США. Лише з 2014 по 2018 рік дохід індустрії зріс майже на 60%. А сьогодні Україна стала однією з 50 найприбутковіших країн у галузі відеоігор у світі. Таким чином, незважаючи на складні умови сьогодення, ігрова індустрія України продовжує розвиватися.

Незважаючи на наявність сьогодні на ринку комп'ютерних ігор безлічі головоломок, потреба у створенні нових ігор цього жанру не зменшується, а навпаки, збільшується. Головоломкові ігри залишаються незмінно популярними та здатними привертати увагу гравців будь-якого віку. Ці ігри надають можливість не просто розважити свій мозок, а й розвинути креативність та логічне мислення. Головоломки можуть зустрічатися як у вигляді самостійних проектів, так і входити до складу компіляцій ігор або бути частиною ігрових платформ. Багато людей продовжують цінувати головоломкові ігри за виклики, які вони створюють для їх когнітивних здатностей, та за можливість підвищити рівень їх інтелектуального розвитку.

Сучасна ігрова індустрія надає безліч майданчиків для реалізації та трансформації оригінальних ігор на базі різних платформ, в тому числі з використанням віртуальної і доповненої реальності, багатокористувацького режиму тощо. Аналізуючи сучасний ринок ігор-головоломок, можна дійти висновку, що жанр комп'ютерних ігор «головоломка», пройшовши довгий шлях становлення, є дуже популярним й нині. За прогнозами, цей жанр має перспективи у розвитку та використанні нових платформ для реалізації та можливості отримання фінансової вигоди.

Таким чином, розробка комп’ютерних ігор-головоломок залишається, без сумніву, дуже актуальною сьогодні.

Об’єктом дослідження в даній роботі є узагальнений процес розробки комп’ютерних логічних ігор-головоломок.

Предметом дослідження є принципи і методи розробки проекту комп’ютерної гри в жанрі «головоломка».

Метою даної роботи є розробка проекту і елементи реалізації комп’ютерної гри в жанрі «головоломка».

Для здійснення поставленої мети в роботі вирішуються такі завдання:

- узагальнена постановка завдання;
- аналітичний огляд ринку комп’ютерних ігор-головоломок;
- розробка технічного завдання на розробку;
- створення проекту комп’ютерної гри-головоломки обраного виду;
- реалізація проекту у тестовий прототип застосунку;
- техніко-економічне обґрунтування розробки;
- дослідження суміжних питань охорони праці;
- оформлення пояснювальної записки та підготування презентації.

РОЗДІЛ 1
ПЕРЕДПРОЕКТНИЙ АНАЛІЗ

1.1 Огляд предметної області комп'ютерних ігор-головоломок і обґрунтування доцільності розробки

1.1.1 Розвиток жанру ігор-головоломок

Галузь відеоігор сьогодні є одним з глобальних секторів розваг і однією з найбільш динамічних серед комп'ютерних технологій взагалі. Комп'ютерні ігри набувають статусу культурного явища і визнаються справжніми творами мистецтва. Кількість гравців у комп'ютерні ігри постійно зростає: за даними Newzoo, 2023 року кількість геймерів у світі сягнула 3,3 млрд (у 2020 році налічувалось приблизно 2,5 мільярда гравців). Геймінг і геймдев продовжують залишатися досить високооплачуваною галуззю. Утворюється справжня потужна екосистема, яка складається з розробників і видавців ігор, профільних засобів масової інформації, спеціальних фінансових і рекламних інструментів, а також спеціалізованих ігрових майданчиків. Крім того, на ринку ігор активно працюють виробники електроніки, такі як Apple, інтернет-корпорації, такі як Google, які мають необхідні ресурси для успішного виведення перспективних ігрових рішень у світ.

Одним з вельми популярних жанрів комп'ютерних ігор є головоломки. Згідно з Вікіпедією, комп'ютерні ігри-головоломки складають широкий жанр відеоігор, в яких робиться акцент на рішення різноманітних логічних і концептуальних завдань з використанням тих або інших компетенцій стосовно вирішення проблем, включаючи логіку, розпізнавання образів, виявлення послідовностей, просторове розпізнавання, аналіз слів тощо [4]. Хоча багато екшн-ігор і пригодницьких ігор мають головоломні елементи у

своєму дизайні рівнів, справжня гра-головоломка зосереджується на вирішенні головоломок як на головному аспекті геймплею.

Ігри-головоломки мають стародавнє коріння, яке сягає глибини історії. Початкові логічні завдання можна відстежити на стінах єгипетських пірамід, в давньогрецьких манускриптах та в інших історичних документах. Величезної популярності ця розвага набула в IX столітті, коли рівень освіти серед населення стрімко зростав. У цей період було видано перше в історії Європи зібрання головоломок, яке під назвою «Задачі для розвитку молодого розуму» представив ірландський просвітитель Алкуїн.

На початку XIX і XX століть значний внесок у популяризацію головоломок зробили американець Сем Лойд та англієць Генрі Дьюден. Їхня діяльність сприяла тому, що ці розумові ігри потрапили до багатьох видань і стали популярними серед різних соціальних шарів.

Наступним значним етапом у розвитку головоломок став винахід угорським ерудитом Ерне Рубіком у 1974 році відомого кубика Рубіка. Гра з цією головоломкою допомагала розвивати в людей навички розв’язування нестандартних та складних завдань.

До цього часу була створена відповідна технічна база для перенесення головоломок з фізичного світу у віртуальний. Перша комп'ютерна рольова гра «Пригоди» з'явилася в Стенфорді у 1960-х роках. Гравець, використовуючи короткі фрази, міг взаємодіяти з комп'ютером, керувати героєм через різні сценарії і розв'язувати головоломки. Одним із перших яскравих прикладів цього жанру є гра Sokoban, створена Хіроюкі Імабаясі у 1980 році, де гравець переміщує ящики по лабіринту з метою поставити їх на певні місця [9].

Справжнім витвором мистецтва серед комп'ютерних головоломок стала гра «Тетріс». Вона була випущена в 1985 році і відзначилася простим, але захоплюючим геймплеєм. У 1987 році з'явилася ще одна легендарна головоломка – «Сапер», яка завдяки своїй простоті та нескладному геймплею стала улюбленою для багатьох поколінь і є одним з класичних представників жанру.

Поява тривимірної графіки сприяла розвитку головоломок на ігрових консолях. В самому початку 1990-х років ігри «The Lost Vikings» та «Lemmings» внесли у жанр свіжий струмінь. Після цього відносно недорогі ігри-головоломки стали вельми популярними на портативних ігрових консолях.

У 2000 році ігри «Pikmin», «Meteos», «Polarium», «LocoRoco» і «Lumines» знову привернули увагу до принципів жанру. А серія головоломок «Brain Training» викликала захоплення в ігровій індустрії Японії у 2005 році. Ця гра поєднувала широкий спектр завдань, захоплюючи увагу гравців на тривалий час, одночасно розвиваючи їх критичний аналіз та логічне мислення.

У 2016 році вийшла гра «The Witness», яка успішно поєднала головоломку з безліччю варіацій у відкритому світі. Гра була досить складною, що стало випробуванням для гравців, але вона окупилася всього за тиждень.

Однією з найвідоміших фізичних головоломок нового покоління є «Angry Birds». Досить відомими сучасними представниками в категорії ігор-головоломок також є «Q.U.B.E. 2», «The Fall Part 2: Unbound», та «Flipping Death».

Популярність жанру головоломок на ігровому ринку зазнала змін у різні періоди. Перший значний пік інтересу був у 1983 році, коли стандартом для комп'ютерних мереж став протокол TCP/IP, що «відкрило двері» до сучасного Інтернету. Цей період став можливістю для ігрової індустрії розширити свої горизонти та розвивати нові проекти. Розширення потенціалу галузі призвело до зростання продажів ігрових продуктів, зокрема головоломок.

З 1989 по 1999 рік, компанія Nintendo активно рекламувала свою нову консоль Nintendo Game Boy в Японії, Європі і США. У той же період на медіа-ринку також здобули популярність Atari Lynx і Sega Game Gear. Однак, відразу ж після цього спостерігалося зниження попиту на ігри, що було пов'язано з високою вартістю консолей.

На початку 2000-х років ціни на консолі знизилися, що призвело до нової динаміки в індустрії. У період від 2000 до 2010 року спостерігався повторний виток популярності ігор-головоломок. Саме в 2000-х роках Інтернет став основним світовим простором для культурної, інформаційної та ігрової діяльності людства.

Отже, можна зробити висновок, що жанр комп'ютерних ігор «головоломка» пройшов досить довгий шлях становлення і залишається популярним до цього дня. У цьому жанрі є потенціал для подальшого розвитку

та використання нових платформ, що відкриває можливість отримання фінансової вигоди.

1.1.2 Класифікація ігор-головоломок

Світ ігор-головоломок захоплює своїм розмаїттям, пропонуючи гравцям випробувати логіку, кмітливість, просторову уяву та фізичні навички. Кожен жанр та піджанр має свої унікальні особливості, що робить його цікавим для певної аудиторії.

Класифікація ігор-головоломок за структурою:

1. Ігри з випадковим набором елементів. Гравцеві пропонується хаос з блоків або частин, які необхідно правильно організувати. Приклади: Тетріс, Bejeweled, Candy Crush.
2. Ігри з фіксованою структурою. Перед гравцем постає ігрова дошка або поле з елементами, де необхідно знайти рішення головоломки, досягнувши певної мети. Приклади: Шахи, Судоку, Хід коня.

Піджанри ігор-головоломок:

1. Логічні головоломки. Вирішення завдань тут ґрунтується на логіці та дедуктивному мисленні. Ці ігри тренують розум, змушуючи шукати нестандартні підходи. Приклади: «Маджонг», «Хід коня», «Реверсі».

2. Фізичні ігри. Гравцеві необхідно використовувати закони фізики ігрового середовища для вирішення завдань. Ці ігри можуть бути як простими, так і складними, пропонуючи різні рівні складності. Приклади: «The Splatters», «Portal», «Angry Birds».
3. Словесні ігри. Використовуються слова, фрази та правила мови для вирішення завдань. Ці ігри розвивають словниковий запас, логіку та креативність. Приклади: «Скрабл», ребуси, кросворди.
4. Візуальні ігри. Завдання вирішуються за допомогою візуальних елементів, таких як зображення, символи або геометричні фігури. Ці ігри розвивають просторову уяву, логіку та спостережливість. Приклади: «Маджонг», піксель-арт головоломки, ігри з пошуком предметів.
5. Пригодницькі ігри. Поєднують в собі елементи головоломки з сюжетом та дослідженням ігрового світу. Гравцеві необхідно не лише вирішувати задачі, але й просуватися по сюжету, розкриваючи таємниці та шукаючи вихід з лабіринтів. Приклади: «Room Escape Artist», «Myst», «The Witness».
6. Ігри-головоломки з кодуванням. Ці ігри кидають виклик гравцям, змушуючи їх використовувати елементи програмування для вирішення завдань. Вони поєднують логічне мислення з основами кодування, пропонуючи унікальний та інтелектуальний досвід. Прикладами таких ігор є «The Incredible Machine», «SpaceChem» та «Infinifactory».
7. Дослідницькі ігри-головоломки. У таких іграх-головоломках гравці використовують метод проб і помилок, щоб просуватися вперед. Цей піджанр включає point-n-click ігри, які часто схожі на пригодницькі ігри та симулятори ходьби. На відміну від логічних головоломок, дослідницькі ігри зазвичай вимагають індуктивного мислення для вирішення завдань. Відмінною рисою цих ігор є те, що гравець має експериментувати з механізмами на кожному рівні, перш

ніж знайти рішення. Елементи головоломки часто не є очевидними протягом всієї гри, тому для їх розкриття потрібні дослідження, вигадки та перевірки. До популярних дослідницьких ігор-головоломок належать «Myst», «Limbo», «The Dig», «Monument Valley», а також «кімнатні ескейпи», такі як «The Room».

8. Ігри в пошук предметів, або ігри з прихованими об'єктами. Цей жанр відеоігри-головоломки занурює гравців у світ детективних розслідувань або пошуку скарбів. Їх суть полягає в тому, щоб знайти перелік предметів, які хитро замасковані на мальовничих сценах. Даний жанр став надзвичайно популярним серед шанувальників казуальних ігор, пропонуючи їм розслабляючий та захоплюючий досвід. Представники: «Mother Goose: Hidden Pictures» (1991), випущена для CD-i – стала першою грою, яка офіційно започаткувала жанр «пошук предметів», «I Spy», яка також внесла значний вклад у розвиток жанру, «Awakening», «Antique Road Trip», «Dream Chronicles» та інші.
9. Ігри-головоломки з розкриттям зображень. Ці ігри вимагають спостережливості та логіки від гравця. Поступово розкриваючи фрагменти зображення, він має здогадатися, що воно приховує. Цей жанр тренує розум та дарує задоволення від розгадки таємниці.
10. Ігри-головоломки із зіставленням плиток. Суть гри полягає в маніпулюванні плитками на ігровому полі, щоб усунути їх згідно з певними правилами. Прародителем жанру є гра «Chain Shot!» (1985), яка започаткувала механіку зіставлення плиток, подібну до таких ігор, як Тетріс. Різновиди ігор із зіставленням плиток:

- заміна місцями: «Bejeweled», «Candy Crush Saga» – ігри, де треба міняти місцями плитки, щоб створити комбінації;
- класичний маджонг: «Mahjong Trails» – ігри, адаптовані з класичного маджонгу, де потрібно знаходити пари ідентичних плиток;

- стрілянина по плитках: «Zuma» – ігри, де треба стріляти плитками по ігровому полю, щоб усунути їх;
- тетрісоподібні рівні: багато сучасних ігор із зіставленням плиток містять рівні, схожі на Тетріс, де мета – розмістити певні плитки так, щоб вони прилягали одна до одної;
- ігри на збіг трьох: «Threes», «Lumines» та інші – ігри, де для усунення плиток потрібно з'єднати три або більше однакових елементів.

Світ цифрових розваг не оминув і класичні головоломки. Сьогодні можна насолодитися численними цифровими адаптаціями таких популярних ігор, як пасьянс та маджонг, на комп'ютері, смартфоні або планшеті. Крім того, багато знайомих словесних, числових та асоціативних головоломок, таких як кросворди, судоку та ребуси, теж знайшли своє місце в цифровому середовищі. Прикладом успішної адаптації є, зокрема, серія «Dr. Kawashima's Brain Training», яка покликана тренувати пам'ять, увагу та логічне мислення.

1.2 Вибір різновиду гри для проекту

При виборі конкретного виду гри-головоломки для проекту було розглянуто різноманітні піджанри з урахуванням цікавості для гравців і можливостей реалізації. Під час аналізу різних варіантів було визначено, що критеріям потенційно цікавого та занурюючого геймплею відповідає гра в жанрі «Лабіринт». Ігри з лабіринтами пропонують цікаві логічні завдання гравцям, вимагаючи від них використання логічного мислення та стратегічного планування для успішного знаходження шляху до виходу.

Таким чином, для реалізації було обрано гру в жанрі «Лабіринт» як різновиду ігор жанру «головоломка». Від проекту очікуватиметься захоплюючий ігровий процес, здатний розважити гравця та сприяти вдосконаленню його уважності, пам'яті і логіки.

1.3 Довідкові відомості про комп'ютерні ігри з лабіринтами

1.3.1 Загальний опис

Лабіринт (англ. *maze*) – це жанр комп'ютерних ігор, в яких успіх гравця залежить переважно від навігації та орієнтації в лабіринті. Визначення того, чи є щось лабіринтом, є відносним, але зазвичай можна визначити, чи вимагається від гравця навигаційне завдання в лабіринті на основі комбінації кімнат та коридорів.

Ігри-лабіринти можуть значно відрізнятися одна від одної як за ступенем наявності, власне, «лабіринту» у грі, так і за їх складністю. Можна, наприклад, порівняти складність лабіринтів у «Berzerk», «Pac-Man» і «Doom». Самі лабіринти можуть мати різний вигляд: зверху («Pac-Man»), з боку («Lode Runner»), з видом від першої особи («Doom»), або бути «прихованими» («Maze Craze»). У деяких випадках гравець може змінювати лабіринт, відкриваючи і закриваючи проходи («Mouse Trap»), або прогризаючи діри і створюючи тунелі («Lode Runner», «Dig Dug»).

Ігри з лабіринтами можуть поєднувати в собі елементи різних жанрів, що робить їх ще цікавішими та різноманітними:

- головоломка: пошук виходу з лабіринту в основному полягає в розв'язанні проблеми, що є ключовим елементом головоломок;
- пригода: дослідження лабіринту може бути захоплюючою пригодою, де гравцю доведеться зіткнутися з перешкодами, ворогами або головоломками у захоплюючому середовищі;
- аркада: ігри на пошук виходу з лабіринту можуть мати динамічний геймплей, що потребує швидкої реакції та спритності, що характерно для аркадних ігор;
- стратегія: у деяких лабіринтах гравцеві потрібно ретельно обдумувати маршрут та розв'язувати логічні задачі, що схоже на елементи стратегії;

- платформер: якщо гра на пошук виходу з лабіринту включає елементи стрибків, подолання перешкод і збір предметів, її можна віднести до жанру платформера.

Загалом, жанр гри на пошук виходу з лабіринту залежить від її конкретних механік, сюжету та цілей.

Деякі приклади:

- класичні ігри: «Pac-Man», «MazeRunner»;
- з елементами RPG: «Лабіринт Легенди»;
- з елементами екшену: «Tomb Raider: Останнє Відкриття»;
- з елементами жаху: «Silent Hill: P.T.»;
- логічні ігри: «Portal», «The Witness».

Деякі лабіринти менше орієнтовані на навігацію, а більше на те, якою послідовністю дій дістатися до різних місць ігрового світу («Lode Runner»). Часто завдання навігації і орієнтації виконується в умовах переслідування ворогами гравця, але це не обов'язково. Існують ігри, в яких лабіринти вбудовані в міні-ігри або є частиною гри. Наприклад, «Blue Labyrinth» у грі «Adventure» або підземний лабіринт «Selenetic Age» у грі «Myst».

Характерними прикладами ігор цього жанру є «Descent» (комбінація авіасимулятора та шутеру), «Dig Dug», «Doom» (шутер), «K. C. Munchkin», платформер «Lode Runner», «Maze Craze», «Mouse Trap»; «Pac-Man»; «Tunnel Runner»; пригодницька гра «Tunnels of Doom», «Spy vs. Spy», «Take the Money and Run».

1.3.2 Історія лабіринтних ігор

Перші лабіринтні ігри не були дуже складними. Наприклад, у пригодницьких іграх кількість локацій становила близько півсотні. Деякий час здійснювалися спроби збільшити їх кількість, але скоро стало зрозуміло, що такий підхід не дозволяє внести в гру щось особливо цікаве, і звичайною

практикою стало створення гри з об’ємом ігрового світу приблизно в сотню локацій. Замість збільшення розміру лабіринта розробники почали використовувати інші рішення. Наприклад, виділяти групу локацій у окремі зони, де ігровий світ мав загальні властивості. Локації в такій зоні можуть відрізнятися графічно від інших зон, це може бути однотипний опис у текстових іграх.

Для перших лабіринтних ігор характерний вид зверху, коли сам лабіринт цілком поміщається на екрані. У подальшому розробники ігор перемістили головного героя в центр екрана, а місцевість стала рухатися навколо нього шляхом її прокрутки. У середині 1990-х років стала більш популярною схема, при якій гравець бачив частину лабіринту своїм поглядом з першої особи. У той же час описані методи можуть поєднуватися, наприклад, коли одночасно відображається карта і поле зору персонажа. У цей же час з’явилися методи поєднання тривимірного зображення персонажа зверху з переміщенням персонажа в ізометричній проекції,
а рух персонажа забезпечується за рахунок прокручування екрану («The Immortal», «Dusk of the Gods», «Veil of Darkness»).

До 1990-х років гравцю зазвичай доводилося самостійно орієнтуватися в лабіринті, але поступово стандартним елементом стало автоматичне картографування. При цьому воно показувало не лише сам лабіринт (його карту), а й ті області, де головний герой перебував, а також додаткову інформацію, наприклад, знайдені написи.

1.3.3 Особливості

Особливості лабіринтів ігор не є суворими. Наприклад, у деяких рольових іграх обсяг ігрового світу може становити тисячі і навіть десятки тисяч локацій. Сам простір може бути організований по-різному – не лише як прямокутна сітка з осередків, а й, наприклад, може використовуватися

шестикутна карта. Герої ігор можуть подорожувати просто неба, в замках, печерах та інших варіантах сеттингу, і ігровий простір в них може розглядатися як лабіринт. При цьому найчастіше лабіринти у закритих приміщеннях більш складні у навігації.

Для ігор-рогаликів характерна процедурна генерація лабіринту для кожної нової гри, що підвищує їхню реграбельність, але це призводить до того, що найчастіше гравцям стає складно виграти. Прикладом такої гри є «NetHack» (рис. 1.1). Для ігор інших жанрів частіше використовується підхід фіксованого лабіринту, у якому різноманітність забезпечується розміщенням ворогів, ходом поєдинків, випадковими предметами, які гравець знаходить, і в цьому випадку з'являється можливість спроектувати гру так, щоб забезпечити її проходження.

Різні ігри можуть пропонувати лабіринти зі своїми особливостями. Так, для «Forgotten Realms» і «Wizardry» характерні вузькі прошарки між локаціями, а в «Might and Magic» і «Eye of the Beholder» в якості локацій виступають цілісні квадратні блоки.

Рис. 1.1 — Зовнішній вигляд гри *NetHack* у 2D та 3D

Щодо проектування ігор, дослідження лабіринту в іграх рідко є самоціллю, і найчастіше глибоко розроблені математичні принципи виявляються марними, оскільки в міру вдосконалення ігрового процесу часто доводиться їх порушувати. У той самий час повне дослідження

лабіринту не завжди виправдано, з іншого боку, гравець може пропустити закладені у нього важливі речі. Для вирішення цих проблем насамперед користувачеві надається карта та відповідні інструменти навігації, оскільки орієнтація по пам'яті може бути складною.

