

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)
на тему:

«Розробка системи головоломок для логічної гри
«What are the ghosts talking about?»»

Виконала студентка групи КН-19-1

ВЕНГЛОВСЬКА Юлія Володимирівна

Керівник роботи:

МАРЧУК Галина Вікторівна

Рецензент:

ЛЕВКІВСЬКИЙ Віталій Леонідович

Житомир – 2023

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СПЕЦІАЛЬНІСТЬ 122 «Комп'ютерні науки»**

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

Марина ГРАФ

«05» січня 2023 р.

ЗАВДАННЯ

на кваліфікаційну роботу освітнього ступеня «бакалавр»
за спеціальністю 122 «Комп'ютерні науки»
(освітня програма «Комп'ютерна графіка та розробка ігор»)

Здобувач вищої освіти: ВЕНГЛОВСЬКА Юлія Володимирівна

Керівник роботи: МАРЧУК Галина Вікторівна

Тема роботи: «Розробка системи головоломок для логічної гри «What are the ghosts talking about?»»,

затверджена наказом закладу вищої освіти від «05» січня 2023 р., №10/с

Термін здачі закінченої роботи: «10» червня 2023 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження): використання рушія Unity для розробки системи головоломок для 2D логічної гри

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Дата	
		Завдання видав	Завдання прийняв
1	Галина МАРЧУК	01.09.2022	01.11.2022
2	Галина МАРЧУК	01.11.2022	01.04.2023
3	Галина МАРЧУК	01.04.2023	01.05.2023

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі. Пошук, огляд та аналіз аналогічних розробок. Формулювання завдання. Опрацювання літературних джерел	25.09.22 – 30.09.22 01.10.22 – 05.10.22 06.10.22 – 08.10.22 08.10.22 – 10.11.22	Виконав
2	Проектування структури	11.11.22 – 20.11.22	Виконав
3	Написання програмного коду	21.11.22 – 20.03.23	Виконав
4	Тестування	21.03.23 – 25.04.23	Виконав
5	Попередній захист	10.06.2022	Виконав

Здобувач вищої освіти Юлія ВЕНГЛОВСЬКА
Керівник Галина МАРЧУК

РЕФЕРАТ

Випускна кваліфікаційна робота складається з програмного додатку гри та пояснювальної записки.

Пояснювальна записка до випускної кваліфікаційної роботи складається з вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 60 сторінках друкованого тексту.

Пояснювальна записка має 29 сторінок додатків. Список використаних джерел містить 20 найменувань і займає 2 сторінки. В роботі наведено 61 рисунок та 1 таблиця. Загальний обсяг роботи – 89 сторінок.

Ключові слова: UNITY, ДОДАТОК, ГРА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ГОЛОВОЛОМКА.

КН.КР.Б – 122 – 23 – ПЗ

					КН.КР.Б – 122 – 23 – ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка системи головоломок для логічної гри «What are the ghosts talking about?» Пояснювальна записка	Літ.	Арк.	Аркуші
Розроб.		Ю.В. ВЕНГЛОВСЬКА					3	89
Керівник		Г.В. МАРЧУК				Житомирська політехніка, група КН-19-1		
Рецензент		В.Л. ЛЕВКІВСЬКИЙ						
Н. Контр.								
Зав. каф.		М.С. ГРАФ						

ABSTRACT

The graduation thesis of the bachelor consists of a software application of the game and an explanatory note.

An explanatory note to the final thesis consists of an introduction, three chapters, conclusions, a list of used literature and an appendix.

The text part is laid out on 60 pages of printed text.

The explanatory note has 29 pages of annexes. The list of used sources contains 20 names and occupies 2 pages. The work contains 61 figures and 1 table. The total volume of work is 89 pages.