Для розширення лабіринту розробники можуть використовувати різні методи. Це можуть бути як локації з певною метою (ускладнення доступу до чогось цінного, пастка зі складним виходом тощо), так і більш великі обсяги ігрового світу з деякими особливостями. Наприклад, це може бути довга подорож з деякою кінцевою метою («довга дорога», «бездонна яма» тощо), де герой потрапляє до фактично одновимірної координатної сітки з виходом з іншого боку. Так, цей прийом часто застосовували розробники компанії Sierra Entertainment у своїх іграх «King's Quest V» та інших. Для орієнтації користувача при попаданні в таку зону гра може повідомляти про місцезнаходження в ній об'єктів (в «HeroQuest II» вказується відстань до мети в кількості екранів). Для полегшення подорожей героя можуть застосовуватися додаткові прийоми, наприклад, якщо гравець уже пройшов певний лабіринт, то гра дає можливість перемістити з початку в кінець без витрат часу.

Просте представлення ігрового світу з оглядом лабіринту зверху (як двовимірний вигляд у «Indiana Jones and the Fate of Atlantis» або ізометричний у «Conquests of the Longbow») призводить до того, що рішення стає досить простим, на рівні завдань з журналів для дітей. Для підвищення грабельності лабіринтної складової під час проектування вводяться додаткові складності: обмеження на час проходження, небезпеки на шляху тощо.

Деякі типи лабіринти є такими, що для них складно скласти карту, і це дає своєрідний виклик гравцю. Наприклад, якщо після проходження з одного екрана на інший спроба повернутися назад призводить до того, що гравець потрапляє не в початкову локацію («Fantastic Voyage»), або якщо екрани не вкладаються в прямокутну сітку («Zak McKracken and

the Alien Mindbenders», «Goody»). Такі лабіринти іноді називають квазі-лабіринтами, і орієнтація в них потребує своєрідних підходів. Проблемою може бути те, що подібна складність може призвести до того, що гравець вирішить, що він заблукав, хоча вихід може бути близько. Одним із рішень може стати надання предмета-компаса, який показує напрямок для виходу («The Secret of Monkey Island»).

Одним із способів побудови лабіринту є використання поверхів та сходів. Перші при цьому виступають як окремі рівні або зони, а останні є переходом між ними. Гра може будуватися так, що гравець постійно рухається вгору або вниз, а ігровий процес при цьому ускладнюється від рівня до рівня. Самі рівні можуть бути оформлені в окремі зони, в кожній з яких є свої особливості.

1.4 Огляд деяких подібних ігор на ринку

1.4.1 *Лабіринти: Maze Game*

«Лабіринти: Maze Game» – одна з популярних лабіринтних ігор для платформи *Android* від компанії *RV AppStudios*. Остання на сьогодні версія застосунку 1.4.3 від 14.12.2023 р. розміщена у магазині *Google Play* за адресою *https://play.google.com/store/apps/details?id=com.rvappstudios.maze.games.puzzle.mazes.labyrinth* (перша версія вийшла 10.08.2021 р.). Інсталяційний пакет важить 51 МБ, кількість скачувань перевищує 1 млн, середня оцінка від 5 тис. користувачів магазину *Google Play* складає 4,2 зіркі з п’ятьох. Гра є безкоштовною для завантаження і встановлення, містить ненав’язливу рекламу в додатку. Придатна для використання як на смартфоні, так і на планшеті під управлінням *Android* версії 4.4 і вище.

За описом від розробника, даний застосунок представляє собою захоплюючу сучасну версія класичної гри з лабіринтами. Гравцеві треба вести точку через лабіринт головоломки. Треба знайти спосіб вийти з

лабіринту, щоб пройти рівень. Гра поєднує в собі складні пазли з важкими для проходження іграми та забавними сюрпризами, в яку можна грати годинами.

Функції гри «Лабіринти: Maze Game»:

- більше 1500 безкоштовних головоломок;
- 4 захоплюючих лабіринту для гри;
- приголомшлива графіка та спокійна музика;
- чутливе управління змахуванням;
- підтримка багатокористувацького режиму гри в реальному часі;
- підтримка 25 мов інтерфейсу;
- підказки, які допоможуть, якщо невідомо, що робити далі;
- досягнення та бонуси для розблокування.

Приклади екранів гри показані на рис. 1.2.

За відгуками деяких з користувачів, гра може викликати нудьгу через відсутність складних завдань і високого ступеня автоматизації проходження, який зводить до мінімуму ручне керування процесом. Інші користувачі відмічають більшу цікавість змагального онлайн-режиму, приємний зовнішній вигляд гри, можливість налаштування кольорової палітри та відсутність нав'язливої реклами.

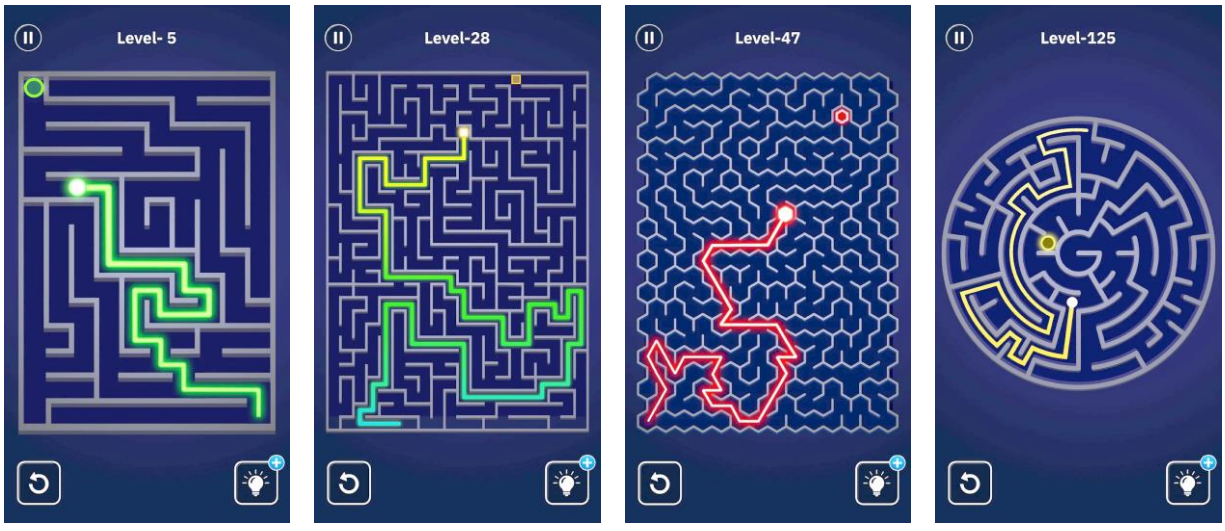

Рис. 1.2 — Екрани гри «Лабіринти: Maze Game»

В цілому, гра «Лабіринти: Maze Game» має інтуїтивно зрозумілий інтерфейс, просте управління, приємний дизайн, а також велику кількість лабіринтів-головоломок різних рівнів складності для початківців і більш досвідчених гравців. З опису слідує, що програма не виконує автоматичну генерацію рівнів, а працює з базою готових лабіринтів.

1.4.2 *«Maze: Path of Light»*

«Maze: Path of Light» (в українській версії магазину – *«Лабіринт: нескінченно простий*») – ще одна гра з лабіринтами для платформи *Android* від постачальника *Infinity Games, Lda.* Гра може бути встановлена також на Windows – для цього потрібна програма «Google Play Ігри (бета)». Гра розміщена в магазині застосунків *Google Play* за адресою *https://play.google.com/store/apps/details?id=com.rikudogames.maze*. Остання версія – 4.7.8 від 4.01.2024 р., дата першого випуску – 8.07.2020 р. Інсталяційний пакет важить 67 МБ (може залежати від конкретної версії *Android* і пристрою). Кількість встановлень перевищує 10 млн. За оцінками більш ніж 110 тисяч користувачів магазину *Google Play* середня оцінка застосунку – 4,3 з п’яти. Гра є безкоштовною для завантаження і встановлення, але містить рекламу і платний контент. Придатна для встановлення на пристроях під управлінням *Android* версій 6.0 і вище.

За описом від розробника, гра пропонує класичні квадратні лабіринти, а також багато інших оригінальних лабіринтів, включаючи круглі, трикутні, лабіринти особливих форм у вигляді кролика, гітари або дерева. Гра позиціонується як засіб для веселого проведення часу в пошуках виходу з головоломки. Підтримується 7 різних режимів гри, серед яких унікальні переплетені лабіринти з порталами, через які можна телепортуватися. В ході гри слід переміщатися від сузір’я до сузір’я та відкривати понад 100 комбінацій лабіринтів різних розмірів, типів, форм та режимів гри.

Ключові особливості:

– легкість грального процесу: треба проводити пальцем, щоб керувати світлом у лабіринті та шукати вихід;
– рівні складності: можна розслаблятися з маленькими лабіринтами або спробувати вирішити складніші;
– різноманітність рівнів: реалізація лабіринтів 4 типів: квадратних, трикутних, шестикутних та круглих.

У симуляторі лабіринту є кілька режимів гри:

1. Класичні лабіринти.
2. Обмеження за часом.
3. Обмеження за кількістю ходів.
4. Збір жетонів.
5. Плетені лабіринти з порталами для телепортації.

Гра містить понад 5 000 унікальних лабіринтів з яскравими кольорами та розслаблюючою музикою.

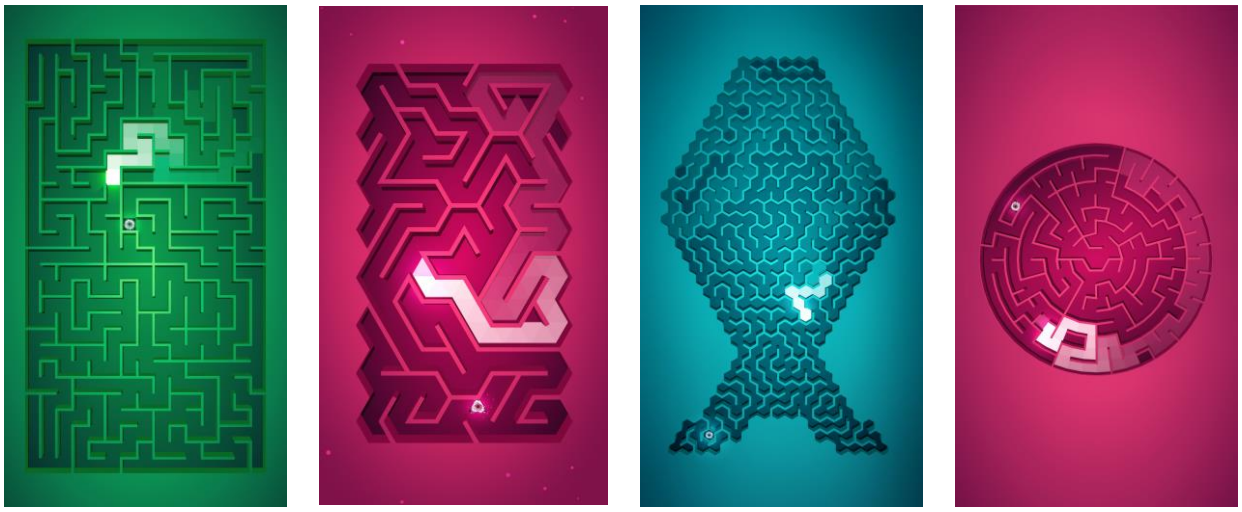

Рис. 1.3 — Приклади екранів гри *«Maze: Path of Light»*

Користувачі у відгуках відмічають приємну графіку та плавну анімацію, простоту рівнів, загальну приємність та м’якість ігрового процесу, ефект розслаблення. Також плюсом є ненав’язливість реклами, робота без обов’язкової наявності Інтернет-з’єднання, чуйність розробників.

Декому з користувачів не вистачає більш складних рівнів, наприклад, декількох шляхів до виходу, більш довгих глухих коридорів тощо.

Висновки до першого розділу

В першому розділі представлений аналітичний огляд предметної області ігор-головоломок, вибір різновиду гри для створення (лабіринт), а також аналіз деяких подібних ігор на ринку.

Виконана робота дозволяє перейти до подальшого проектування і реалізації гри у відповідності до завдання на дипломне проектування.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ГРИ

2.1 Концепт гри

Робоча назва гри: «Приборкувач лабіринтів», «Maze Tamer».

Жанр: казуальна головоломка.

Загальний опис гри.

«Приборкувач лабіринтів» – захоплююча гра-головоломка, де гравець має пройти лабіринт від поточного розташування до виходу. Гра кидає виклик логіці, просторовому мисленню та пам’яті гравця, адже з кожним успішним проходженням лабіринт ставатиме складнішим. Крім того, більш складні лабіринти перестають поміщатися на екрані, і треба запам’ятовувати, куди повернути, аби знайти вихід за мінімум кроків.

Ігровий процес та механіка гри.

Перед початком гри гравцеві пропонується ввести своє ім’я або обрати із списку збережених імен.

Ігровий рівень представляє собою двовимірну або тривимірну сцену з лабіринтом, який вміщується або повністю в екран, або лише його фрагмент. На початку кожного рівня лабіринт генерується випадковим чином, гарантуючи таким чином гравцеві унікальні враження при кожному проходженні гри з початку.

Гравець має знайти вихід із лабіринту, намагаючись використовуючи при цьому мінімум зайвих рухів. Після проходження лабіринту першого рівня генерується наступний, більш складний. Лабіринт третього рівня – ще більш складний та великий, так що він вже перестає поміщатися в екран. Коли гравець проходить лабіринтом ближче до краю екрану, камера зсуває лабіринт так, щоб відкрити його приховану область. Таким чином, гравцеві треба запам’ятовувати ту частину лабіринту, яка приховується при зсувах.

Для допомоги гравцю доступні підказки, які можна використовувати обмежену кількість разів.

Лабіринт може містити ізольовані області, так що спроба пройти його за правилом однієї руки може призвести до зациклення руху.

За проходження рівня нараховуються бали, які тим вище, чим менше рухів зробив гравець в пошуках виходу. Таким чином, гравці можуть змагатися між собою, збираючи бали та посідаючи позиції в таблиці лідерів.

Розвиток навичок: гра тренує логіку, просторове мислення та пам’ять гравця.

Цільова аудиторія.

Гра «Приборкувач лабіринтів» орієнтована на широку аудиторію будь-якого віку. Вона сподобається дітям, шанувальникам розваг, головоломок та логічних ігор, а також, людям, які прагнуть потренувати свої розумові здібності.

Візуальний стиль: мінімалістичний та стильний, кольори гармонійні, з використанням градієнтів. Загальний дизайн підкреслює атмосферу таємниці, не відволікаючи увагу гравця від головної мети – проходження лабіринту.

Звукове оформлення: спокійна музика і приємні на слух звукові ефекти, які створюють атмосферу таємниці, але не втомлюють увагу гравця одноманітністю.

Платформа.

Гру планується випустити, в першу чергу, на персональному комп’ютері (Windows). Починаючи з другої версії можливий випуск також на мобільних пристроях (iOS, Android).

Висновок.

«Приборкувач лабіринтів» – це захоплююча та динамічна гра-головоломка, яка кидає виклик логіці, орієнтації, просторовому мисленню та пам’яті гравця. Завдяки випадковій генерації лабіринтів, різноманіттю

складнощів та захоплюючому геймплею гра стане чудовою розвагою для гравців будь-якого віку та рівня досвіду.

2.2 Вибір виду лабіринту

Взагалі, лабіринтом (від грецького *λαβύρινθος*) називається структура (зазвичай у двовимірному або тривимірному просторі), яка складається з заплутаних шляхів до виходу, а також (часто) шляхів, що можуть вести у глухий кут (рис. 2.1). У стародавніх греків і римлян під лабіринтом мався на увазі більш менш просторий простір, що складається з численних залів, камер, дворів і переходів, розташованих за складним і заплутаним планом, з метою заплутати і не дати виходу необізнаній в плані лабіринту людині.

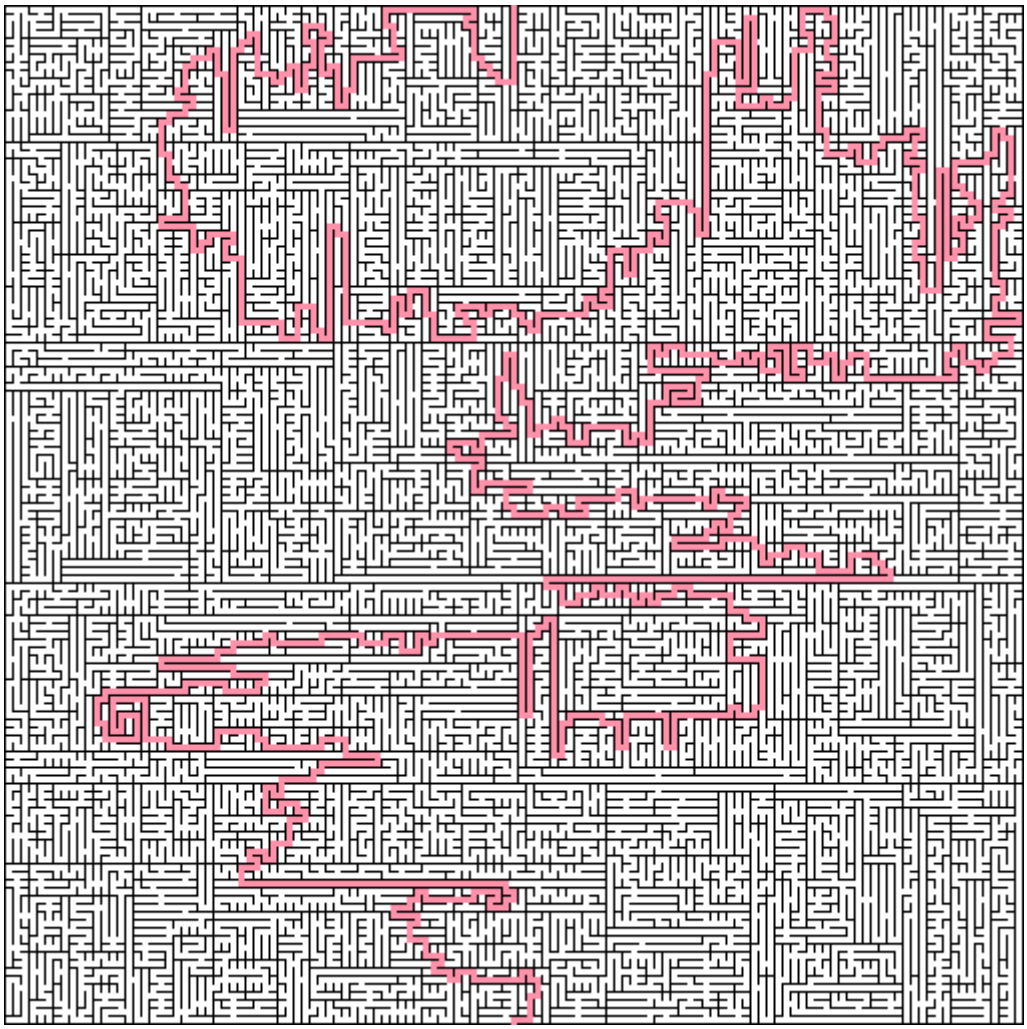

Рис. 2.1 – Приклад звичайного двовимірного лабіринту і його рішення

Існує багато видів лабіринтів [16]. Розглянемо їх за різними критеріями класифікації.

За розмірністю можна розрізняти такі лабіринти:

- двовимірні (2D);
- тривимірні (3D), зокрема, такі, що складаються з декількох двовимірних як висотний будинок з декількох поверхів;
- лабіринти з переплетіннями (2,5D), де проходи можуть накладатися один на один;
- лабіринти більшої розмірності (наприклад, 3D-лабіринти з минулим і майбутнім).

За топологією розрізняють:

- звичайні лабіринти – у евклідовому просторі;
- Planair-лабіринти із незвичайною топологією на різних поверхнях, наприклад: на поверхні куба, на поверхні стрічки Мебіуса, на поверхні тора, де попарно з'єднані ліва і права, верхня і нижня сторони лабіринту тощо.

Геометрія окремих комірок, з яких складається лабіринт, називається тесселяцією. Розрізняють такі типи лабіринтів за тесселяцією:

- ортогональні – з квадратними або прямокутними комірками;
- дельта-лабіринти – з трикутними комірками;
- сигма-лабіринти – з шестикутними комірками;
- тета-лабіринти – складаються з концентричних кіл проходів, в яких початок знаходиться в центрі, а кінець – на зовнішньому краї (або навпаки);
- іпсилон-лабіринти – з восьмикутними комірками;
- дзета-лабіринти – розташовані на прямокутній сітці, але на додаток до горизонтальних та вертикальних проходів допускаються діагональні проходи під кутом 45 градусів;

- інші лабіринти з постійною неортогональною тесселяцією, наприклад, лабіринти, що складаються з пар прямокутних трикутників;
- crack-лабіринти – аморфні лабіринти без постійної тесселяції, в яких стіни та проходи розташовані під випадковими кутами.

Ще один критерій класифікації лабіринтів – маршрутизація. Цей критерій пов'язаний із типами проходів у межах геометрії, визначеної в описаних вище категоріях. За маршрутизацією розрізняють такі типи лабіринтів:

6. «Ідеальний» лабіринт – без петель або замкнутих ланцюгів та без недосяжних областей. Також він називається лабіринтом з одиночною сполукою (*simply-connected maze*). З кожної точки такого лабіринту існує єдиний шлях до будь-якої іншої точки. Модель такого лабіринту можна описати як дерево, що сполучає множину вершин.
7. «Плетений» (*Braid*) лабіринт – не має глухих кутів. Також називається лабіринтом із багатократними сполуками (*purely multiply connected maze*). У такому лабіринті використовуються проходи, що замикаються та повертаються один до одного, таким чином, вони передбачають ходьбу колами замість попадання в глухий кут.
8. Одномаршрутний (*Unicursal*) лабіринт – не має розвилок. Такий лабіринт містить один довгий прохід, що звивається, який змінює напрямок на всьому протязі лабіринту. Пройти його не складно, головне випадково не повернути назад на півдорозі.
9. Розріджений лабіринт – не прокладає проходи через кожну комірку, тобто деякі з них не створюються. Може мати недосяжні області. Подібну концепцію можна застосувати і при додаванні стін, завдяки чому можна отримати нерівномірний лабіринт із широкими проходами та кімнатами.

Ще один критерій класифікації лабіринтів – так звана текстура. Класифікація за текстурою описує стиль проходів за різної маршрутизації та геометрії. Існують такі текстури лабіринтів:

1. Зміщення (*Bias*): у лабіринті зі зміщеними проходами прямі проходи схильні більше йти в одному напрямку, ніж в інших. Наприклад, у лабіринті з високим горизонтальним зміщенням присутні довгі проходи зліва направо, і лише короткі проходи зверху вниз, що з'єднують їх.
2. Прольоти: даний показник визначає, як довго йдуть довгі проходи, перш ніж з'являться вимушені повороти. У лабіринті з низьким показником прольотів немає прямих проходів довше трьох-чотирьох комірок, і він виглядає досить випадковим. У лабіринті з високим показником прольотів великий відсоток довгих проходів, через що він дещо схожий на мікрочіп.
3. Елітність: показник, який визначає довжину рішення (тобто шляху від початку до цілі) відносно розміру лабіринту. У елітних лабіринтів рішення досить коротке і відносно пряме, а в неелітних часто проходить по великій частині площі лабіринту.
4. Симетрія: симетричний лабіринт має симетричні проходи, наприклад, симетрією обертання відносно центру, або відбитий по горизонтальній або вертикальній осях. Лабіринт може бути частково або повністю симетричним, і може повторювати патерн будь-яку кількість разів.
5. Однорідність: однорідний алгоритм генерує всі можливі лабіринти з рівною ймовірністю. Таким чином, лабіринт має однорідну текстуру, якщо він виглядає як типовий лабіринт, згенерований однорідним алгоритмом. Неоднорідний алгоритм генерує можливі лабіринти у будь-якому просторі з різною ймовірністю (у тому числі – з нульовою).

6. Плинність (*River*): ця характеристика означає, що при створенні лабіринту алгоритм шукатиме і очищатиме сусідні комірки (або стіни) до поточної, тобто буде текти (звідси і термін «плинність») у ще нестворені частини лабіринту, як вода. В ідеальному лабіринті з меншим показником плинності більше коротких глухих кутів, а в більш «плинному» лабіринті менше глухих кутів, але вони виявляться довшими.

З розглянутих вище різновидів оберемо для початку один з найпростіших варіантів: двовимірний звичайний ортогональний лабіринт, ідеальний або неідеальний, без явно вираженої текстури і зміщення. У подальшому за необхідності можна буде додати в гру й інші види лабіринтів.

2.3 Алгоритм процедурного генерування лабіринту

Алгоритми створення лабіринтів можна розділити на два основні типи: ті, що додають стіни, та ті, що вирізають проходи в стінах. Один і той самий лабіринт часто можна згенерувати обома способами. Існують і комбіновані варіанти, які задіють обидва підходи, але, можливо, з різним пріоритетом.

Алгоритми першого типу починають із порожньої області (або зовнішньої межі), в процесі роботи додаючи нові стіни і формуючи таким чином лабіринт. У реальному світі лабіринти, які складаються з огорож, перекриттів або дерев'яних стін, створюються саме за таким підходом.

Алгоритми другого типу починають із суцільного блоку та у процесі роботи вирізають у ньому проходи. У реальному світі такими лабіринтами є, наприклад, тунелі шахт або лабіринти усередині труб.

Загальний підхід до створення стандартного ідеального лабіринту додаванням стін – це «виростити» його зі стін так, щоб забезпечити відсутність петель та ізольованих областей. Для цього треба розпочати із

зовнішньої стіни і випадковим чином додавати фрагмент стіни, який з нею стикається. Далі треба продовжувати випадковим чином додавати в лабіринт сегменти стін, перевіряючи, що кожен новий сегмент торкається одним кінцем існуючої стіни, а його інший кінець знаходиться в ще не створеній частині лабіринту. Якщо додається сегмент стіни, обидва кінці якого відокремлені від решти лабіринту, це створить непоєднану стіну з петлею навколо. Якщо додати сегмент, обидва кінця якого приєднуються до лабіринту, це утворить недосяжну область.

Майже аналогічно способу додавання стін працює вирізання проходів. При цьому підході частини проходів вирізаються в такий спосіб, щоб існуючого проходу торкався лише один кінець існуючого сегменту.

Існує безліч конкретних алгоритмів створення ідеальних лабіринтів. Їх також можна поділити на два основні типи. Алгоритм на основі дерева вирощує лабіринт подібно до дерева, додаючи нові фрагменти до вже згенерованої частини. Таким чином, на кожному етапі цей алгоритм має правильний ідеальний лабіринт. Алгоритм на основі множин виконує побудови фрагментів в довільних місцях, а потім з'єднує їх, створюючи правильний лабіринт в кінці. Деякі алгоритми, наприклад алгоритм вирощування лісу, використовують одночасно обидва підходи.

В таблиці 2.1 наведені характеристики відомих алгоритмів генерування лабіринтів [16].

Для проекту гри оберемо алгоритм *Recursive backtracker*. Він досить простий в реалізації, швидкий і генерує якісні лабіринти за характеристиками маршрутизації і плинності. При правильній реалізації алгоритм *Recursive backtracker* виконується досить швидко – швидше працюють лише вузькоспеціалізовані алгоритми.

Загальна (принципова) блок-схема алгоритму генерування лабіринту *Recursive backtracker* показана на рис. 2.2.

Таблиця 2.1

Характеристики алгоритмів генерування лабіринтів

Алгоритм	% глухих кутів	Тип	Пріоритет	Зміщеність відсутня?	Однорідний?	Пам'ять	Час	% рішення
Одномаршрутний	0	Дерево	Стіни	Так	Ні	N^2	379	100,0
Рекурсивний Backtracker	10	Дерево	Проходи	Так	Ні	N^2	27	19
Полювати і вбивати	11 (21)	Дерево	Проходи	Так	Ні	0	100 (143)	9,5 (3,9)
Рекурсивний поділ	23	Дерево	Стіни	Так	Ні	N*	10	7.2
Двійкове дерево	25	Множина	Обидва	Ні	Ні	0*	10	2
Сайдвіндери	27	Множина	Обидва	Ні	Ні	0*	12	2.6
Алгоритм Еллера	28	Множина	Обидва	Ні	Ні	N*	20	4,2 (3,2)
Алгоритм Вілсона	29	Дерево	Обидва	Так	Так	N^2	48 (25)	4.5
Алгоритм Альдоза-Бродера	29	Дерево	Обидва	Так	Так	0	279 (208)	4.5
Алгоритм Фарбала	30	Множина	Обидва	Так	Ні	N^2	33	4.1
Алгоритм Пріма (істинний)	30	Дерево	Обидва	Так	Ні	N^2	160	4.1
Алгоритм Пріма (спрощений)	32	Дерево	Обидва	Так	Ні	N^2	59	2.3
Алгоритм Пріма (модифікований)	36 (31)	Дерево	Обидва	Так	Ні	N^2	30	2.3
Вирощування дерева	49 (39)	Дерево	Обидва	Так	Ні	N^2	48	11
Вирощування лісу	49 (39)	Обидва	Обидва	Так	Ні	N^2	76	11

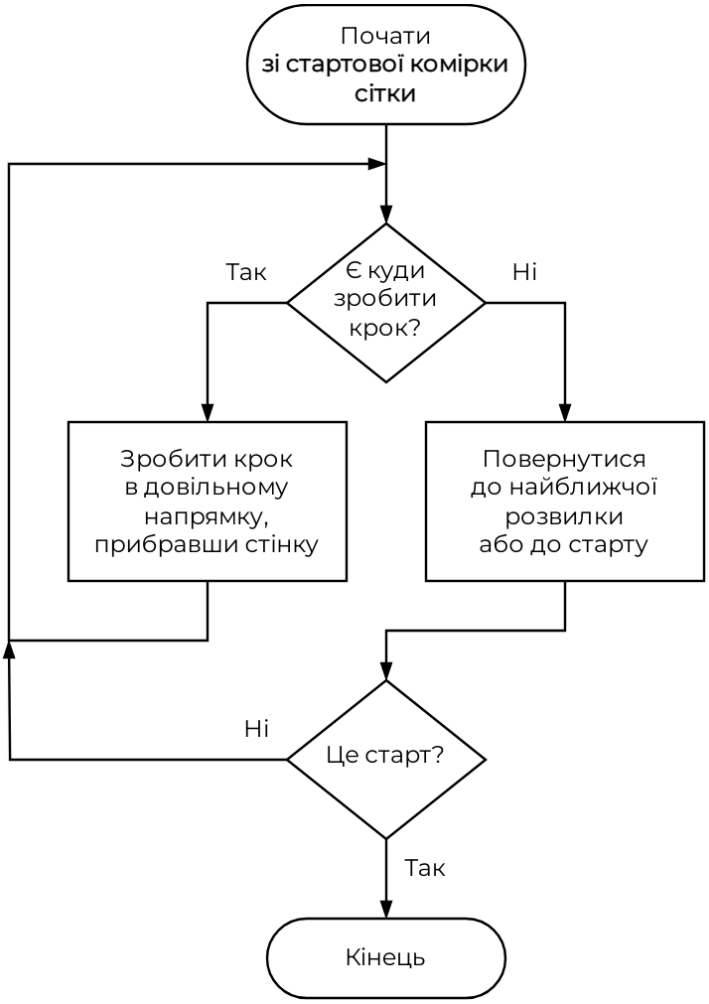

Рис. 2.2 – Принципова блок-схема алгоритму *Recursive backtracker*

Починаючи зі стартової комірки, на кожному кроці ми рухаємося у випадковому напрямі й прибираємо стінку, на яку натрапили. Коли ми приходимо в глухий кут, звідки не можна нікуди піти, ми вертаємося назад до комірки, з якої ми можемо повернути в якомусь іншому напрямі. Такі дії продовжуються, поки ми не повернемося до початкової комірки.

Деталізована блок-схема алгоритму *Recursive backtracker*, яка специфікує певні особливості його програмної реалізації, показана на рис. 2.3. Як видно з блок-схеми, для визначення можливого напрямку прямого руху формується список комірок, які є сусідніми для поточної. Для реалізації можливості повертання по пройденому шляху (зворотного руху) використовується стек, обсяг якого може доходити до розмірів лабіринту. При прямому просуванні алгоритм вирізає прохід у ще не створеній частині,

якщо вона існує поруч із поточною коміркою. Щоразу, коли ми переміщаємося до нової комірки, записуємо попередню комірку у стек. Якщо поруч із поточною позицією немає нестворених комірок, то витягаємо зі стеку попередню позицію. Лабіринт завершено, коли в стеку більше нічого не залишається.

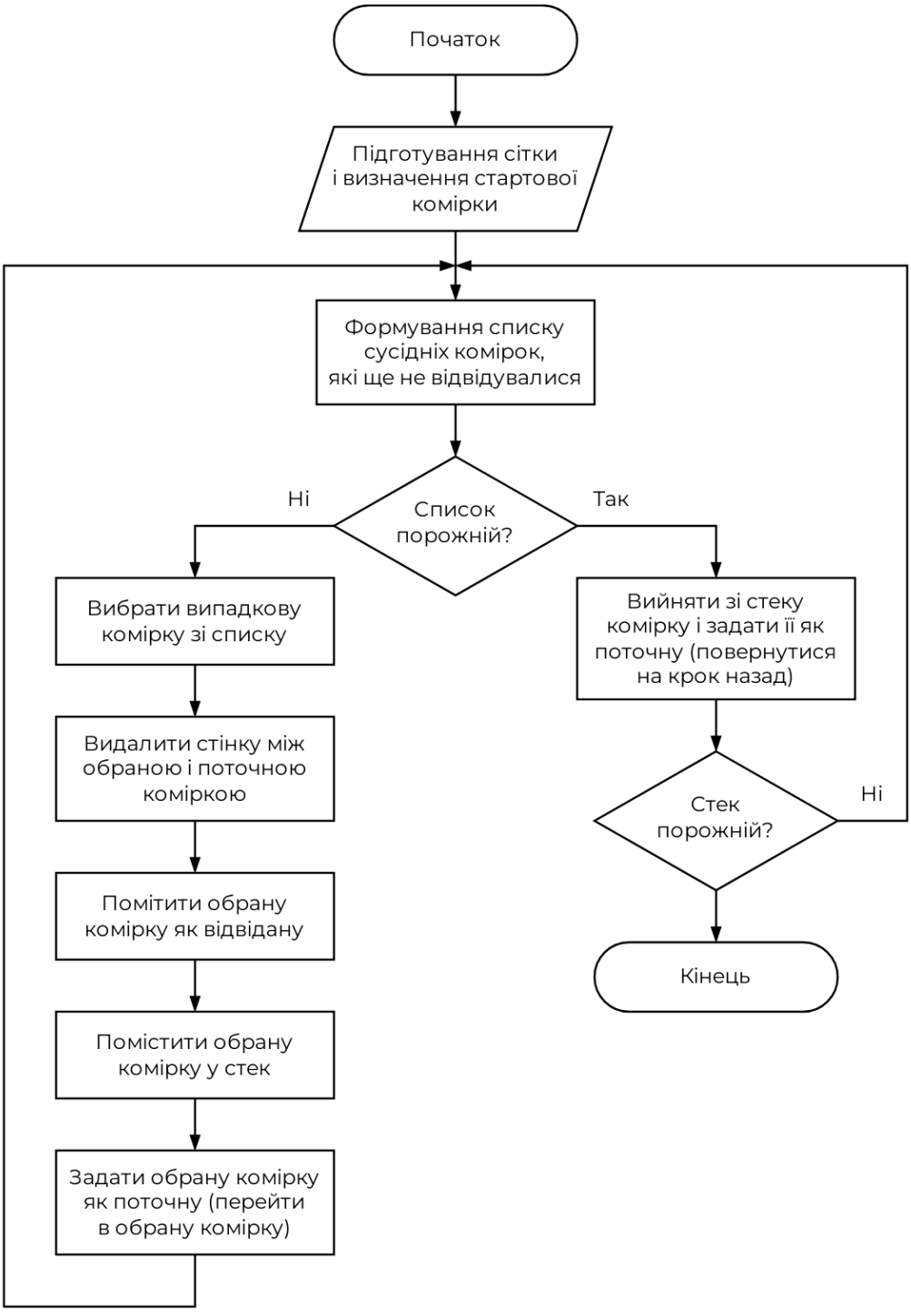

Рис. 2.3 – Деталізована блок-схема алгоритму *Recursive backtracker*

Робота алгоритму призводить до створення лабіринтів з максимальним показником плинності: глухих кутів в них менше, а самі вони довші. Рішення такого лабіринту зазвичай виявляється вельми довгим і звивистим. Важливо також, що це рішення – єдине, тобто не існує інших шляхів від початку до виходу.

2.4 Діаграми UML

Для проектної документації були розроблені UML-діаграми варіантів використання і класів (2.13).

Діаграма варіантів використання (Use Case) відображає функціональні компоненти гри (рис. 2.12).

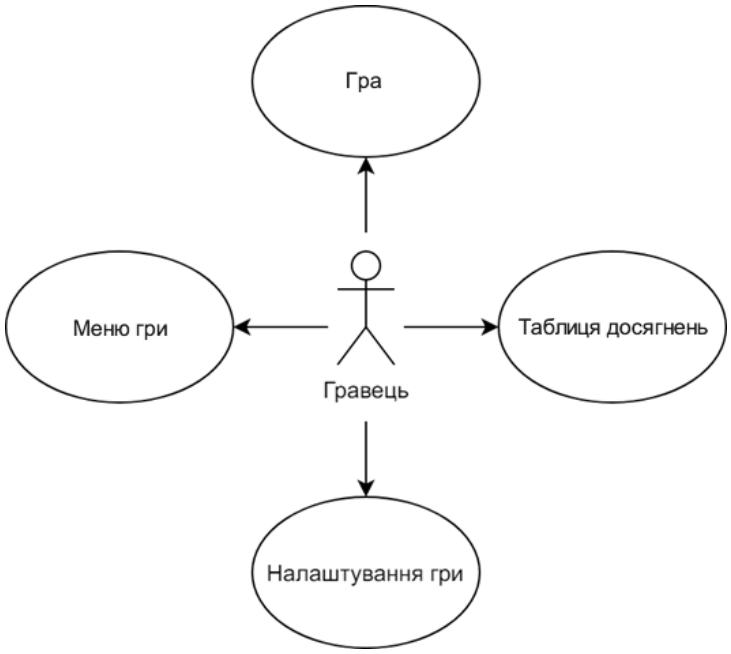

Рис. 2.4 – Діаграма варіантів використання

Діаграма класів показує структуру класів проекту гри і зв’язків між ними (рис. 2.5).

Центральним класом гри є клас *Maze* (Лабіринт), який представляє собою масив клітинок *Cell*. За генерацію лабіринту відповідає клас *MazeGenerator*, а за його реалізацію на сцені – клас *MazeSpawner*. Клас

HintRenderer відповідає за формування підказки у вигляді шляху до виходу, а клас *PlayerControls* – за управління персонажем.

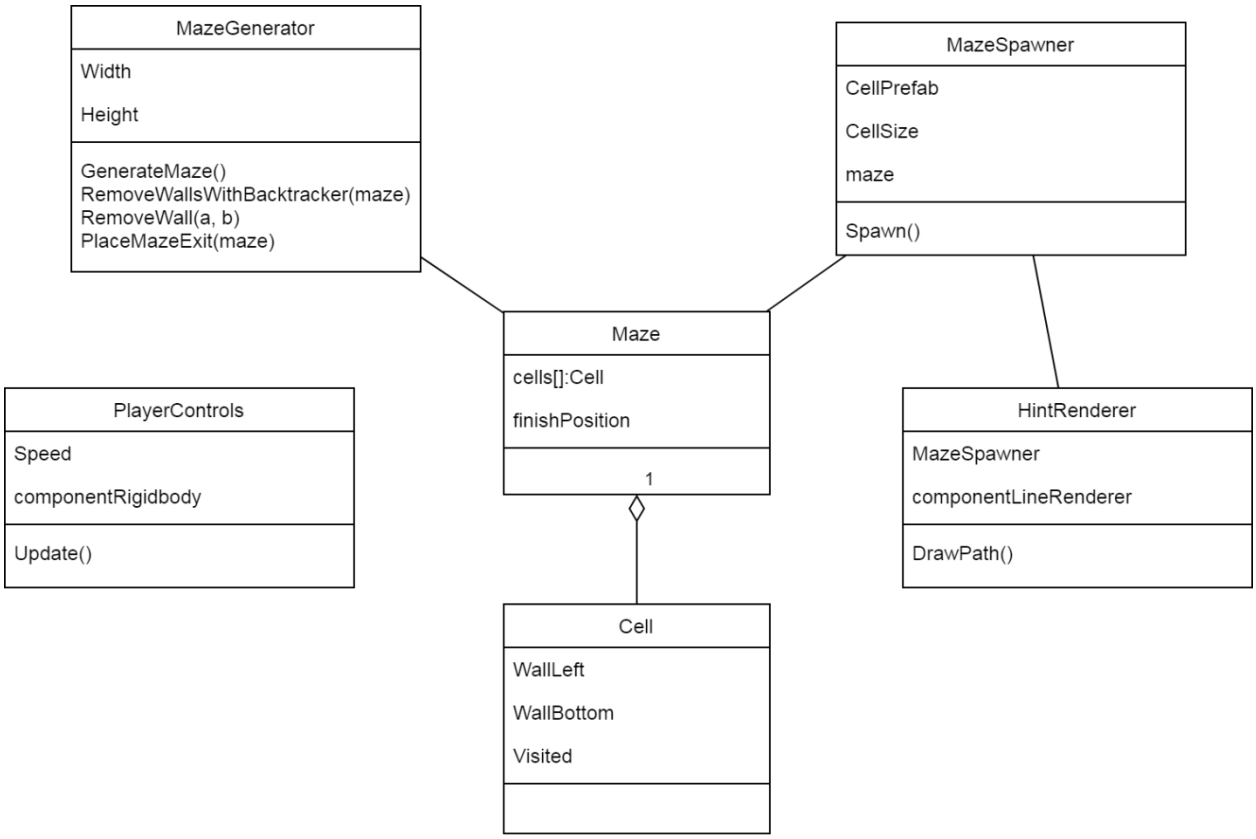

Рис. 2.5 – Діаграма класів проекту гри

2.5 Елементи дизайну гри

2.5.1 Особливості геймплею і система нарахування балів

При старті нової гри генерується невеликий лабіринт розміром 20x10 клітин. Після проходження першого рівня генерується другий – розміром 22x11 клітин. З кожним новим рівнем кількість клітин по горизонталі збільшується на 2, а по вертикалі – на 1. З часом увесь лабіринт перестає поміщатися в екран – лише його частина. Коли гравець переміщує персонажа до краю цього фрагменту, який не є краєм лабіринту, камера зсувається, відкриваючи приховану область.

Гра замірює час проходження лабіринту і кількість кроків, які зробив гравець (у клітинах) – чим менші ці параметри, тим більше балів нараховується.

Кількість балів за рівень розраховується по формулі:

$$Б = Ш \cdot В \cdot 5 - К - Ч \geq 0,$$

де $Ш$ – ширина лабіринту у клітинах;

$В$ – висота лабіринту у клітинах;

$К$ – кількість зроблених кроків;

$Ч$ – витрачений час, с.

Якщо в результаті віднімання виходить від'ємне число, воно прирівнюється нулю.

2.5.2 Ескізи стартового екрану і меню гри

За допомогою нейронних мереж, що генерують зображення, було створено декілька варіантів тематичного зображення для стартового екрану по запиту «Приборкувач лабіринту» (рис. 2.6).

Серед згенерованих зображень було обрано варіант, показаний на рис. 2.6, в. На його основі було зроблено стартовий екран з назвою гри (рис. 2.7).

Після завантаження гри відкривається вікно головного меню програми, схематичний ескіз якого показаний на рис. 2.8.

Після натискання «Розпочати нову гру» з'являється екран введення імені користувача (рис. 2.9).

Кількість балів за проходження гри накопичуються у таблиці досягнень (рис. 2.10).