Keywords: UNITY, APP, GAME, SOFTWARE, PUZZLE.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ .9	
1.1 Постановка задачі	9
1.2 Аналіз існуючого програмного забезпечення за тематикою випускної кваліфікаційної роботи.....	10
1.3 Концепт-документ (Concept Document).....	13
1.4 Обґрунтування вибору інструментальних засобів	14
Висновки до першого розділу	15
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	16
2.1 Спроектовані головоломки	16
2.2 Розробка архітектури головоломок в ігровому додатку	27
2.3 Основні етапи реалізації гри.....	33
Висновки до другого розділу.....	39
РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ ..41	
3.1 Порядок встановлення та налаштування системи	41
3.2 Інтерфейс та порядок роботи з програмним додатком	41
3.3 Тестування роботи програмного забезпечення.....	53
Висновки до третього розділу:	57
ВИСНОВКИ.....	58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	59
ДОДАТКИ.....	61
Додаток А	62

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПЗ – Програмне забезпечення

WIP - Work In Progress – малюнок або рендер незавершеної роботи

UI – User Interface – інтерфейс, елементи, з якими взаємодіє користувач

NPC - Non-Player Character – персонажі, якими гравець не управляє, та з якими зазвичай можна взаємодіяти.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. Деякі головоломки відомі з глибокої давнини. Оригінальні логічні завдання знаходять на стінах єгипетських пірамід, у давньогрецьких манускриптах та інших історичних пам'ятниках.

Головоломки зберігають актуальність і важливість сьогодні. Вони важливі для розвитку різних аспектів людського життя, включаючи когнітивні, соціальні та психологічні аспекти.

Головоломки сприяють розвитку когнітивних навичок, таких як мислення, логіка, увага, пам'ять та концентрація. Вони можуть також допомогти підвищити творчість і просторову уяву.

Безперечно, стимулюється мозок і підтримується в хорошій формі. Вони можуть допомогти зберегти гнучкість мозку і зменшити ризик розвитку деменції та інших захворювань.

Головоломки можуть бути чудовим засобом розваги і відпочинку. Вони можуть допомогти знизити рівень стресу і допомогти зосередитися на чомусь позитивному.

Оскільки головоломки сприяють розвитку різних когнітивних навичок, вони можуть бути використані як навчальний інструмент в школах та університетах. Багато навчальних програм вже використовують логічні ігри та квести як засіб для покращення мислення, логіки та інших когнітивних навичок учнів.

Головоломки є не тільки важливим засобом для розвитку і підтримки розумової активності, але й можуть мати великий комерційний потенціал у світі комп'ютерних ігор. Ігри даного жанру займають значну частку на ринку і продовжують зростати в популярності серед гравців усіх вікових груп.

Статистика досить втішна. За 2022 рік ігри в жанрі головоломки були на третьому місці в рейтингу за виручкою (\$9,1 млрд) та на четвертому місці за кількістю завантажень (4,5 млрд).

Враховуючи зростаючу популярність головоломок в світі комп'ютерних ігор, запланований проект має високий потенціал на ринку. З

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		7

урахуванням тенденцій розвитку галузі, можна очікувати, що цей проект буде успішним і матиме велику кількість прихильників серед гравців.

Метою випускної кваліфікаційної роботи є проектування різноманітних ігрових механік та втілення системи головоломок для логічної гри "What are the ghosts talking about?".

Об'єктом дослідження є використання технологій Unity, зокрема Unity 2D з використанням патернів проектування.

Предметом дослідження є розробка системи головоломок для логічної гри "What are the ghosts talking about?" в середовищі Unity та використання мови C# для роботи з нею.

В рамках теми кваліфікаційної роботи було опубліковано тези:

Венгловська Ю. В., Марчук Д.К., ВИКОРИСТАННЯ ПАТЕРНІВ У ІГРОВИХ ПРОЕКТАХ. XIII Міжнародна науково-технічна конференція. Інформаційні комп'ютерні технології - 2023. 30-31 березня 2023 р. м. Житомир.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1 Постановка задачі

Основною метою розробки системи головоломок є: ознайомлення з історією виникнення логічних ігор, їх вплив на людський розвиток та психіку, вивчення основних механік. Гра з головоломками допоможе розвивати когнітивні навички у дітей та дорослих, розважати та підвищувати уважність до деталей.