а)

б)

в)

г)

Рис. 2.6 – Варіанти зображень для стартового екрану, згенеровані нейромережами:
а) dreamAI; б) fisionbrain AI; в) і г) Stable Diffusion

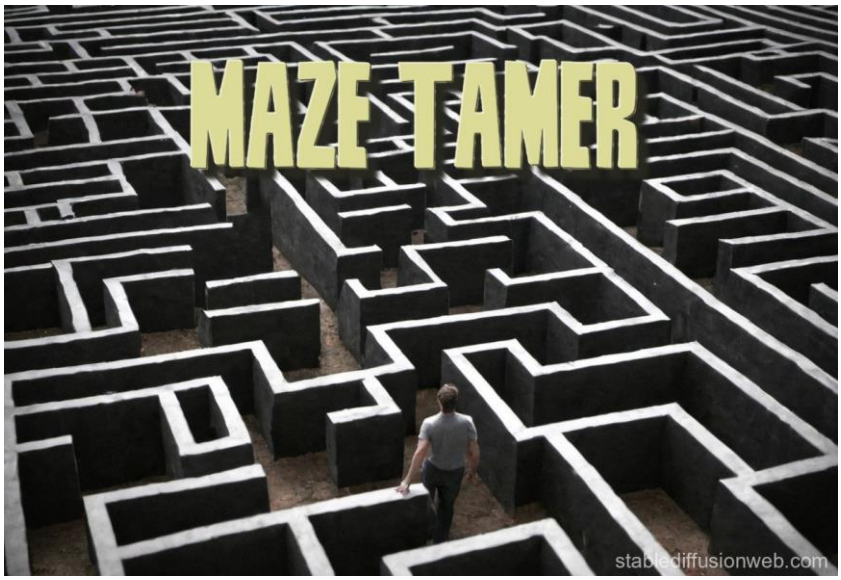

Рис. 2.7 – Варіант стартового екрану гри

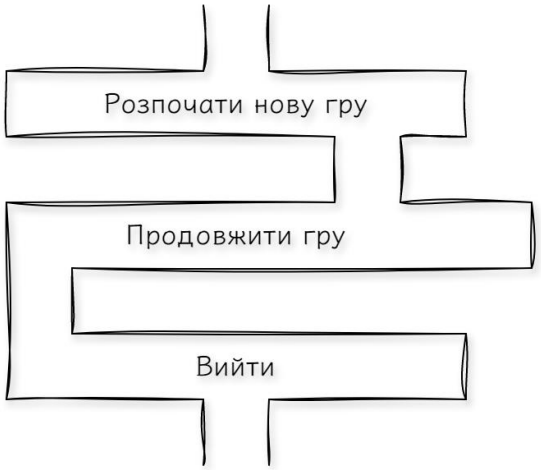

Рис. 2.8 – Концепт екрану з ігровим меню

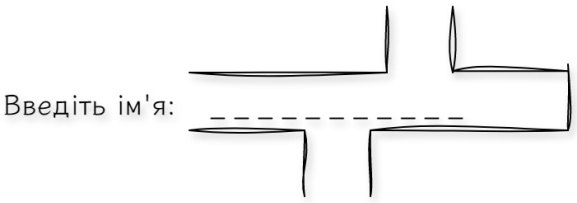

Рис. 2.9 – Концепт екрану з введенням імені користувача

Рис. 2.10 – Концепт екрану з таблицею досягнень гравців

Для тексту в ескізах екранів використаний шрифт з бібліотеки Google Fonts під назвою «LXGW WenKai TC». Він доступний за веб-адресою https://fonts.google.com/specimen/LXGW+WenKai+TC.

2.5.3 Ескіз екрану ігрового рівня

На рис. 2.11 показаний ескіз основного ігрового рівня. По натисканню кнопки «Розпочати заново» персонаж опиняється у стартовій позиції; лабіринт при цьому не змінюється. По натисканню «Отримати підказку» на 3 секунди з'являється лінія підказки, яка показує шлях до виходу. По кнопці «Вийти» здійснюється вихід до головного меню.

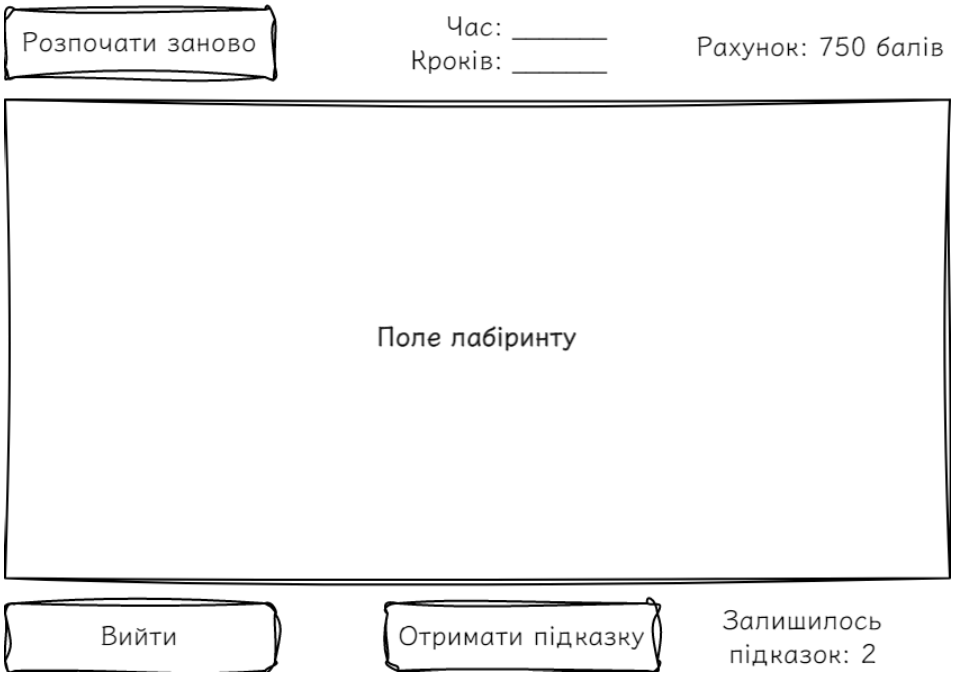

Рис. 2.11 – Ескіз екрану з основним ігровим рівнем

2.6 Вибір рушія для реалізації проекту

Одними з популярних засобів розробки комп'ютерних ігор є середовища розробки ігрових рушіїв. Вони дозволяють за невеликий час зробити багатоплатформний проект. Ціною за це можуть бути певні компроміси по оптимізації, але розробники постійно вдосконалюють свої продукти, так що це часто не є вирішальним.

Найбільш популярними ігровими рушіями зараз є Unity (рис. 2.12) та Unreal Engine (рис. 2.13). Який з них краще – однозначної відповіді немає.

Обидва є потужними інструментами, які пропонують безліч функцій та можливостей.

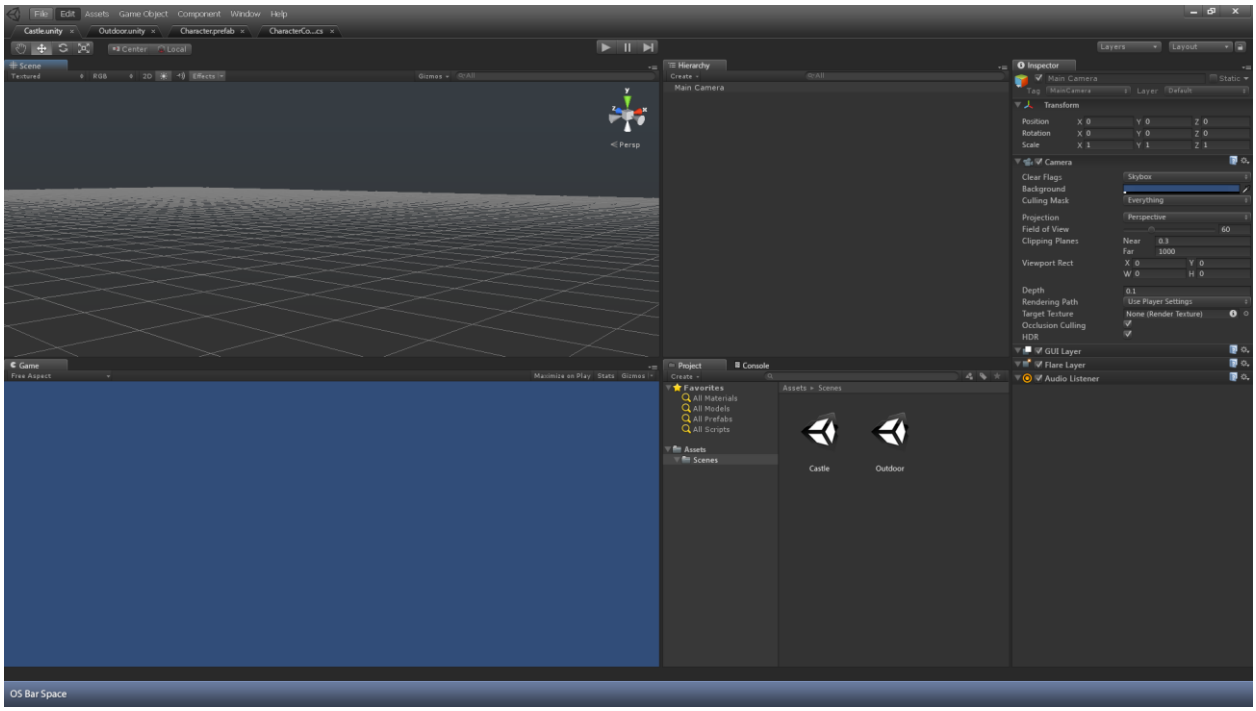

Рис. 2.12 – Інтерфейс користувача ігрового рушія Unity

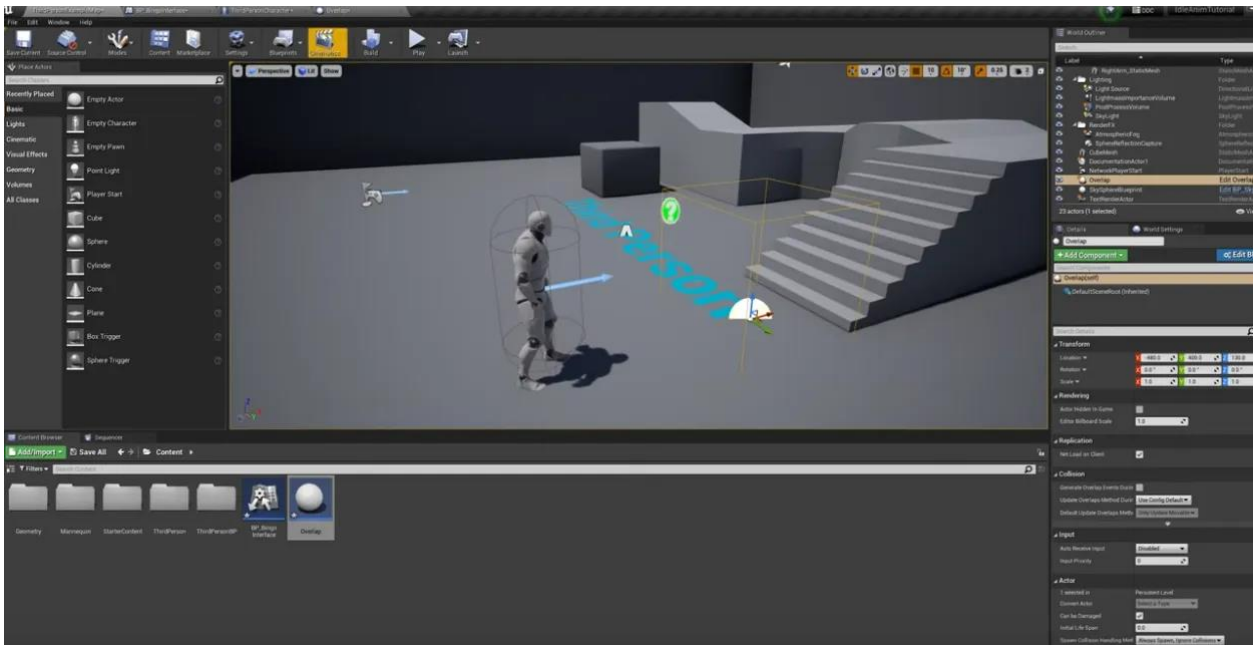

Рис. 2.13 – Інтерфейс користувача ігрового рушія Unreal Engine

Вибір кращого рушія залежить, в першу чергу, від конкретних потреб та особливостей проекту.

Також на вибір рушія впливають переваги і недоліки кожного з них. Розглянемо основні з них.

Перевагами Unity є:

- відносна легкість у вивченні та використанні, що важливо для початківців;
- велика та активна спільнота розробників;
- широкий спектр навчальних матеріалів та ресурсів;
- універсальність: він підходить як для 2D, так і для 3D ігор, а також VR/AR проектів;
- оптимізованість для мобільних платформ;
- безкоштовність для невеликих проектів.

Недоліки Unity:

- у середньому менша потужність для складних проектів, ніж Unreal Engine;
- деякі функції доступні лише у платній версії (Unity Pro).

Переваги Unreal Engine:

- він більш потужний та гнучкий, підходить для створення складних ігор з високоякісною графікою;
- професійний рівень візуалізації та ефектів;
- підходить для створення ігор для ПК, консолей та мобільних пристроїв;
- велика бібліотека готових ассетів та плагінів;

Недоліки Unreal Engine:

- він дещо складніший у вивченні, ніж Unity, і, відповідно, потребує більше часу та навичок;
- менша спільнота розробників порівняно з Unity;
- більш високі вимоги до комп’ютера;

Взагалі, найкращий спосіб вибрати рушій – це спробувати обидва та подивитися, який з них більше підходить. Але також на вибір суттєво впливає наявність досвіду в розробника. Оскільки в нашому випадку досвіду більше з Unity, обираємо його.

Висновки до другого розділу

В другому розділі кваліфікаційної роботи були отримані і висвітлені такі результати:

1. Розроблений концепт гри-головоломки і вибраний конкретний вид лабіринту із загальної їх класифікації.

2. Розібраний алгоритм процедурного генерування лабіринту у більш загальному та більш детальному варіантах.

3. Створені концепти і ескізи основних екранів гри.

4. Вибраний ігровий рушій Unity як основний інструментальний засіб реалізації проекту гри.

Ці напрацювання дозволяють здійснити подальшу реалізацію гри у вигляді застосунку.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ ГРИ

3.1 Створення сцени і прототипу керованого персонажу

Після створення проекту Unity зробимо тестову двовимірну сцену під назвою *Maze2D*, на якій буде процедурно генеруватися 2D-лабіринт поки що фіксованої розмірності.

В якості ігрового персонажа, яким можна буде керувати для проходження лабіринту, на даному етапі використовуватимемо простий зелений кружечок.

Створюємо на сцені ігровий об’єкт *Player* (рис. 3.1), додаємо до нього компонент *Sprite Renderer*, в якості спрайту призначаємо вбудований в Unity графічний ресурс *Knob* і задаємо для нього зелений колір.

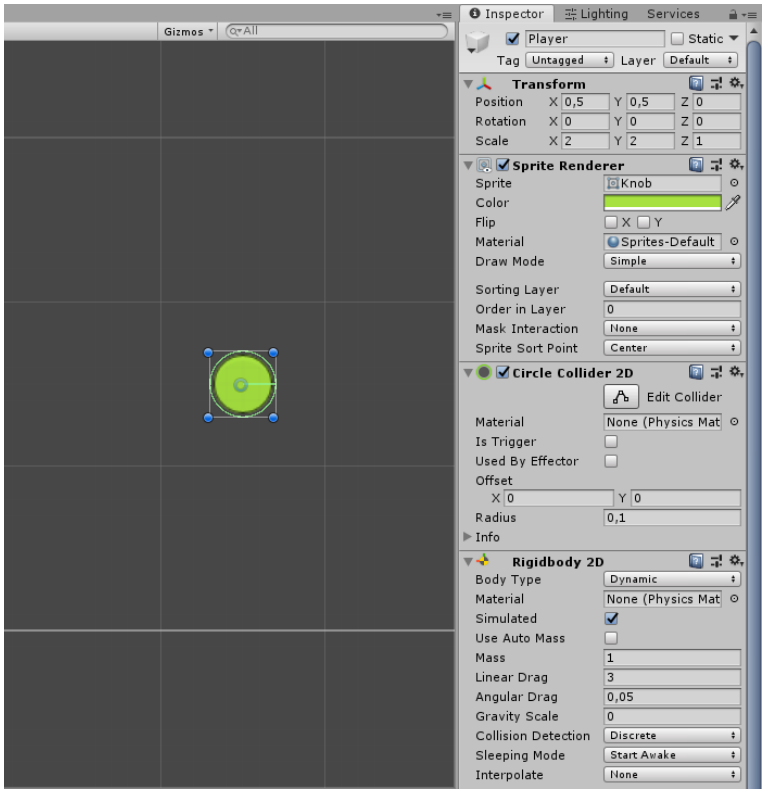

Рис. 3.1 – Ігровий об’єкт *Player* в редакторі властивостей Unity

За допомогою компоненту *Transform* налаштовуємо для об'єкту *Player* розмір (x2) і позицію (0,5). За допомогою компоненту *Circle Collider 2D* додаємо до нього і налаштовуємо колайдер, щоб об'єкт не міг проходити крізь стіни лабіринту. Також додаємо компонент *Rigidbody 2D*, за допомогою якого будемо реалізовувати фізику руху і зіткнень.

Для керування об'єктом *Player* з клавіатури додаємо скрипт *Player Controls* такого змісту:

```
using UnityEngine;

public class PlayerControls : MonoBehaviour
{
    public float Speed = 2;

    private Rigidbody2D componentRigidbody;

    private void Start()
    {
        componentRigidbody = GetComponent<Rigidbody2D>();
    }

    private void Update()
    {
        componentRigidbody.velocity = Vector2.zero;
        if (Input.GetKey(KeyCode.LeftArrow)) componentRigidbody.velocity +=
Vector2.left * Speed;
        if (Input.GetKey(KeyCode.RightArrow)) componentRigidbody.velocity +=
Vector2.right * Speed;
        if (Input.GetKey(KeyCode.UpArrow)) componentRigidbody.velocity +=
Vector2.up * Speed;
        if (Input.GetKey(KeyCode.DownArrow)) componentRigidbody.velocity +=
Vector2.down * Speed;
    }
}
```

3.2 Реалізація клітинки як елементу лабіринту

Лабіринт в нас буде прямокутний, горизонтально орієнтований і складатиметься з квадратних клітинок. Для формування клітинок створюємо об'єкт Cell, який міститиме два підоб'єкти: стінка зліва *Wall Left* і стінка знизу *Wall Bottom*. Стінку формуємо за допомогою компонента *Line Renderer*, використовуючи при цьому стандартний матеріал *Default-Line*. Задаємо координати кінцевих точок відрізка і ширину, заокруглення кінців, а також колір (рис. 3.2). За допомогою компоненту *Edge Collider 2D*

додаємо до стінки колайдер, щоб ігровий персонаж не міг проходити крізь стіни лабіринту.

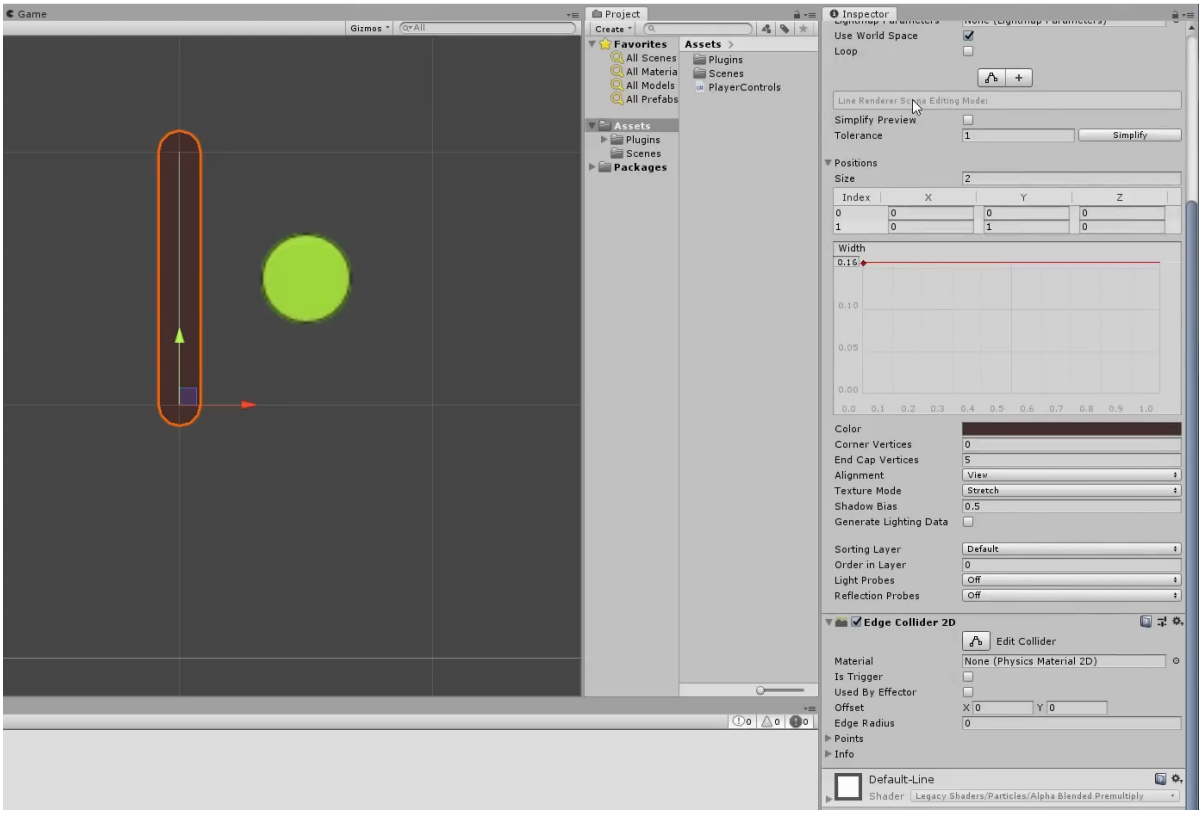

Рис. 3.2 – Формування лівої стінки клітинки

Нижню стінку комірки отримаємо копіюванням об’єкту лівої стінки, змінивши координати в неї самої та в її колайдера (рис. 3.3). Також знімемо прапорець *World Space*, щоб комірка мала локальні координати, а не світові, і її зручно було переміщати.

Розмножувати клітинки будемо у скрипті, тому створюємо на основі об’єкту *Cell* префаб, а зі сцени його прибираємо.

Далі нам знадобиться об’єкт породжувача лабіринту, який буде відображати його на сцені. Для цього створюємо скрипт *MazeSpawner.cs*, в класі якого створюємо поле *CellPrefab* типу *GameObject*:

```
public class MazeSpawner : MonoBehaviour
{
    public GameObject CellPrefab;
```

Після цього зв’язуємо створене поле з префабом *Cell* в редакторі Unity.