У випускній кваліфікаційній роботі необхідно розробити частину додатку, що полягає у створенні великої кількості різноманітних головоломок, що не будуть повторюватися. Для реалізації поставленої задачі потрібно виконати такі етапи:

1. Вибір теми та жанру логічної гри:
 - 1.1.Визначення теми та жанру гри, на основі яких будуть розроблятися головоломки
 - 1.2.Визначення цільової аудиторії гри
2. Розробка механіки головоломок:
 - 2.1.Розробка концепції головоломок
 - 2.2.Розробка логіки та правил головоломок
 - 2.3.Додавання різних складностей (низька, середня та висока)
3. Тестування головоломок:
 - 3.1.Перевірка правильності роботи
 - 3.2.Перевірка логіки та правил
 - 3.3.Визначення, чи є можливість щось покращити
4. Поліпшення головоломок:
 - 4.1.Виправлення помилок та усунення недоліків
 - 4.2.Додавання нових головоломок та рівнів складності

1.2 Аналіз існуючого програмного забезпечення за тематикою випускної кваліфікаційної роботи

Для аналізу було обрано декілька ігор, спираючись на жанр, графіку та деякі основні механіки.

«Lost in Play»(https://store.steampowered.com/app/1328840/Lost_in_Play/) - логічна гра з приємною графікою. Грати доведеться за брата і сестру, які подорожують світом дитячих фантазій. Тут є багато як класичних головоломок, так і унікальних, які розробники придумали самі. Іноді завдання доволі складні, але пояснення правил міні-ігор дуже наочні та корисні (рис.1.1).

Переваги:

- більше тридцяти логічних міні-ігор;
- діалоги відсутні взагалі, присутня вигадана мова, але все одно сюжет зрозумілий та цікавий.

Недоліки:

- є декілька відгуків про часті припинення роботи додатку.



Рисунок 1.1 - Lost in Play

«Machinarium» (<https://store.steampowered.com/app/40700/Machinarium/>) - point&click гра. У ній є головоломки різних типів, зазвичай вони пов'язані з розблокуванням дверей, запуском механізмів та розв'язанням загадок. Багато

з них потребують уваги до деталей, адже потрібно знайти застосування предметам, що знаходяться навколо (рис.1.2).

Переваги:

- різноманітність головоломок;
- гра стимулює гравця мислити креативно;
- інтерактивність: гравець може взаємодіяти з різними предметами та персонажами.

Недоліки:

- не дуже високий рівень складності, що може сподобатися не всім любителям логічних ігор;
- вирішення деяких головоломок займає багато часу;
- іноді дії, що має виконати гравець, здаються нелогічними.



Рисунок 1.2 - Machinarium

«Rusty Lake Paradise»

(https://store.steampowered.com/app/744190/Rusty_Lake_Paradise/?) - логічна гра з головоломками, виконана у дещо похмурому стилі з гарною графікою (рис.1.3).

Переваги:

- наявні різні типи головоломок: логічні міні-ігри, розгадування кодів, паролів тощо;

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		11

- гра має темну атмосферу, містить елементи містики та сюжет пересікається з різними легендами та фольклорними мотивами. Це робить гру унікальною та цікавою;
- є можливість скористатися підказками.

Недоліки:

- гру можна пройти досить швидко;
- іноді складно зрозуміти, що робити далі;
- є відгуки про незручність управління.



Рисунок 1.3 - Rusty Lake Paradise

Отже, було вирішено створити власну логічну гру. Важливим є те, щоб вона відповідала таким критеріям:

- простий та зручний інтерфейс;
- різні механіки (drag&drop, малювання ліній, ввід чисел, переливання води);
- наявність підказок;
- різноманіття головоломок, як класичних, так і незвичних для гравців;
- правильно скорегована складність;

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		12

- рівні мають бути збалансовані, щоб на кожному була приблизно однакова кількість загадок, які буде не занадто складно знайти та вирішити.

1.3 Концепт-документ (Concept Document)

Жанр і цільова аудиторія:

Жанр – головоломка/квест.

У грі буде три рівні складності, що зацікавить людей різного віку.

Легка складність розрахована на дітей 7-10 років.

Середня – на 11-18 років.

Складна буде цікава дорослим, основна аудиторія – 19-30.

Основні особливості гри:

Гра буде вирізнятися серед інших цікавим сюжетом, унікальними та досить складними головоломками, декількома рівнями складності та системою підказок.

Геймплей:

Головна героїня знаходиться в старому будинку. Вона ходить по кімнатах, обстежуючи кожен куток і відшукує відповіді на свої запитання.