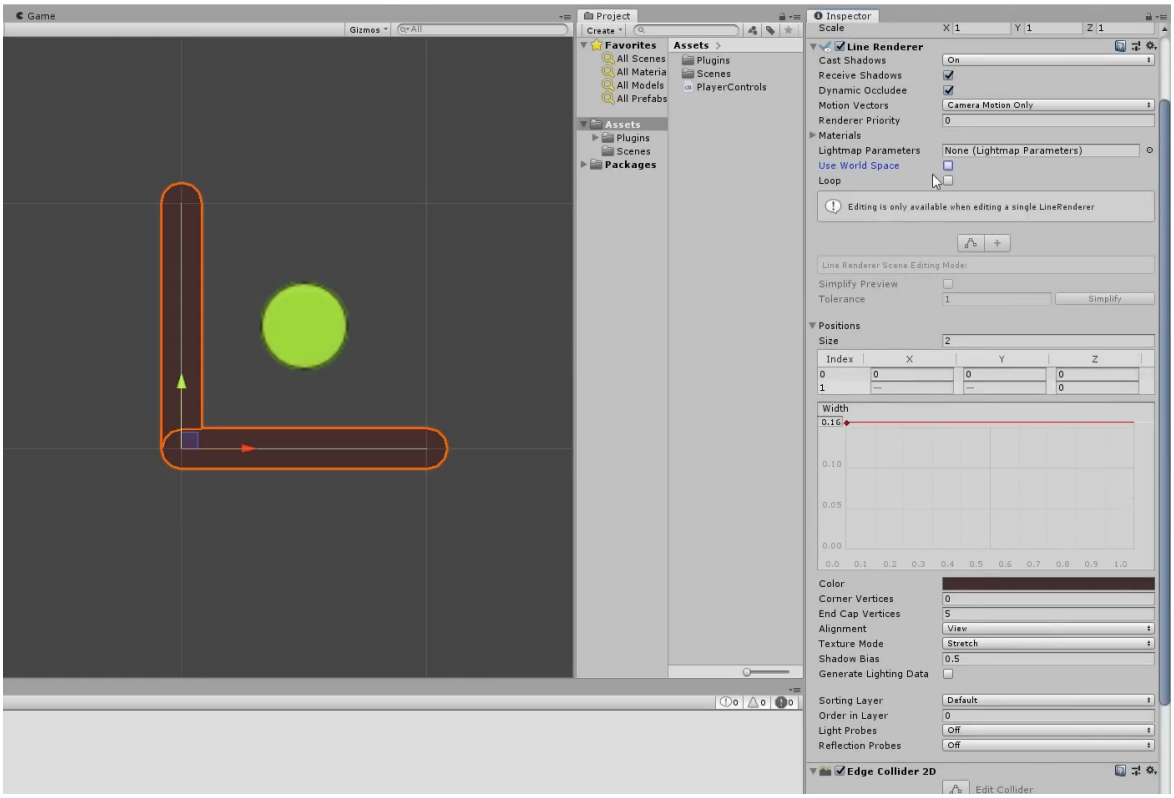

Рис. 3.3 – Дві стінки однієї клітинки

3.3 Створення сітки як основи лабіринту

Далі створюємо окремий скрипт, який відповідатиме за генерування лабіринту: *MazeGenerator.cs*. Відокремлення скрипту генерування від скрипта відображення дасть можливість за необхідності швидко і зручно замінити алгоритм генерації лабіринту або додати інші такі алгоритми в майбутньому. В скрипті *MazeGenerator.cs* створюємо публічний метод *GenerateMaze()*, який повертатиме об’єкт-лабіринт класу *Maze*, а також змінні параметрів генерації: для початку – ширину і висоту лабіринту у кількості комірок:

```
public class MazeGenerator
{
    public int Width;
```

```
    public int Height;

    public Maze GenerateMaze()
    {
```

Для програмної реалізації комірок створюємо клас *MazeGeneratorCell*. Кожен об'єкт такого класу представлятиме комірку з двома булевими змінними для стінок *WallLeft* та *WallBottom*, за допомогою яких можна буде «вимикати» (прибирати) відповідні стінки комірки. Початково робимо стінки «увімкненими», тобто існуючими (*true*):

```
public class MazeGeneratorCell
{
    public int X;
    public int Y;

    public bool WallLeft = true;
    public bool WallBottom = true;
}
```

Для програмного представлення лабіринту створюємо клас *Maze*, який міститиме двовимірний масив комірок і позицію фінішу. Пізніше в нього можна буде додати інші параметри нашого лабіринту, такі як положення старту (який у нас зараз завжди в лівому нижньому куті), розміри тощо. Помістимо у скрипт *Maze.cs* також клас програмної комірки *MazeGeneratorCell*:

```
public class Maze
{
    public MazeGeneratorCell[,] cells;
    public Vector2Int finishPosition;
}

public class MazeGeneratorCell
{
      ...
```

Метод *GenerateMaze()* класу *MazeGenerator* пов `GenerateMaze()`ертатиме двовимірний масив програмних комірок типу *MazeGeneratorCell*:

```
public MazeGeneratorCell[,] GenerateMaze()
{
    MazeGenerarorCell[,] cells = new MazeGeneratorCell[Width, Height];
    ...
    return maze;
```

```
}
```

Новостворений масив *cells* спочатку нічим не заповнений (точніше, заповнений значеннями *null*), тому заповнимо його вручну об’єктами класу *MazeGeneratorCell*:

```
for (int x = 0; x < cells.GetLength(0); x++)
{
    for (int y = 0; y < cells.GetLength(1); y++)
    {
        cells[x, y] = new MazeGeneratorCell {X = x, Y = y};
    }
}
```

Після цього необхідно створити, власне, об’єкт лабіринту *maze*, ініціалізувати його поля масивом комірок та позицією фінішу і повернути з методу:

```
Maze maze = new Maze();

maze.cells = cells;
maze.finishPosition = PlaceMazeExit(cells);

return maze;
```

Метод *GenerateMaze()* можна викликати в класі *MazeSpawner*, створивши при цьому необхідні об’єкти. По-перше, при старті треба створити новий генератор лабіринту, далі викликати в нього метод *GenerateMaze()* і пройтися циклом по всьому лабіринту, щоб створити ігрові об’єкти комірок на основі раніше створеного префабу *Cell*:

```
private void Start()
{
    MazeGenerator generator = new MazeGenerator();
    maze = generator.GenerateMaze();

    for (int x = 0; x < maze.cells.GetLength(0); x++)
    {
        for (int y = 0; y < maze.cells.GetLength(1); y++)
        {
            Instantiate(CellPrefab, new Vector2(x, y),
              Quaternion.identity);
        }
    }
```

Якщо запустити проект в Unity, можна побачити, що з комірок створюється сітка (рис. 3.4).

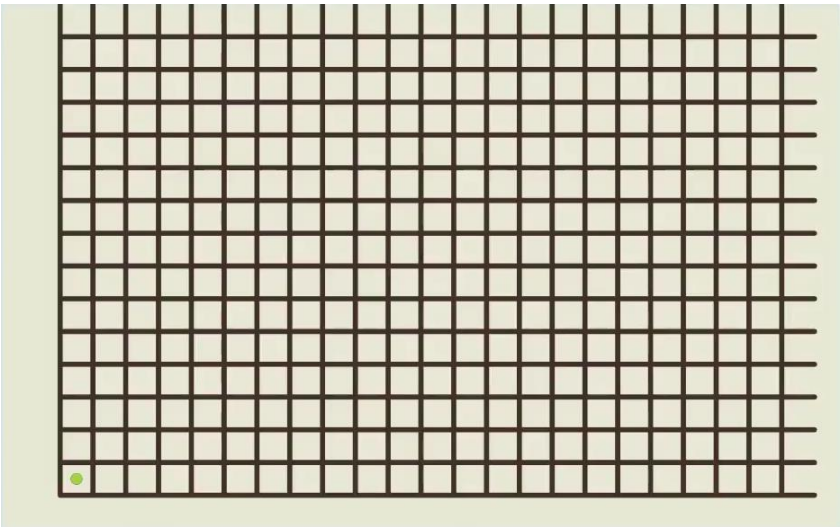

Рис. 3.4 – Сітка з клітин як основа майбутнього 2D-лабіринту

Видно, що для правильного вигляду поля лабіринту треба буде щось зробити з верхнім рядком та правим стовпцем. Розберемося з цим пізніше.

3.4 Реалізація алгоритму генерування лабіринту

Для генерування лабіринту треба управляти стінками комірок («вимикати» деякі з них). Відповідно, нам знадобляться посилання на стінки клітинок. Для цього створимо ще один скрипт і, відповідно, клас – *Cell*:

```
using UnityEngine;

public class Cell : MonoBehaviour
{
    public GameObject WallLeft;
    public GameObject WallBottom;
}
```

Як видно з фрагменту коду, клас *Cell* містить лише два поля типу *GameObject* для стінок комірки: лівої та нижньої. Зв'язок між цими полями і ігровими об'єктами в Unity здійснюємо штатним чином в редакторі: додаємо скрипт *Cell* до ігрового об'єкту *Cell*, після чого перетягуємо

об’єкти стінок у відповідні поля скрипту в інспекторі. Завдяки цьому ми позбавляємося необхідності шукати відповідні ігрові об’єкти по іменах.

Тепер можна оновити цикл формування комірок в класі *MazeSpawner*, а саме прибрати стінки, які «вимкнув» генератор лабіринту:

```
      Cell c = Instantiate(CellPrefab, new Vector3(x, y),
Quaternion.identity).GetComponent<Cell>();

      c.WallLeft.SetActive(maze[x, y].WallLeft);
      c.WallBottom.SetActive(maze[x, y].WallBottom);
```

Далі треба реалізувати основний алгоритм генерації лабіринту в класі *MazeGenerator*. Для цього перш за все створюємо заготівку методу *RemoveWallsWithBacktracker()*:

```
private void RemoveWallsWithBacktracker(MazeGeneratorCell[,] maze)
{
    ...
}
```

Даний метод прийматиме на вході лабіринт у вигляді двовимірного масиву *maze* і видалятиме в його комірках деякі стінки.

Алгоритм починає роботу зі стартової комірки з координатами [0, 0]:

```
MazeGeneratorCell current = maze[0, 0];
```

На кожному кроці ми рухаємося у випадковому напрямі й прибираємо стінку, на яку натрапили. Коли ми приходимо в глухий кут, ми вертаємося назад до комірки, з якої ми можемо повернути в якомусь іншому напрямі. Такі дії продовжуються, поки ми не повернемося до початкової комірки.

Для реалізації можливості вертання нам знадобиться стек:

```
Stack<MazeGeneratorCell> stack = new Stack<MazeGeneratorCell>();
```

В циклі do-while здійснюємо звернення до стеку, поки там є комірки:

```
do
{
    ...
} while (stack.Count > 0);
```

Для перевірки можливості зробити крок створюємо в циклі список *unvisitedNeighbours* для сусідів поточної комірки, яких ми ще не відвідували:

```
List<MazeGeneratorCell> unvisitedNeighbours = new
      List<MazeGeneratorCell>();
```

Для того, щоб фіксувати (а потім і дізнаватися), відвідували ми комірку чи ні, створюємо в класі *MazeGeneratorCell* поле-прапор *Visited* типу *bool*:

```
public bool Visited = false;
```

На початку алгоритму помічаємо стартову комірку як вже відвідану:

```
current.Visited = true;
```

Далі треба перевити сусідів поточної комірки. Починаємо з лівого сусіда, який має координату [x – 1]. Якщо лівий сусід ще не відвідувався, додаємо його до списку не відвіданих комірок:

```
int x = current.X;
int y = current.Y;

if (x > 0 && !maze[x - 1, y].Visited)
      unvisitedNeighbours.Add(maze[x - 1, y]);
```

Аналогічно перевіряємо і обробляємо інших сусідів – знизу, справа і зверху:

```
if (y > 0 && !maze[x, y - 1].Visited)
      unvisitedNeighbours.Add(maze[x, y - 1]);
if (x < Width - 2 && !maze[x + 1, y].Visited)
      unvisitedNeighbours.Add(maze[x + 1, y]);
if (y < Height - 2 && !maze[x, y + 1].Visited)
      unvisitedNeighbours.Add(maze[x, y + 1]);
```

Далі перевіряємо, чи можемо ми кудись піти від поточної комірки. Якщо є декілька сусідніх комірок, треба випадково вибрати одну з них і піти до неї:

```
if (unvisitedNeighbours.Count > 0)
```

```
            {
                MazeGeneratorCell chosen =
unvisitedNeighbours[UnityEngine.Random.Range(0, unvisitedNeighbours.Count)];
                ...
            }
```

Щоб перейти до обраної комірки, треба прибрати відповідну стінку. Для цього пізніше розробимо спеціальну функцію *RemoveWall()*, куди передаватимемо поточну і обрану клітинки:

```
                RemoveWall(current, chosen);
```

Ще треба відмітити обрану комірку як відвідану, щоб повторно до неї не заходити при прямому русі, додати її у стек, щоб була можливість повернутися до неї при зворотному русі, а також привласнити змінній *current* об’єкт обраної комірки:

```
                chosen.Visited = true;
                stack.Push(chosen);
                current = chosen;
```

Якщо сусідів поточної комірки немає, повертаємося назад по стеку:

```
            else
            {
                current = stack.Pop();
            }
```

Для прибирання стінки створюємо спеціальну функцію *RemoveWall*():

```
    private void RemoveWall(MazeGeneratorCell a, MazeGeneratorCell b)
    {
        if (a.X == b.X)
        {
            if (a.Y > b.Y) a.WallBottom = false;
            else b.WallBottom = false;
        }
        else
        {
            if (a.X > b.X) a.WallLeft = false;
            else b.WallLeft = false;
        }
    }
```

Сюди передаються дві сусідні клітинки: *a* (поточна) та *b* (обрана). Вони знаходяться або на одній горизонталі (співпадає координата *y*), або на

одній вертикалі (співпадає координата *x*). Якщо співпадає координата *x*, то координата *y* або більша, або менша на 1 від координати *y* іншої комірки. Аналогічно, якщо співпадає *y*, то *x* або більша або менша на 1. Зокрема, якщо ($a.Y > b.Y$), це значить, що обрана клітинка *b* знаходиться нижче поточної клітинки *a*. Відповідно, треба в комірки *a* видалити нижню стінку:

```
if (a.Y > b.Y) a.WallBottom = false;
```

Інакше обрана клітинка *b* знаходиться вище поточної клітинки *a*, і треба в комірки *b* видалити нижню стінку, яка є верхньою для комірки *a*:

```
else b.WallBottom = false;
```

Аналогічно видаляються стінки, якщо поточна і обрана клітинки є сусідніми по горизонталі (співпадає координата *x*):

```
if (a.X > b.X) a.WallLeft = false;
else b.WallLeft = false;
```

Якщо запустити зараз проект, можна побачити, що лабіринт вже майже повністю правильно генерується. Треба лише прибрати зайві стінки зверху і справа в методі *GenerateMaze()*:

```
for (int x = 0; x < cells.GetLength(0); x++)
{
    cells[x, Height - 1].WallLeft = false;
}

for (int y = 0; y < cells.GetLength(1); y++)
{
    cells[Width - 1, y].WallBottom = false;
}

RemoveWallsWithBacktracker(cells);
```

Тепер лабіринт виглядає цілком нормально (рис. 3.4). При кожному запуску конфігурація лабіринту змінюється. Завдяки колайдерам, які ми налаштували раніше для персонажа і стінок, по лабіринту можна вже ходити нашим ігровим персонажем. Але поки що ходити нема куди, оскільки немає фінішної позиції.

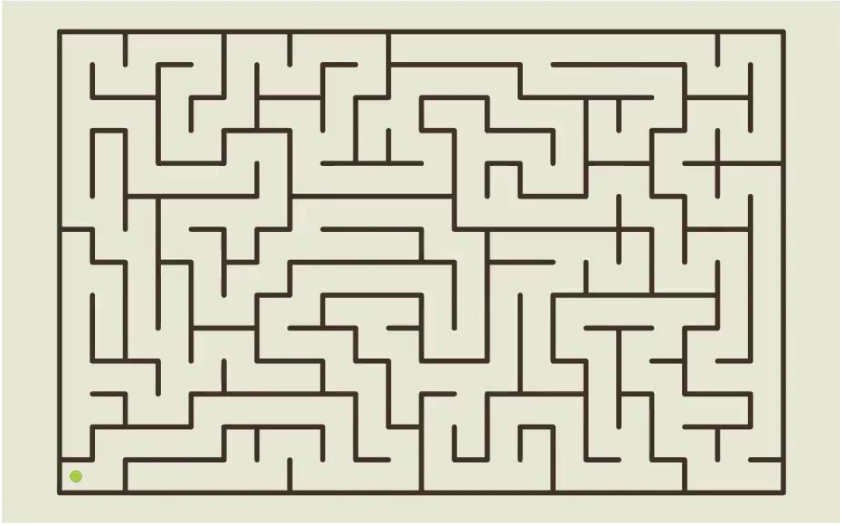

Рис. 3.4 – 2D-лабіринт, згенерований алгоритмом рекурсивного повернення

3.5 Формування в лабіринті фінішної позиції

Реалізуємо функцію *PlaceMazeExit()*, яка буде визначати фінішну (цільову) позицію у вигляді виходу (відсутньої стінки) на краю лабіринту. Функція прийматиме об’єкт лабіринту на вході, а на виході повертатиме позицію фінішу:

```
private Vector2Int PlaceMazeExit(MazeGeneratorCell[,] maze)
```

Щоб визначити, де саме краще поставити фініш, введемо ще одну властивість в класі комірки *MazeGeneratorCell* – відстань у кроках від старту до цієї комірки:

```
public int DistanceFromStart;
```

Ця відстань допоможе нам поставити фініш подалі від старту – так буде цікавіше. Якщо б ми обирали фініш випадково, могло б статися, що він опинився прямо біля старту.

На початку генерування лабіринту відстань від старту є нульовою:

```
current.DistanceFromStart = 0;
```

Для будь-якої іншої комірки в процесі генерування лабіринту відстань до старту дорівнюватиме поточній довжині стеку:

```
chosen.DistanceFromStart = stack.Count;
```

Ще простіше в процесі просування в прямому напрямку просто нарощувати цю змінну з кожним кроком (це також ефективніше з точки зору використання ресурсів):

```
chosen.DistanceFromStart = current.DistanceFromStart + 1;
```

Для визначення найвіддаленішої комірки від старту введемо в функцію *PlaceMazeExit()* змінну *furthest*:

```
MazeGeneratorCell furthest = maze[0, 0];
```

Також врахуємо, що фініш може знаходитися лише в комірках, які розташовані у зовнішніх рядках або стовпцях. Відповідно, треба пройтися циклом по всіх краях лабіринту.

Спочатку перевіримо усі комірки в верхньому та нижньому рядках. Якщо поточна комірка знаходиться далі від старту, ніж найвіддаленіша на даний момент, привласнюємо дану комірку змінній *furthest*:

```
        for (int x = 0; x < maze.GetLength(0); x++)
        {
            if (maze[x, Height - 2].DistanceFromStart >
furthest.DistanceFromStart) furthest = maze[x, Height - 2];
            if (maze[x, 0].DistanceFromStart > furthest.DistanceFromStart)
furthest = maze[x, 0];
        }
```

Аналогічно перевіряємо усі комірки в лівому та правому стовпцях:

```
        for (int y = 0; y < maze.GetLength(1); y++)
        {
            if (maze[Width - 2, y].DistanceFromStart >
furthest.DistanceFromStart) furthest = maze[Width - 2, y];
            if (maze[0, y].DistanceFromStart > furthest.DistanceFromStart)
furthest = maze[0, y];
        }
```

В подальшому можна буде вдосконалити алгоритм, щоб брати для фінішу не найвіддаленішу комірку, а якусь середню, або ж випадкову з декількох віддалених, щоб гравцеві було не надто нудно проходити лабіринт, а сам він при цьому був відносно складний на вигляд. Також можна буде встановлювати фініш в залежності від рівня складності, заданого в налаштуваннях гри.

Після того, як комірка для фінішу знайдена, треба прибрати відповідну стінку:

```
        if (furthest.X == 0) furthest.WallLeft = false;
        else if (furthest.Y == 0) furthest.WallBottom = false;
        else if (furthest.X == Width - 2) maze[furthest.X + 1,
furthest.Y].WallLeft = false;
        else if (furthest.Y == Height - 2) maze[furthest.X, furthest.Y +
1].WallBottom = false;
```

Якщо найвіддаленіша комірка знаходиться у правому стовпці, треба видалити її праву стінку. Проте, оскільки в кожної нашої комірки є лише ліва та нижня стінки, прибираємо ліву стінку її правого сусіда (в стовпці сітки, який ми не використовуємо).

Залишилось повернути з функції *PlaceMazeExit()* координати фінішу:

```
        return new Vector2Int(furthest.X, furthest.Y);
```

В результаті генерується лабіринт з фінішем (виходом) (рис. 3.5).

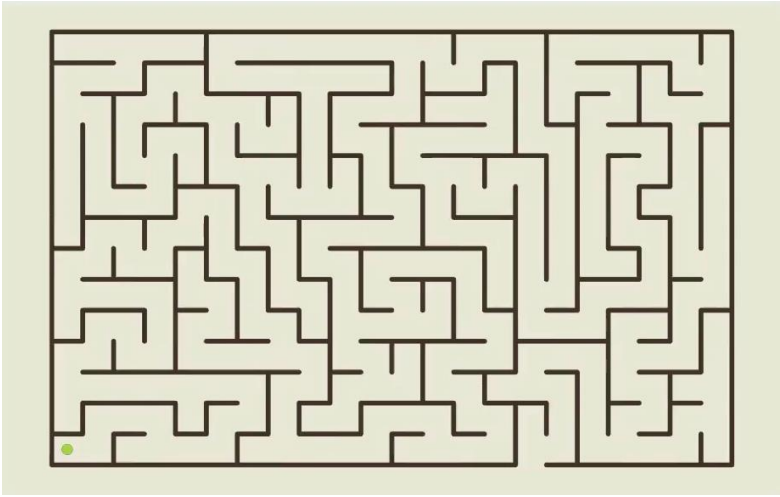

Рис. 3.5 – Згенерований двовимірний лабіринт з виходом

3.6 Реалізація функції підказки для проходження лабіринту

За технічним завданням гравцеві мають бути доступні підказки стосовно правильного проходження лабіринту. Для реалізації підказок нам знадобиться алгоритм проходження лабіринту, а також певна візуальна реалізація підказки. Реалізуємо на першому етапі підказку у вигляді суцільної ламаної лінії зеленого кольору, яка буде з'єднувати поточне положення гравця з виходом. Цю лінію можна буде вмикати по запиту підказки від гравця на невеликий час.