Кожна кімната має свою власну головоломку, і героїня старанно розв'язує їх одну за одною.

Головоломки виглядають просто, але потребують глибокого аналізу, кмітливості і творчого мислення. Потрібно зосередитися на деталях і пройти кожне завдання, щоб отримати остаточну відповідь.

Сюжет:

Дівчина-художниця малює у себе вдома, коли їй приходить лист від незнайомця. Це запрошення на вечірку. Героїня дуже допитлива, тому вона вирішує піти у назначене місце. Там вона бачить гарний будинок, але, коли заходить всередину, двері зачиняються. Дівчина у паніці, коли біля неї з'являється привид, який видається доволі дружнім. Він розповідає свою історію.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

Колись чоловік із дружиною жили у цьому прекрасному будинку. Вони були щасливі, ніколи не сварилися, багато чого досягли за життя. Чоловік був талановитим художником та часто малював портрети дружини. Проте одного разу жінка захворіла та незадовго померла. Від розпачу та туги помер і її чоловік. Після смерті подружжя в їх домі оселилися різні потойбічні створіння, деякі з них були злі та розірвали найкращий портрет жінки.

Чоловік приречений скитатися у вигляді привида до тих пір, поки хтось не збере картину до купи. Героїню ця історія розчулила, тому вона зголошується допомогти. Все одно виходу немає: поки усі таємниці цього будинку не будуть розпутані, він не випустить нікого.

Платформа:

Наразі планується, що гра буде розроблятися тільки під PC-платформу.

1.4 Обґрунтування вибору інструментальних засобів

Популярні на даний час рушії для створення ігрових додатків: Unity, Unreal Engine, GameMaker, Godot та інші.

Було обрано Unity, бо він:

- легкий та зручний інтерфейс;
- багатофункціональність: рушій має багато готових компонентів та інструментів для розробки як 2D, так і 3D ігор;
- кроссплатформенність: підтримується розробка для платформ Windows, Mac, Android, iOS та інші;
- велика спільнота: по Unity є багато безкоштовних та платних уроків, курсів, майстер-класів, на яких можна отримати знання та навички;
- мова розробки – C#, високорівнева мова програмування, яку не дуже складно освоїти.

Для написання скриптів обрано Visual Studio, бо це дуже зручна програма, у якій дуже зручно працювати з Unity-проектами.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

Висновки до першого розділу

Визначено мету розробки та поставлено завдання. Переглянуто три популярні гри жанрів point&click, головоломки та квести, визначено їх переваги та недоліки. Проаналізовано потреби гравця та поставлено задачу проекту, а саме який в загальному має бути функціонал.

Створено концепт-документ, в якому визначено жанр та аудиторію, особливості, геймплей, сюжет та платформу.

Обґрунтовано вибір інструментарію (Unity та Visual Studio), розглянувши переваги, що будуть в нагоді для проекту.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						15
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Спроектовані головоломки

Даний підрозділ описує усі наявні у грі головоломки. Для наочного представлення було показано те, як головоломки виглядали у прототипі, майже усі без готової графіки.

У більшості головоломок є три рівня складності, але було показано лише один – той, що був спроектований як прототип.

- Рівень 1: ванна кімната

1. Головоломка з трубами (рис.2.1), які треба повернути під правильними кутами.
2. Головоломка з качками (рис. 2.2), на які потрібно натиснути в правильному порядку. Пояснення до загадки заховане на рівні у вигляді картин, в яких кольори та кількість квітів показують відповідний порядок та кількість натисків на качок.
3. Класична головоломка – п'ятнашки (рис. 2.3).