Лінію проходження буде представляти в нас ігровий об'єкт *Hint*. Додаємо до нього компонент *Line Renderer*, щоб налаштувати параметри візуалізації: товщину, колір тощо (рис. 3.6). Для згладжування ламаної лінії в кутах встановлюємо параметр *Corner Vertices* у значення 5.

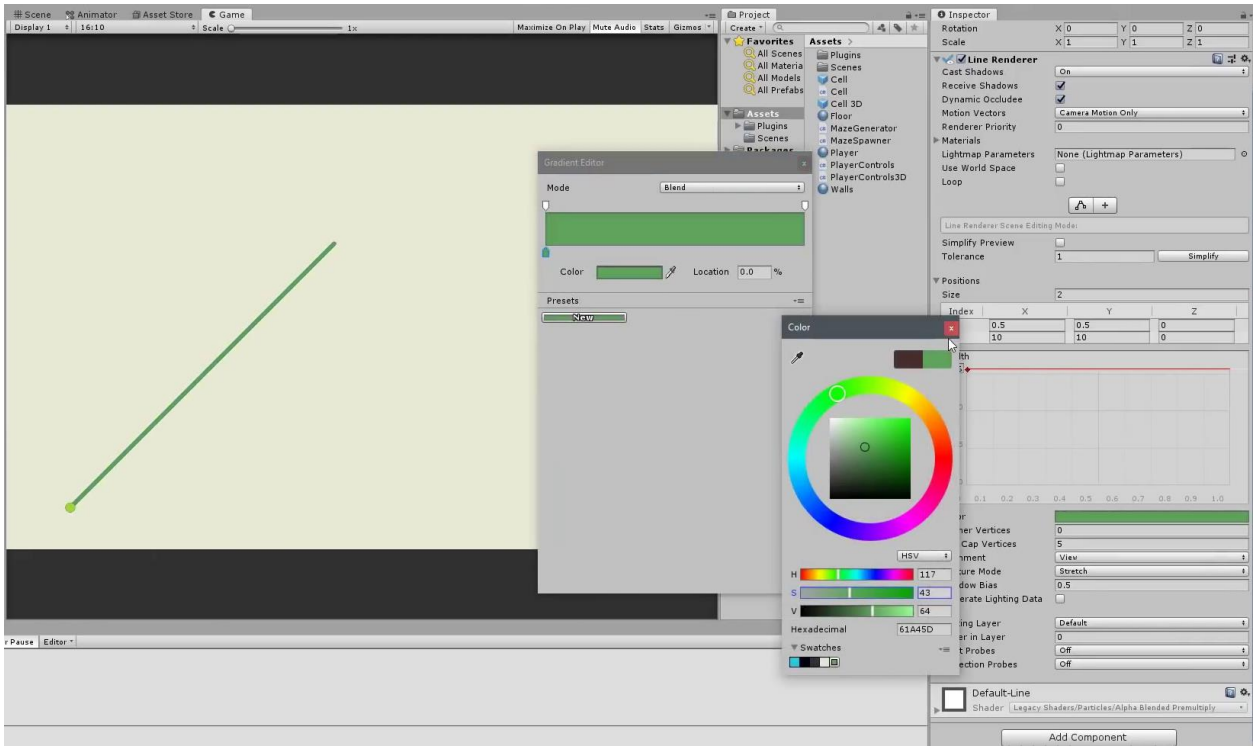

Рис. 3.6 – Налаштування параметрів візуалізації лінії підказки

Також додаємо до об'єкту *Hint* компонент скрипта *HintRenderer*, який буде прокладати лінію. В цьому скрипті (і, відповідно, класі) нам

знадобиться сам лабіринт у вигляді масиву комірок, який є об'єктом класу *Maze* і повертається методом *GenerateMaze()* класу *MazeGenerator*. Цей об'єкт також доступний як поле класу *MazeSpawner*, тому пропишемо увесь об'єкт *MazeSpawner* як поле класу *HintRenderer*:

```
public class HintRenderer : MonoBehaviour
{
    public MazeSpawner MazeSpawner;
```

Алгоритм побудови шляху від старту до фінішу досить простий. Раніше при побудові лабіринту ми прораховували кількість кроків від початку до поточної комірки, щоб знайти найвіддаленішу комірку. Це значення записувалося у поле *DistanceFromStart* програмного об'єкту комірки *MazeGeneratorCell*. Завдяки цьому ми можемо йти тепер назад від відомого нам фінішу по комірках, значення *DistanceFromStart* в яких кожен раз на 1 менше за відповідне значення попередньої комірки. Поки що реалізуємо такий варіант підказки. Побудова шляху від довільної комірки, де знаходиться керований персонаж, до фінішу вимагатиме реалізації за окремим алгоритмом. Але навіть в такому випадку наявність шляху від старту до фінішу, скоріше за все, зможе йому допомогти.

Для реалізації лінії правильного шляху напишемо в класі *HintRenderer* метод *DrawPath()*. Перш за все, в ньому треба отримати з об'єкту породжувача лабіринту *MazeSpawner* об'єкт лабіринту *maze*, а з нього – координати фінішної комірки:

```
    public void DrawPath()
    {
        Maze maze = MazeSpawner.maze;
        int x = maze.finishPosition.x;
        int y = maze.finishPosition.y;
        Vector2Int currentposition = maze.finishPosition;
```

Створимо тепер список позицій, які формуватимуть шлях між фінішем і стартом:

```
        List<Vector3> positions = new List<Vector3>();
```

Далі циклом проходимось по всіх комірках, поки не дійдемо до стартової позиції:

```
while (currentPosition != Vector2Int.zero)
{
    positions.Add((Vector2)currentPosition);
    MazeGeneratorCell currentCell = maze.cells[currentPosition.x,
        currentposition.y];
```

Спочатку перевіряємо сусіда зліва: що він існує, що ми до нього можемо пройти (у поточної клітинки немає лівої стінки), а також що дистанція від старту в неї на 1 менша за аналогічну дистанцію поточної клітинки. Якщо усі умови виконуються, переходимо вліво:

```
if (currentposition.x > 0 &&
    !currentCell.WallLeft &&
    maze.cells[currentPosition.x - 1,
        currentPosition.y].DistanceFromStart ==
        currentCell.DistanceFromStart - 1)
{
    currentPosition.x--;
}
```

Аналогічно перевіряємо нижню клітинку і, якщо підходить вона, переходимо до неї:

```
else if (currentPosition.y > 0 &&
    ! currentCell.WallBottom &&
    maze.cells[currentPosition.x,
        currentPosition.y - 1].DistanceFromStart ==
        currentCell.DistanceFromStart - 1)
{
    currentPosition.y--;
}
```

Так само опрацьовуємо праву і верхню клітинки:

```
else if (currentPosition.x < maze.cells.GetLength(0) - 1 &&
    !maze.cells(currentPosition.x + 1,
        currentPosition.y).WallLeft &&
        maze.cells(currentPosition.x + 1,
        currentPosition.y).DistanceFromStart ==
        currentCell.DistanceFromStart - 1)
{
    currentPosition.x++;
}
else if (currentPosition.y < maze.cells.GetLength(1)  1 &&
    !maze.cells[currentPosition.x,
        currentPosition.y + 1].WallBottom &&
```

```
            maze.cells(currentPosition.x,
            currentPosition.y + 1).DistanceFromStart ==
            currentCell.DistanceFromStart - 1)
        {
            currentPosition.y++;
        }
```

В результаті сформований список позицій на шляху від фінішу до старту. Цей список ми й передаємо компоненту *LineRenderer* для промальовування лінії підказки:

```
componentLineRenderer.positionCount = positions.Count;
componentLineRenderer.SetPositions(positions.ToArray());
```

Функцію *DrawPath()* можна викликати в скрипті *MazeSpawner* для перевірки правильності її роботи:

```
HintRenderer.DrawPath();
```

Тепер при запуску проекту бачимо зелену лінію підказки, тобто все коректно працює (рис. 3.7).

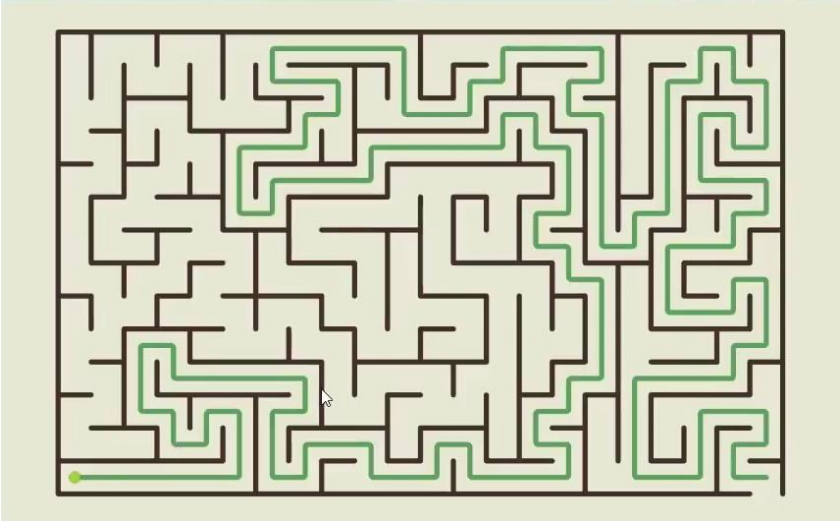

Рис. 3.7 – Лінія підказки, що показує шлях від старту до фінішу

3.7 Реалізація тривимірного лабіринту

Тривимірний лабіринт у грі буде генеруватися за тим самим алгоритмом і, відповідно, мати ті ж характеристики, що й двовимірний лабіринт, створений раніше.

Для реалізації тривимірного лабіринту створюємо в проекті 3D-сцену шляхом копіювання 2D-сцени під назвою *Maze3D*.

Основну камеру переключаємо на перспективну проекцію і повертаємо так, щоб вона бачила початок сцени (рис. 3.8).

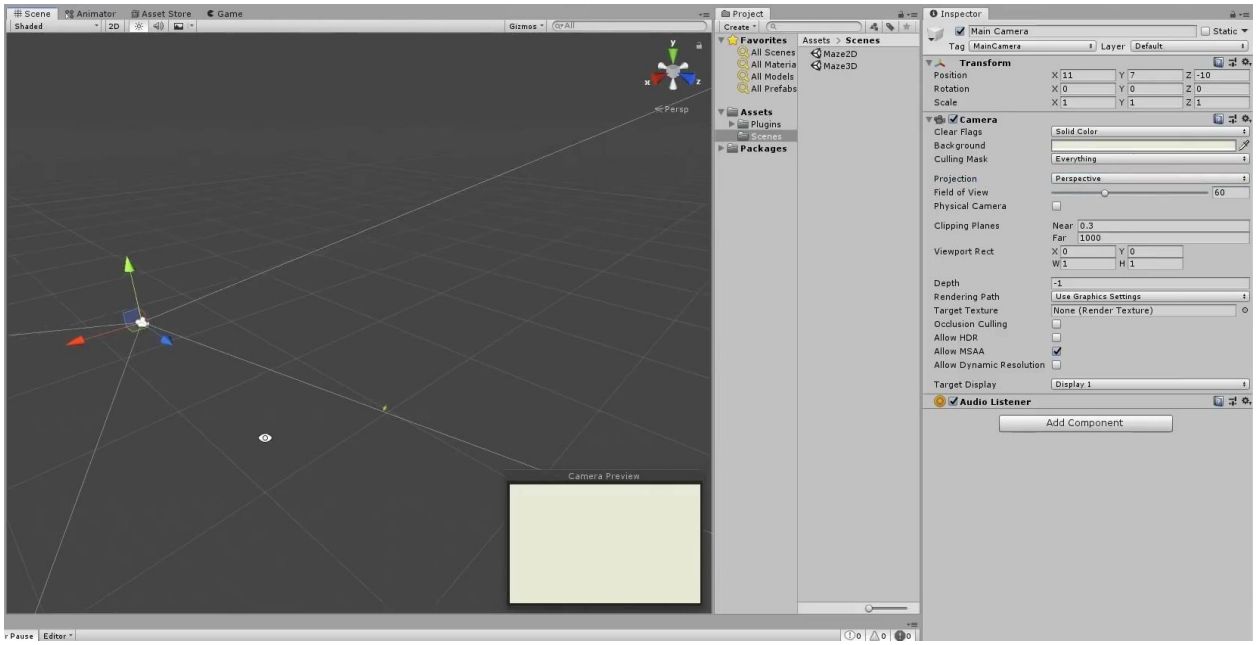

Рис. 3.8 – Налаштування камери на 3D-сцені

Як і раніше для 2D-лабіринту, нам знадобиться *Game Object* для однієї комірки лабіринту – *Cell*. В даному випадку він має бути паралелепіпедом з трьома стінками, одна з яких є підлогою. Створюємо в середині об'єкту *Cell* ще один об'єкт – *Cube*. Камеру переключимо в режим *Directional Light*, щоб було видно його грані. Перейменовуємо об'єкт *Cube* на *Floor* – це буде підлога нашої комірки. Зменшуємо висоту куба (розмір за віссю *y*) до 0,2 одиниць Unity – це буде товщина підлоги, а розміри за осями *x* та *z* встановлюємо у 10 одиниць Unity (рис. 3.9).

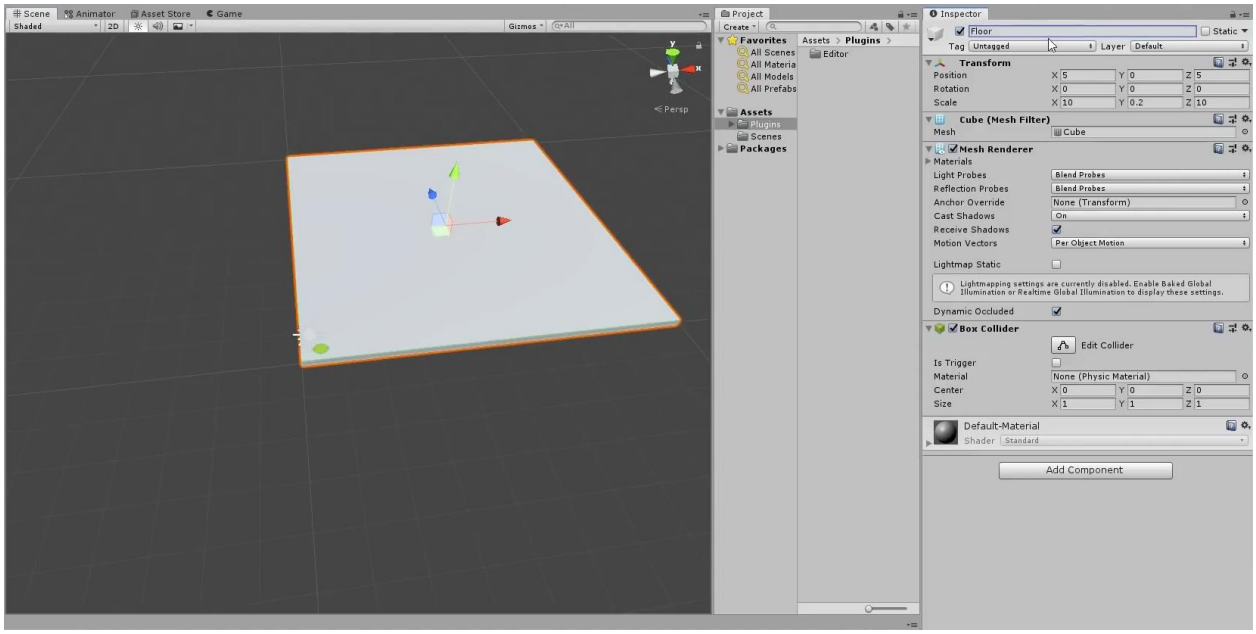

Рис. 3.9 – Об’єкт *Floor* – підлога комірки тривимірного лабіринту

Тепер зробимо дві копії цього об’єкту для стінок (рис. 3.10) і зробимо їх підоб’єктами об’єкту *Cell*. Якщо дивитись на тривимірний лабіринт зверху, одна стінка буде лівою, а друга – нижньою, як і у двовимірному лабіринті, тому назвемо їх *Wall Left* і *Wall Bot*. Товщину стінок встановлюємо у 0,5 одиниць, висоту – у 3 одиниці, а ширину – у 10 одиниць Unity.

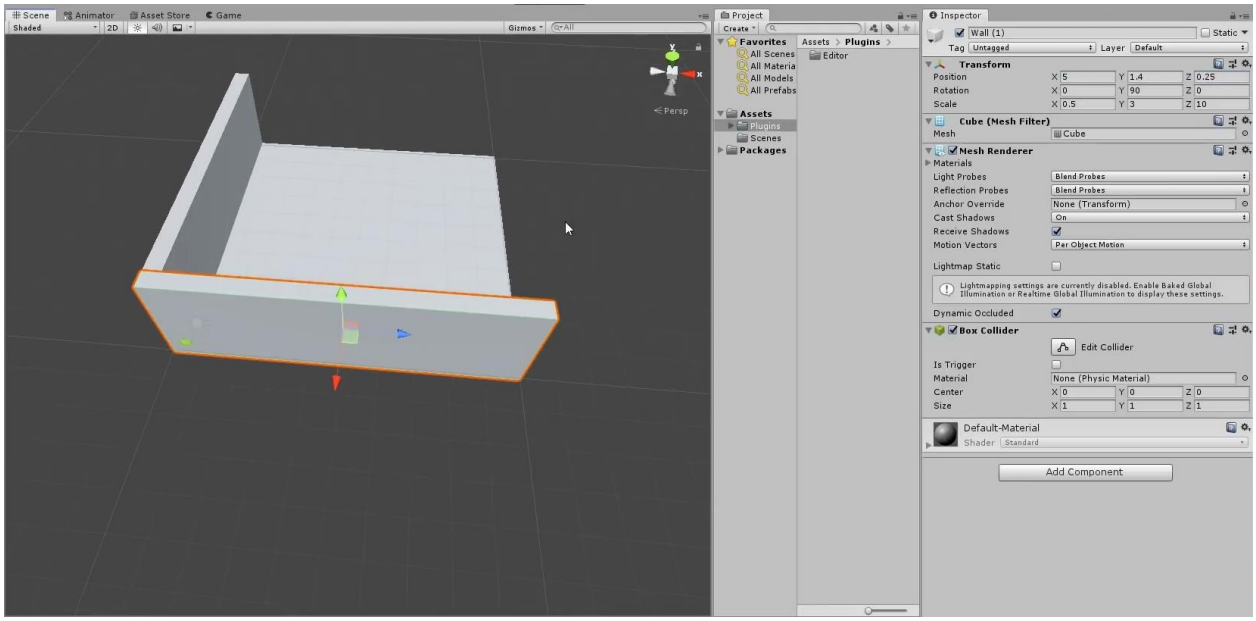

Рис. 3.10 – Комірка тривимірного лабіринту з підлогою та двома стінками

Далі додаємо до ігрового об’єкту *Cell* компонент скрипта *Cell*, який розробили раніше, і зв’язуємо підоб’єкти стінок з полями *Wall Left* і *Wall Bottom* відповідно.

Щоб комірку можна було розмножувати, зробимо ігровий об’єкт *Cell* префабом і приберемо його зі сцени, а щоб не було конфлікту імен з префабом двовимірної комірки, перейменуємо його на *Cell 3D*. Далі зв’яжемо цей префаб зі скриптом *Maze Spawner*.

Усі процедури, які відповідають за генерацію лабіринту, вже написані і мають працювати. Проте якщо запустити зараз проект, лабіринт генеруватиметься неправильно: як видно на рис. 3.11, він нагадує якийсь недобудований багатоповерховий будинок. Це відбувається тому, що лабіринт зараз будується за осями *x* та *y*, а в нашій сцені вісь *y* відповідає за вертикаль.

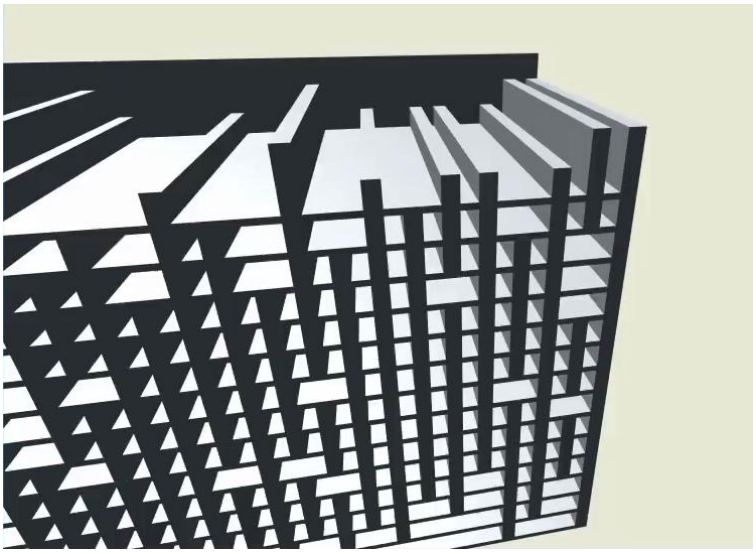

Рис. 3.11 – Тривимірний лабіринт, який генерується вгору

Нам зараз треба, щоб лабіринт будувався за осями *x* та *z*. Для цього можна змінити вектор в коді скрипту *MazeSpawner*, проте ми реалізуємо більш універсальне рішення: пропишемо в ньому змінну *CellSize* типу *Vector3* з розміром однієї комірки в умовних модельних одиницях: 1 і 1 одиниця по осях *x* та *y* і 0 по осі *z*:

```
public Vector3 CellSize = new Vector3(1,1,0);
```

Коли ми будемо створювати комірку, будемо брати не просто *x*, а домножувати його на *CellSize.x* і так далі. Що стосується змінної *z*, то у двовимірній моделі, яка в нас запрограмована, її немає. Тому замість неї ми також будемо брати змінну *y* і домножувати на розмір по осі *z*:

```
Cell c = Instantiate(CellPrefab, new Vector3(x * CellSize.x,
    y * CellSize.y, y * CellSize.z), Quaternion.identity);
```

Нам треба задати ненульовий *z*, щоб лабіринт розростався саме за цією віссю.

Далі встановимо поля *Cell Size* в компоненті *Maze Spawner* у відповідності до геометричних розмірів об'єкту: *X* = 10, *Y* = 0, *Z* = 10.

Тепер комірки лабіринту стають на свої місця (рис. 3.12).

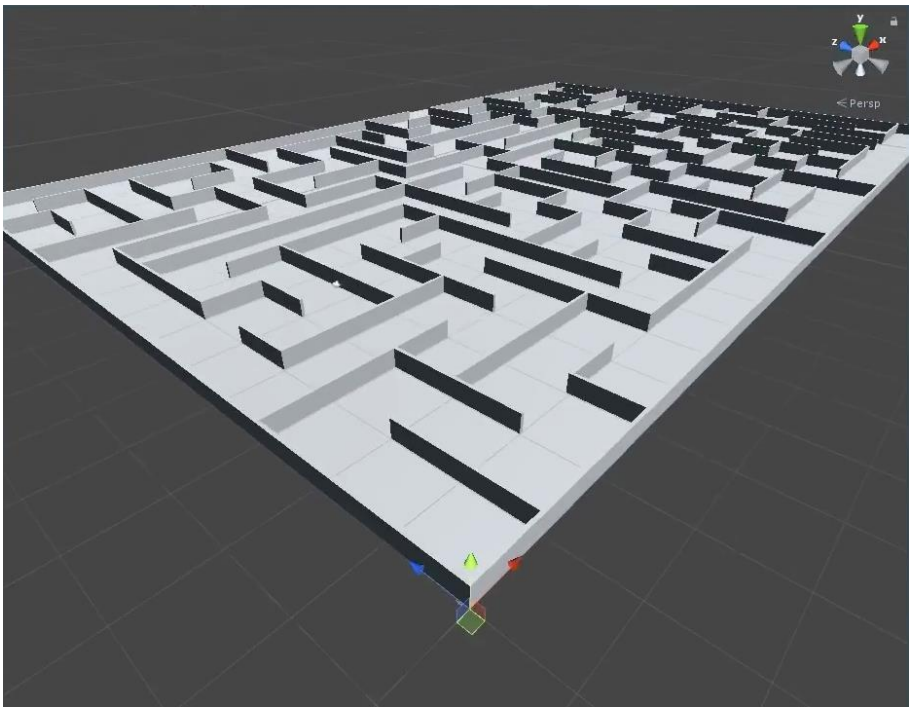

Рис. 3.12 – Тривимірний лабіринт, згенерований у горизонтальній площині

Налаштовуємо камеру так, щоб вона показувала початок лабіринту, як на рис. 3.12.

3.8 Створення ігрового персонажа для тривимірного лабіринту

Об’єкт гравця *Player* для 3D-лабіринту створюємо у вигляді кульки (*Sphere*) (рис. 3.13). Додаємо до нього компоненти *Mesh Renderer* для промальовування зовнішнього вигляду кульки, *Sphere Collider* для визначення границь взаємодії і *Rigidbody* для фізики руху.

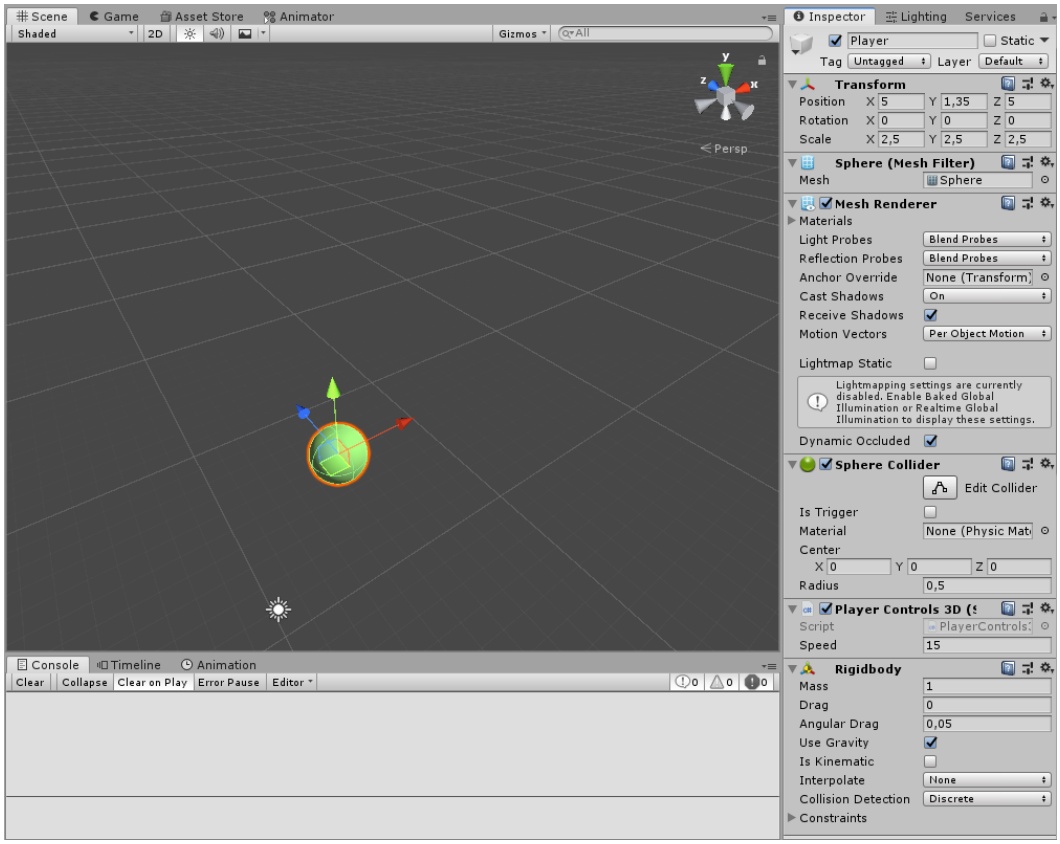

Рис. 3.13 – Об’єкт гравця для тривимірного лабіринту

Для управління гравцем створюємо скрипт *Player Controls 3D*:

```
public class PlayerControls3D : MonoBehaviour
{
    public float Speed = 2;

    private Rigidbody componentRigidbody;

    private void Start()
    {
        componentRigidbody = GetComponent<Rigidbody>();
    }

    private void Update()
```

```
    {
        componentRigidbody.velocity = Vector2.zero;
        if (Input.GetKey(KeyCode.LeftArrow)) componentRigidbody.velocity +=
Vector3.left * Speed;
        if (Input.GetKey(KeyCode.RightArrow)) componentRigidbody.velocity +=
Vector3.right * Speed;
        if (Input.GetKey(KeyCode.UpArrow)) componentRigidbody.velocity +=
Vector3.forward * Speed;
        if (Input.GetKey(KeyCode.DownArrow)) componentRigidbody.velocity +=
Vector3.back * Speed;
    }
}
```

Як видно, код майже ідентичний управлінню двовимірним гравцем, за виключенням того, що тут клавіши-стрілки *Up* і *Down* впливатимуть на властивості *forward* і *back* компоненту *Rigidbody*. Швидкість руху залежить від параметра *Speed*, який можна налаштовувати.

3.9 Реалізація лінії підказки для тривимірного лабіринту

Оскільки загальний алгоритм побудови лінії підказки не залежить від геометричної розмірності лабіринту, скопіюємо об’єкт *Hint* зі сцени *Maze2D* у сцену *Maze3D*. Налаштуємо знову посилання на об’єкт *MazeSpawner* у скрипті *HintRenderer* і на об’єкт *Hint* у скрипті *MazeSpawner*, щоб все працювало.

Також знову треба задати площину промальовування лінії підказки і масштабувати її у відповідності до розмірів лабіринту в одиницях Unity:

```
    public void DrawPath()
    {
        Maze maze = MazeSpawner.maze;
        int x = maze.finishPosition.x;
        int y = maze.finishPosition.y;
        List<Vector3> positions = new List<Vector3>();

        while ((x != 0 || y != 0) && positions.Count < 10000)
        {
            positions.Add(new Vector3(x * MazeSpawner.CellSize.x, y *
MazeSpawner.CellSize.y, y * MazeSpawner.CellSize.z));

            MazeGeneratorCell currentCell = maze.cells[x, y];

            if (x > 0 &&
                !currentCell.WallLeft &&
                maze.cells[x - 1, y].DistanceFromStart <
currentCell.DistanceFromStart)
```

```
        {
            x--;
        }
```

Результат можна побачити на рис. 3.14.

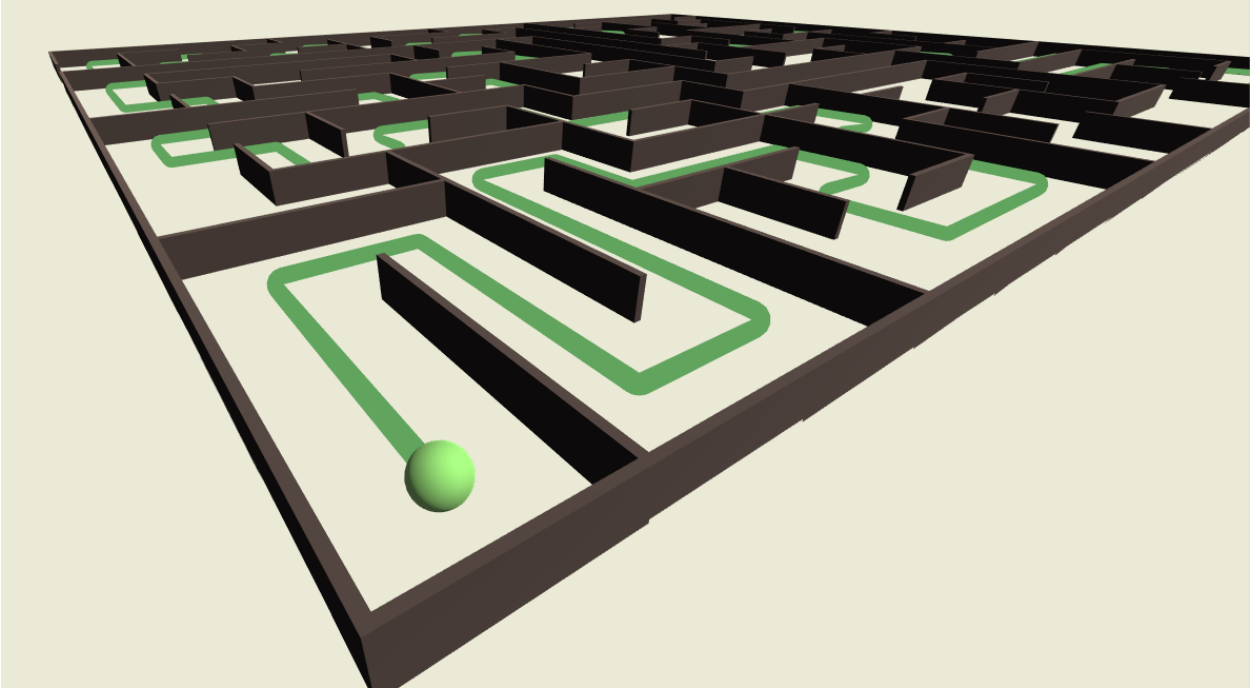

Рис. 3.14 – Лінія підказки у тривимірному лабіринті

3.10 Слідування камери за ігровим персонажем

Оскільки тривимірний лабіринт не поміщається увесь на екрані, треба, щоб камера переміщувалася за гравцем у горизонтальній площині – тоді гравець завжди бачитиме найближче своє оточення.

Для цього створимо новий ігровий об’єкт *FollowCamera* і однойменний скрипт такого змісту:

```
using UnityEngine;

public class FollowCamera : MonoBehaviour
{
    public GameObject player; // об'єкт гравця
    [SerializeField]
    public Transform target;
    private Vector3 offset;

    void Start()
```

```
    {
        offset = target.transform.position - player.transform.position;
    }

    void LateUpdate()
    {
        target.transform.position = player.transform.position + offset;
    }
}
```

З полем *player* в інспекторі зв'яжемо наш ігровий об'єкт *Player*, а з полем *target* – об'єкт камери *Main Camera*. Спочатку (в обробнику *Start()*) знаходимо різницю між поточним положенням камери і гравця. В методі *LateUpdate()* встановлюємо положення камери у відповідності до поточного положення гравця.

В результаті камера постійно слідуватиме за ігровим персонажем. При переміщенні персонажа сам він буде знаходитися в одній точці екрану, а лабіринт зсуватиметься відносно екрану.

Висновки до третього розділу

За результатами роботи, виконаної в даній частині, можна зробити наступні висновки:

1. Створений проект Unity з варіантами 2D і 3D-сцен і прототипами ігрового персонажу.

2. Запрограмоване процедурне генерування лабіринту по алгоритму *Recursive backtracker* з використанням 2D і 3D-комірок, реалізованих у вигляді префабів.

3. Реалізована лінія підказки, яка показує правильний шлях від початкової клітинки лабіринту до виходу з нього. Також реалізований рух камери за гравцем.

Далі проект має виступити в якості основи для подальшої реалізації гри за дизайн-документом: з багатьма рівнями, що ускладнюються по мірі проходження, функціями нарахування балів, обмеження кількості «життів», створення лабіринтів інших конфігурацій тощо.

РОЗДІЛ 4
ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ПРОЕКТУ

4.1 Організаційне і маркетингове обґрунтування проекту

Метою цієї кваліфікаційної роботи є розробка програмного продукту, а саме – комп'ютерної гри у жанрі «головоломка». Даний програмний продукт був створений за допомогою ігрового рушія і середовища розробки Unity.

Проект відноситься до сфери розваг, відповідно, його характер є розважальним. Цільове завдання проекту є комбінованим: окрім розважання гра розвиває увагу, пам'ять, орієнтацію у просторі та інші когнітивні здібності гравця. За своїм масштабом він належить до категорії малих проектів, які характеризуються невеликим обсягом коду і обмеженими ресурсами, необхідними для його роботи. По термінах реалізації проект є короткостроковим і має невеликий життєвий цикл (до 3-х років). По складності проект відноситься до проектів середньої складності.

Комп'ютерна гра, яка розробляється в даному проекті, відноситься до категорії казуальних ігор. Це означає, що для неї не потрібні спеціальні пристрої, такі як ігрові консолі або потужні відеокарти, також вона має прості, інтуїтивно зрозумілі правила та досить короткий цикл проходження рівнів.

Створення простих казуальних відеоігор визначається як одна з важливих тенденцій геймінг-індустрії, яка стає все більш вагомою в сучасному світі. Казуальні ігри здобувають все більшу популярність серед геймерської спільноти: обсяг світового ринку казуальних ігор у 2023 році склав понад 3 мільярди доларів, і очікується, що він зросте щонайменше вдвічі протягом 2024-25 років. Посилення цієї тенденції зумовлене в першу

чергу збільшенням кількості смартфонів і мобільних пристроїв, які можуть використовуватися для ігор.

Казуальні ігри привертають увагу розробників та інвесторів завдяки високій монетизації та ефективному використанню рекламних можливостей. Через це такі ігри є одними з найбільш прибуткових видів комп'ютерних ігор. Також привабливість казуальних ігор полягає в тому, що вони привертають найбільш різноманітну та широку аудиторію будь-якого віку. Згідно з аналітичним агентством DFC Intelligence, на сьогодні у комп'ютерні ігри грають понад 3,7 мільярда людей у всьому світі.

На підставі вказаних фактів можна прийти до висновку, що розробка комп'ютерної гри в даному проекті має великий шанс стати комерційно успішною. Основним інструментом розповсюдження цього продукту є міжнародні спеціалізовані майданчики і онлайн-магазини, а потенційними споживачами – будь-які користувачі, які час від часу грають у комп'ютерні ігри, насамперед – мобільні. Очікуваною конкурентною перевагою проекту є приваблива для споживача цінова політика. Планується розповсюдження продукту шляхом продажу його компанії-замовнику або через самостійне розміщення на ігрових платформах, таких як Steam, itch.io та Kongregate. Оскільки гра не є технологічно складною і була розроблена переважно однією людиною протягом короткого часу, планується встановлення мінімально можливої ціни на всіх платформах: 0,49 долара США на платформі Steam і 1 долар США на платформі itch.io. Платформа Kongregate працює на основі прибутку від реклами, проте вбудовувати рекламу в даний продукт не планується.

Для досягнення поставленої мети в роботі було проведено аналіз сучасних тенденцій у розробці програмного забезпечення, вивчено інструменти розробки програмних застосунків та проаналізовано деякі подібні проекти конкурентів (розділ 1).

4.2 Оцінка науково-технічної ефективності проекту

Умови відкритої ринкової економіки сприяють розширенню асортименту критеріїв оцінки науково-технічних розробок. Це призводить до зростання числа основних категорій ефективності науково-дослідницьких та розробницьких робіт, які необхідно визначити для проведення цієї оцінки. До них належать:

1. Науково-технічний ефект, що виявляється у підвищенні науково-технічного рівня, поліпшенні характеристик техніки і технологій, що випливає з відкриття нових законів та закономірностей природи, а отже, і нових технологічних засобів виробництва речовин, матеріалів та видів продукції;

2. Економічний ефект від розробки та впровадження комп'ютерних ігор полягає в забезпеченні економічних результатів для кожного виробничого суб'єкта та в цілому для економіки країни. Економічна вигода проекту відображає вплив його результативності на розвиток галузей, регіонів та підприємств, які беруть участь у впровадженні новітніх технологічних рішень;

3. Соціальний ефект, що відображає зміни умов діяльності людини в суспільстві. Його прояв спостерігається у змінах характеру та умов праці, підвищенні рівня життя населення, поліпшенні умов побуту, розширенні можливостей духовного розвитку особистості та в стані довкілля;

4. Маркетинговий ефект відображає потреби ринку в наукових дослідженнях і розробках та можливість їх реалізації.

Науково-технічний ефект (НТЕ) виконаних прикладних робіт оцінюємо за показником $O_{НТР}$, який відображає ступінь досягнення максимально можливого рівня (1):

$$O_{НТЕ} = K^{Ф}_{НТЕ} / K^{П}_{НТЕ}, \qquad (4.1)$$

де $K^{Ф}_{НТЕ}$ – показник (коефіцієнт) фактичного рівня науково-технічної ефективності;

$K^{П}_{НТЕ}$ – показник (коефіцієнт) потенційно можливого рівня науково-технічної ефективності (дорівнює одиниці).

Значення показника $K^{Ф}_{НТЕ}$ визначається на основі шкали експертних оцінок (табл. 4.1) [Б].

Таблиця 4.1

Шкала експертних оцінок

для виміру рівня науково-технічної ефективності проектів

№	Групи показників	Характеристика показників	Інтервал рейтингового числа	Коефіцієнт значущості показників
1	Науково-технічний рівень	Перевищує кращі світові аналоги	10	0,30
		Відповідає світовому рівню	7 – 9	
		Нижче кращих світових аналогів	5 – 6	
		Перевищує кращі вітчизняні аналоги	3 – 4	
		Відповідає вітчизняному рівню	1 – 2	
		Нижче вітчизняного рівня	0	
2	Перспективність	Першочергова значущість	8 – 10	0,25
		Значущий	5 – 7	
		Корисний	1 – 4	
3	Потенційний масштаб практичного використання	Світовий ринок	10	0,20
		Галузі національної економіки	7 – 9	
		Галузь (регіон)	3 – 6	
		Окремі підприємства (об’єднання)	1 – 2	
4	Ступінь вірогідності досягнення позитивних результатів	Великий	10	0,25
		Середній	5 – 9	
		Малий	1 – 4	

Предмет оцінки та аналоги, які порівнюються за тими ж показниками, представлені у вигляді зіставлення відхилень по значеннях кожного з цих показників, мають бути однаковими для порівнюваних варіантів.

Визначимо коефіцієнт науково-технічної ефективності на основі експертної оцінки рівня розробки. Для цього необхідно:

- створити перелік конкретних показників, необхідних для виміру рівня розробки;
- сформувати групу аналогів, які представлені на світовому та вітчизняному ринках;
- провести відповідні розрахунки для порівняння показників та визначення балів за таблицею 4.1.

Серед конкретних показників виокремлюються:

- для нової техніки: продуктивність, використання інженерних ресурсів на виробництво одиниці продукції, потреба у робітниках, які обслуговують обладнання, експлуатаційні витрати на одиницю продукції;
- для нових матеріалів та речовин: вміст корисних речовин для виробництва готової продукції, відсоток відходів в загальному обсязі переробленої сировини, вартість одиниці нового матеріалу, додаткові витрати на екологічну компенсацію;
- для нових технологій: якість виготовленої продукції, енергоефективність та трудомісткість виробництва, собівартість одиниці продукції.» [12]

На основі співставлення даних таблиці 4.2 встановлюємо бали по характеристиках чотирьох груп і на цій основі розраховуємо значення інтегрального показника *НТЕ*:

$$НТЕ = \Sigma\, Б_i \cdot К_i\,, \qquad (4.2)$$

де $i = 1 \div 4$,

$Б_i$ – рейтингові бали,

$К_i$ – коефіцієнт значущості показників.

Таблиця 4.2

Порівняльні показники для виконання оцінки НТЕ

ПОКАЗНИКИ	Варіанти технології	
	Розробленої	співвідносної (аналога)
Рівень новизни	Високий	Середній
Якість продукції	Висока	Середня
Споживання на 1 версію продукту – тепла, Гкал – електроенергії, кВт·годину – води, $м^3$	 6 800 3	 8 1200 4
Трудомісткість виробництва, людино-годин	90	60

Рівень науково-технічної ефективності НДДКР розраховано на основі наведених даних прикладу (таблиця 4.3).

Таблиця 4.3

Експертна оцінка і розрахунок величини інтегрального показника *НТЕ*

№	Групи показників	Рейтинг експертів			Середня за експертними оцінками	*НТЕ*
		1	2	3		
1	Науково-технічний рівень	6	7	6	6,33	6,33 · 0,3 = 1,9
2	Перспективність	7	7	8	7,33	7,33 · 0,25 = 1,83
3	Потенційний масштаб практичного використання	8	8	8	8	8 · 0,2 = 1,6
4	Ступінь вірогідності досягнення позитивних результатів	8	9	6	7,67	7,67 · 0,25 = 1,92
Всього						7,25

$$НТЕ = 6{,}33 \cdot 0{,}3 + 7{,}33 \cdot 0{,}25 + 8 \cdot 0{,}2 + 7{,}67 \cdot 0{,}25 =$$
$$= 1{,}9 + 1{,}83 + 1{,}6 + 1{,}92 = 7{,}25.$$

Отриманий результат слід порівняти з максимально можливим значенням, яке дорівнює 10 балам ($10 \cdot 0{,}3 + 10 \cdot 0{,}25 + 10 \cdot 0{,}2 + 10 \cdot 0{,}25$).

Отже, оцінка рівня НТЕ може бути зроблена за допомогою інтегрального коефіцієнта оцінки НТЕ ($К_{НТЕ}$):

$$К_{НТЕ} = \frac{НТЕ}{10} \cdot 100\ \% \qquad (4.3)$$

Підставляючи дані з таблиці 4.3, розрахуємо величину $К_{НТЕ}$:

$$К_{НТЕ} = \frac{7{,}25}{10} \cdot 100\ \% = 72{,}5\ \%.$$

Як бачимо, значення коефіцієнта науково-технічної ефективності перевищує середнє значення, яке складає 50 %. Відповідно, можемо зробити висновок про рівень науково-технічної ефективності на основі шкали:

- повністю задовольняє – від 50 до 60 %;
- достатній – від 61 до 80 %;
- досить високий – від 81 до 90 %;
- високий – від 91 до 100 %.

Отже, рівень науково-технічної ефективності застосованої технології можна визнати достатнім, і розроблений продукт пропонується випускати на ринок.

4.3 Розрахунок витрат на реалізацію і впровадження продукту

4.3.1 Розрахунок витрат на матеріали

Склад витрат на матеріали, необхідні для реалізації проекту, наведений в табл. 4.4.

Таблиця 4.4

Витрати на матеріали

Найменування матеріальних витрат	Одиниця виміру	Кіль-кість	Ціна за одиницю, грн.	Вартість, грн.
Папір 500 арк.	пачка	1	170	170
Фарба (картридж)	шт.	1	750	750
Флеш-накопичувач, 32 ГБ	шт.	1	130	130
Разом				1050
Транспортно-заготівельні витрати – 10%				105
Усього:				1155

4.3.2 Розрахунок основної заробітної плати

Таблиця 4.5

Витрати на заробітну плату

Найменування робіт	Трудомісткість робіт, дні	Місячний оклад, грн	Денна заробітна плата, грн	Заробітна плата
1. Програмування	25	15000	682	17045
2. Керівництво і контроль	45	20000	909	40909
3. Розробка дизайну	20	15000	682	13636
Усього:				71591

Додаткова заробітна плата враховує оплату чергових відпусток, премії, інші доплати. Приймається в обсязі 10 % від основної заробітної плати:

$$ЗП_{дод} = ЗП_{осн} \cdot 0{,}10. \quad (4.4)$$

$$ЗП_{дод} = 71591 \cdot 0{,}10 = 7159 \text{ грн.}$$

Єдиний соціальний внесок (*ЄСВ*) приймається в розмірі 22 % від суми основної і додаткової заробітної плати.

$$ЄСВ = (ЗП_{осн} + ЗП_{дод}) \cdot 0{,}22. \quad (4.5)$$

$$ЄСВ = (71591 + 7159) \cdot 0{,}22 = 17\,325 \text{ грн.}$$

Накладні витрати, які також необхідно врахувати, включають адміністративні, загальновиробничі витрати та витрати на збут. Зазвичай їх визначають у розмірі 50 % від основної заробітної плати:

$$НВ = 0{,}5 \cdot ЗП_{осн} \quad (4.7)$$

$$НВ = 0{,}5 \cdot 71591 = 35795 \text{ грн.}$$

На підставі здійснених розрахунків складається калькуляція планової собівартості програмного продукту.

4.3.3 Розрахунок собівартості і ціни продукту

Витрати для розрахунку собівартості програмного продукту наведені в табл. 4.6.

Таблиця 4.6

Витрати для розрахунку собівартості програмного продукту

Найменування статей витрат	Сума витрат, грн.	Питома вага, %
1. Матеріали	1155	0,9 %
2. Основна заробітна плата	71591	53,8 %
3. Додаткова заробітна плата	7159	5,4 %
4. Єдиний соціальний внесок	17325	13 %
5. Накладні витрати	35795	26,9 %
Разом:	133025	100 %

Ціна програмного продукту визначається по формулі [В]:

$$Ц = К \cdot С + Пр, \quad (4.8)$$

де $С$ – витрати на розробку програмної продукції (планова собівартість), грн;

$К$ – коефіцієнт обліку витрат на виготовлення досвідченого зразка програмного продукту як продукції виробничо-технічного призначення ($К$ = 1,1);

$Пр$ – розмір нормативного прибутку, розрахованого по формулі:

$$Пр = (С – С_м) \cdot Рн / 100, \quad (4.9)$$

де: $Рн$ – плановий рівень рентабельності у відсотках (25 %);

$С_м$ – матеріальні витрати.

Прибуток у ціні програмного продукту становить:

$$П = (133025 – 1155) \cdot 0,25 = 32968 \text{ грн.}$$

Ціна програмного продукту становить:

$$Ц = 133025 \cdot 1,1 + 32968 = 179296 \text{ грн.}$$

4.3.4 Розрахунок капітальних витрат

Для оцінки економічної доцільності проекту визначимо капітальні та поточні витрати, пов’язані з використанням програмного продукту. Розрахунок капітальних витрат, пов’язаних з впровадженням програмного продукту, здійснюємо по формулі:

$$К_2 = К_{пп} + К_п + К_{ко} + К_{во}, \quad (4.10)$$

де $К_{пп}$ – ціна програмного продукту;

$К_п$ – передвиробничі витрати;

$К_{ко}$ – вартість комп’ютерного устаткування;

$К_{во}$ – вартість допоміжного устаткування, необхідного для надійної роботи продукту. [4]

Вартість програмного продукту становить 179296 грн.

Передвиробничі витрати $К_{п}$ охоплюють всі витрати, пов’язані з налагодженням і впровадженням продукту, включаючи постановку задач, їх алгоритмізацію, розробку, налагодження і впровадження програмного продукту. Приймаються у розмірі 100 % від вартості розробленого ПП:

$$К_{п} = 179296 \text{ грн.}$$

Вартість комп’ютера ($К_{ко}$) становить 30 000 грн.

Вартість допоміжного устаткування визначається укрупнено в розмірі 10 % від вартості комп’ютера.

$$К_{во} = 30\,000 \cdot 0{,}10 = 3000 \text{ грн.}$$

$$К_{2} = 179296 + 179296 + 30\,000 + 3000 = 391591 \text{ грн.}$$

Висновки до четвертого розділу

1. Виконане організаційне і маркетингове обґрунтування проекту продемонструвало актуальність реалізації проекту.

2. Аналіз коефіцієнта ефективності підтверджує, що впровадження даного продукту є вигідним. Відповідно, розробку програмного продукту можна вважати доцільною.

3. Проведений розрахунок витрат на реалізацію і впровадження продукту є фінансовим обґрунтуванням проекту.

РОЗДІЛ 5
ОХОРОНА ПРАЦІ НА РОБОЧОМУ МІСЦІ

Дана дипломна робота присвячена створенню комп'ютерної гри. Оскільки об'єкт розробки є типовим представником програмного забезпечення, а розробники ігор працюють за персональним комп'ютером (ПК) багато годин, вони мають слідувати правилам і рекомендаціям, що стосуються охорони праці з комп'ютером.

5.1 Загальні положення

Продуктивність та характер діяльності людини прямо пов'язані з виконанням конкретних робіт та ефективністю праці. Остання визначається як впливом людського фактору, так і використанням засобів виробництва, а також умовами, пов'язаними з технологією та організацією праці. В сучасній промисловій сфері більшість працівників займається роботою, пов'язаною з використанням комп'ютерної техніки. Працюючи за комп'ютером, людина стикається з різноманітними факторами, такими як електромагнітні поля, інфрачервоне та іонізуюче випромінювання, шум, вібрації та статична електрика.

Робота за комп'ютером супроводжується нервовим навантаженням та потребує значної ментальної напруги для операторів. Вона також вимагає високого рівня зорової активності та значного фізичного навантаження на руки під час взаємодії з клавіатурою. Раціональна конструкція та розташування елементів робочого місця мають велике значення для забезпечення оптимальної робочої позиції під час праці за комп'ютером.

Під час роботи з комп'ютером важливо дотримуватись належного режиму праці та відпочинку. В іншому випадку працівники можуть

відчувати незадоволення роботою, головний біль, роздратування, порушення сну, втому та відчувати біль в очах, спині, шиї та руках.

Обчислювальна техніка виділяє тепло, що може призвести до підвищення температури та зниження вологості у приміщенні.

Рівень шуму на робочому місці оператора не повинен перевищувати встановлені норми. Для зниження рівня шуму стіни та стеля в приміщенні, де знаходяться комп'ютери, повинні бути облицьовані матеріалами, що поглинають звук.

Оптичне випромінювання включає ультрафіолетове, видиме світло та інфрачервоне випромінювання.

В першу чергу, ультрафіолетове випромінювання впливає на шкіру та очі людини. Проте, аналіз досліджень робочих місць користувачів комп'ютерів показує, що у 86 % випадків ультрафіолетове випромінювання не було виявлено.

Світлове випромінювання, переважно, впливає на очі і може викликати втому та запалення райдужної оболонки. Однак ці симптоми швидко зникають і не спричиняють патологічних змін.

Електромагнітне випромінювання (ЕМВ) у радіочастотному діапазоні є основним джерелом ЕМВ в робочому середовищі, зокрема від монітора. Тому при обиранні місця для комп'ютера необхідно враховувати, що задня і бокові стінки можуть бути джерелом значно більшого ЕМВ, ніж сам екран.

Наукові дослідження свідчать про те, що радіочастотне випромінювання впливає на центральну нервову систему і є значним стрес-фактором.

Щоб зменшити вплив згаданих видів випромінювання, рекомендується використовувати монітори з низьким рівнем випромінювання, а також дотримуватись регламентованого режиму праці та відпочинку.

5.2 Способи зниження впливу шкідливих та небезпечних факторів при роботі з комп'ютером

Щодо зниження впливу шкідливих та небезпечних факторів під час роботи з комп'ютером, необхідно, зокрема, правильно розташовувати обладнання та електричні кабелі, щоб уникнути ризику ураження електричним струмом. Інші заходи, пов'язані з електробезпекою, відповідають загальним вимогам щодо пожежної та електробезпеки.

Екран монітора повинен бути розташованим перпендикулярно до напрямку погляду. Якщо він нахилений, це може спричинити погіршення постави. Відстань між очима та екраном повинна трохи перевищувати звичайну відстань між очима та книгою. Якщо на моніторі, особливо на старих моделях, відсутній захисний екран, необхідно сидіти на відстані витягнутої руки від нього. Ще одним аспектом, що стосується зору, є створення неоднорідного поля зору. Це можна досягти, розмістивши на стінах плакати або картини з спокійними кольорами, наприклад, пейзажі або натюрморти.

Форма спинки крісла має відповідати формі спини. Висота крісла має бути така, щоб користувач не відчував тиску на стегна або куприк. Крісло бажано обладнати підлокітниками, а його розташування має бути зручним, щоб не доводилося напружуватись, щоб дістатися до клавіатури. Дуже важливими є регулярні переміщення та зміна положення тіла, а також перерви у роботі.

Під час напруженої роботи за комп'ютером рекомендується робити перерви тривалістю 15 хвилин кожну годину і займатися іншими справами. Кілька разів на годину корисно виконати серію легких вправ для розслаблення.

Якщо не дотримуватись заходів безпеки під час роботи за комп'ютером, можуть виникнути такі наслідки, як:

- проблеми з органами зору (спостерігаються у 60 % користувачів);

- захворювання серцево-судинної системи (20 % користувачів);
- захворювання шлунково-кишкового тракту (10 % користувачів);
- шкіряні проблеми (5 % користувачів);
- ризик виникнення різноманітних пухлин.

Якщо у приміщенні використовується більше одного комп'ютера, важливо враховувати, що на користувача може впливати випромінювання від інших комп'ютерів, зокрема з бокових, та задньої стінки сусідніх дисплеїв. Тому необхідно встановити спеціальні фільтри та забезпечити, щоб користувач розміщувався на відстані не менше одного метра від бічних і задніх стінок інших дисплеїв.

Отже, для запобігання негативним впливам важливо бути ознайомленим з небезпечними аспектами самого комп'ютера, правилами безпечної роботи з ним, засобами запобігання ризикам, особливо пов'язаним з відомими загрозами, такими як електричні ураження та пожежна небезпека.

5.3 Правила безпеки при роботі з комп'ютером

Перед початком роботи на комп'ютері користувач повинен переконатися у цілісності корпусу та компонентів комп'ютера, а також перевірити наявність заземлення, стан та цілісність живильних кабелів та їх правильне підключення. У разі виявлення несправностей вмикати комп'ютер та розпочинати роботу заборонено.

Під час роботи, після переконання у справності обладнання, можна увімкнути живлення комп'ютера і почати роботу, дотримуючись умов, зазначених у відповідному посібнику з експлуатації.

Заборонено:

- замінювати різні деталі або компоненти під час роботи комп'ютера;
- з'єднувати або від'єднувати вилки та розетки живильної мережі, які знаходяться під напругою;

- відкривати кришки, що закривають доступ до струмопровідних частин живильної мережі під час роботи обладнання;
- використовувати паяльник з не заземленим корпусом;
- замінювати запобіжники під напругою;
- залишати комп'ютер увімкненим без нагляду.

Після закінчення робочого дня необхідно:

- вимкнути живлення комп'ютера, натиснувши відповідну кнопку та вийнявши вилку живильного кабелю з розетки, дотримуючись інструкцій з експлуатації;
- прибрати робоче місце користувача комп'ютера, відклавши використане обладнання та матеріали на призначені для них місця;
- у разі виявлення дефектів під час роботи комп'ютера – повідомити відповідним посадовим особам та спеціалістам.

5.4 Пожежна профілактика

Для забезпечення пожежної безпеки потрібно, в першу чергу, визначити типи та кількість первинних засобів пожежогасіння. При цьому необхідно враховувати фізико-хімічні та пожежонебезпечні властивості горючих речовин, їх взаємодію з вогнегасниками, а також площу виробничих приміщень, установок та відкритих майданчиків.

Для загасання невеликих вогнищ пожеж, де горіння не може відбуватися без доступу повітря призначені азбестові полотна, грубошерстяні тканини та повсті розміром не менше 1 кв. м.

У пожежному стенді мають бути ємності для піску об'ємом не менше 0,1 куб. м. Конструкція ящика з піском має забезпечувати зручний доступ до нього і запобігати потраплянню опадів або проникненню вологи іншими маршрутами.

Комплектація технологічного обладнання вогнегасниками повинна відповідати вимогам технічних умов (паспортів) на це обладнання або відповідним правилам пожежної безпеки.

Необхідно вибирати тип вогнегасників і розраховувати їх необхідну кількість залежно від їх вогнегасної здатності, максимальної площі, класу пожежі горючих речовин і матеріалів у закритих приміщеннях або на об'єктах згідно з ISO N 3941-77.

У замкнутих приміщеннях об'ємом до 50 куб. м для загасання пожеж можна використовувати портативні вогнегасники або додатково використовувати порошкові вогнегасники.

При виборі вогнегасника з врахуванням відповідної температурної межі використання необхідно враховувати кліматичні умови, в яких будуть експлуатуватися будівлі та споруди.

Вогнегасники, які були відправлені на перезарядку з підприємства, повинні бути замінені на відповідну кількість заряджених вогнегасників.

При захисті приміщень з ПК слід враховувати особливості взаємодії вогнегасних речовин з обладнанням, виробами, матеріалами та іншими елементами. Для цих приміщень рекомендується встановлювати вуглекислотні вогнегасники, враховуючи максимально допустиму концентрацію вогнегасної речовини.

Приміщення, які обладнані автоматичними стаціонарними установками пожежогасіння, повинні мати вогнегасники на 50 % від їх розрахункової кількості. Відстань від можливого вогнища пожежі до місця розташування вогнегасника не повинна перевищувати: 20 м для громадських будівель і споруд, 30 м для приміщень категорій А, Б і В, 40 м для приміщень категорії Г та 70 м для приміщень категорії Д.

Використання первинних засобів пожежогасіння для господарських та інших потреб, які не пов'язані з гасінням пожежі, заборонено.

З урахуванням типу будівлі (громадське приміщення) та можливого класу пожежі (Е), оскільки у приміщенні є багато комп'ютерів, визначаємо необхідну кількість вогнегасників – один порошковий вогнегасник об'ємом 5 літрів.

Висновки до п’ятого розділу

У даній частині дипломного проекту були розглянуті питання щодо гігієнічних норм організації і обладнання робочих місць користувачів персональних комп’ютерів, питання пожежної профілактики приміщень, виконані необхідні розрахунки об’єму недоторканого запасу води для зовнішнього пожежогасіння. Отримані результати дозволяють обладнати приміщення для роботи користувачів обчислювальної техніки з дотриманням вимог безпеки їх праці.

ЗАГАЛЬНІ ВИСНОВКИ

Розробка комп'ютерних ігор сьогодні є важливим і перспективним напрямком, який сприяє економічному розвитку, технологічному прогресу та культурному збагаченню нашої країни. Індустрія комп'ютерних ігор залучає висококваліфікованих фахівців, створює нові робочі місця та сприяє зростанню експорту українського програмного забезпечення на світовий ринок. Крім того, розвиток цієї галузі стимулює інновації та дослідження в сфері інформаційних технологій, що підвищує конкурентоспроможність України в глобальному контексті.

Підсумовуючи виконану роботу, можна зробити такі висновки:

1. Виконаний передпроектний аналіз показав, що такий різновид ігор-головоломок, як лабіринти, має дуже давню історію, але продовжує зацікавлювати гравців майже незалежно від їх віку, що робить цю розробку стабільно актуальною.

2. Проект гри, розроблений в даній роботі, передбачає процедурне генерування рівнів, що є однією з головних особливостей гри. Завдяки цьому можна очікувати на інтерес аудиторії і, відповідно, подальше зростання популярності гри.

3. Обгрунтований вибір ігрового рушія дозволив перейти до практичної реалізації проекту в середовищі Unity.

4. Реалізований тестовий проект Unity дозволив відпрацювати на практиці рішення, знайти помилки і добитися без збійного функціонування рівня. Звісно, що дана реалізація носить лише тестовий характер. Але важливо при цьому, що цього етапу не можна позбутися при створенні більш повноцінної гри.

5. Виконано техніко-економічний аналіз розробки, а також дослідження питань охорони праці користувача.

Таким чином, результати даної роботи мають значну потенційну затребуваність і хороші перспективи подальшого розвитку.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

ЕМВ – електромагнітне випромінювання.

НДДКР – науково-дослідні та дослідно-конструкторські роботи.

НДР – науково-дослідна робота.

НТЕ – науково-технічна ефективність.

ПК – персональний комп’ютер.

ПП – програмний продукт.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Басюркіна Н. Й., Свистун Т. В. Методичні вказівки до оцінки науково-технічної ефективності розробки нової технології, нового обладнання та інших інновацій. Для студентів всіх спеціальностей СВО «бакалавр» і «магістр» денної і заочної форм навчання. – Одеса : ОНТУ, 2022. – 18 с.
2. Вігуржинська С. Ю, Колесник В. І. Дипломне проектування економічної частини проекту: Методичні вказівки для студентів, що навчаються за комп’ютерними спеціальностями "Інформаційні управляючі системи та технології", "Інформаційні технології проектування", "Комп’ютерні системи та мережі" та "Спеціалізовані комп’ютерні системи". – Одеса: ОНАХТ, 2016. – 22 с.
3. Вісловух А. М. Охорона праці користувачів персональних комп’ютерів (ПК): Навчальний посібник. – К.: ІПК ДСЗУ, 2007. – 55с.
4. Головоломка (жанр комп`ютерних ігор) // Википедия [Електронний ресурс]. – Режим доступу: *https://ru.wikipedia.org/wiki/Головоломка_(жанр_комп'ютерних_ігор)/*.
5. Катренко Л. А., Катренко А. В. Охорона праці в галузі комп’ютинґу. — Львів : Магнолія – 2006, 2012. – 544 с.
6. Коллінз М. FORTNITE Official. Як малювати / Майк Коллінз, Кірстен Мюрей. – Артбукс, 2020. – 112 с.
7. Князєва Н. О. Дипломне проектування: Методичні вказівки до дипломної роботи спеціаліста / Н. О. Князєва, С. В. Шестопалов, С. Л. Жуковецька. — Одеса : ОНАХТ, 2016. — 46 с.
8. Стіл К. Геймер на 100%. Переходь у режим профі / Крейґ Стіл, Берат Пекмезчі. – Ранок, 2021. – 96 с.
9. Шраєр, Дж. Кров, піт і пікселі. Тріумфальні та бурхливі історії по той бік створення відеоігор. – К. : Book Chef, 2020. – 336 с.

10. Halpern J. Developing 2D Games with Unity. – Apress, 2019. – 380 p.
11. Hocking Joe. Unity in Action. – Packt Publishing, 2018. – 400 p.
12. Heussner T. The Advanced Game Narrative Toolbox. – CRC Press, 2019. – 232 p.
13. Hussain F. Advanced SFML Techniques: Pushing the Boundaries of Multimedia (SFML Fundamentals Book 3) / Frahaan Hussain, Kameron Hussain. – Independently published, 2023-10-23. – 344 p.
14. Irish D. The Game Producer's Handbook. – Cengage Learning PTR, 2005. – 352 p.
15. Keith Clinton. Agile Game Development: Build, Play, Repeat. – 2nd Edition. – Pearson Eduction, Inc., 2021. – 572 p.
16. Pullen, W. Labyrinth Cards. – Ozark Mountain Publishing, Inc, 2015. – 250 p.
17. Lavieri E. Getting Started with Unity 2018. – Packt Publishing, 2018. – 336 p.
18. Murray J. W. C# Game Programming Cookbook for Unity 3D. – 2nd ed. – CRC Press, 2021. – 316 p.
19. Okita A. Learning C# Programming with Unity 3D. – 2nd Edition. – CRC Press; Taylor & Francis Group, LLC, 2020. – 689 p.
20. Smith G. Basic Math for Game Development with Unity 3D: A Beginner's Guide to Mathematical Foundations / Gregory Smith, Kelvin Sung. – Apress, 2019. – 403 p.
21. Smith M. Unity 2018 Cookbook. – 3rd edition. – Packt Publishing, 2018. – 794 p.
22. Wells R. Unity 2020 By Example. – Packt Publishing, 2020. – 677 p.