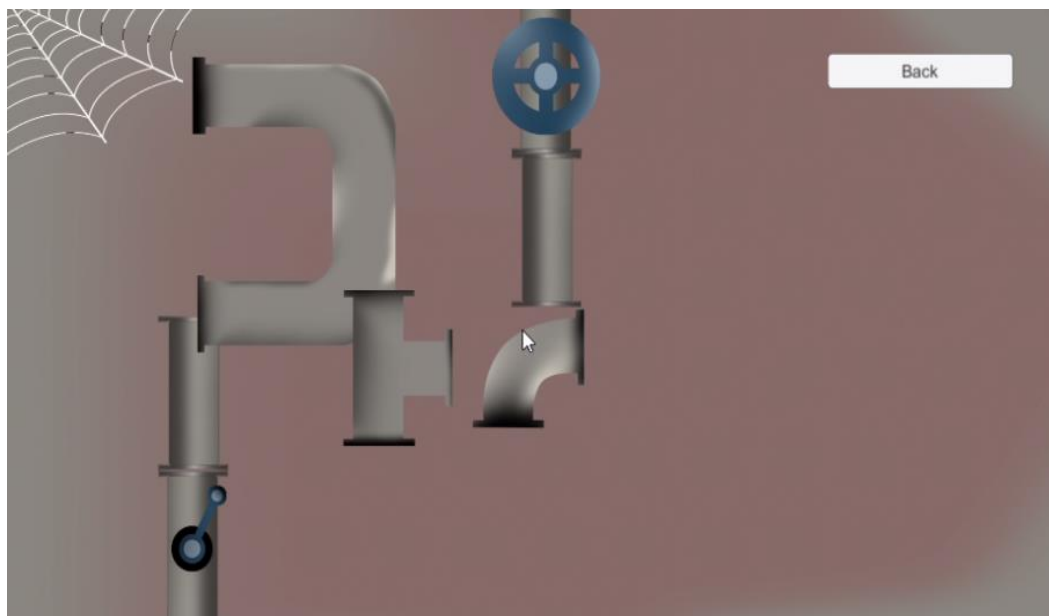


Рисунок 2.1 - Труби



Рисунок 2.2 - Качки

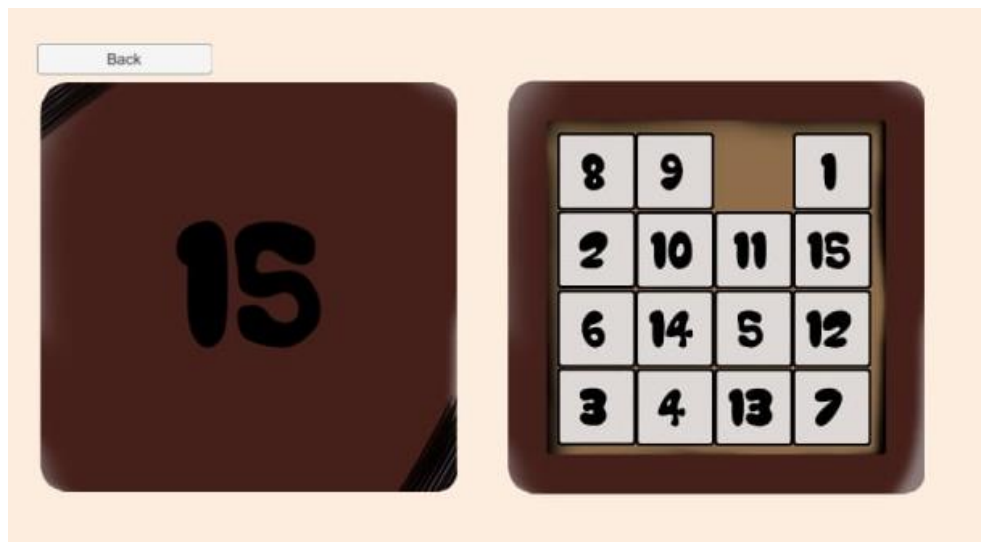


Рисунок 2.3 - П'ятнашки

- Рівень 2: зала
- 1. Ліхтарики на ялинці (рис.2.4). При натисканні на один, ті, що знаходяться поруч, загораються, якщо не були запалені, та навпаки.
- 2. Головоломка з подарунком – рис.2.5. Потрібно натиснути на бантики у потрібному порядку (по зростанню кількості петель).
- 3. Головоломка з метеликом – рис.2.6. Навколо силуету розташовані лампочки. Потрібно натискати на кнопку – ліву чи праву – в залежності від того, з якого боку розташована

наступний світильник. Якщо було здійснено неправильний вибір, усі ліхтарики гаснуть і потрібно почати спочатку.

4. Класична головоломка з предметом під склянкою. Спочатку гравцеві показується, куди покладено предмет, після чого склянки починають переставлятися місцями. Потрібно вгадати місцеположення предмета.

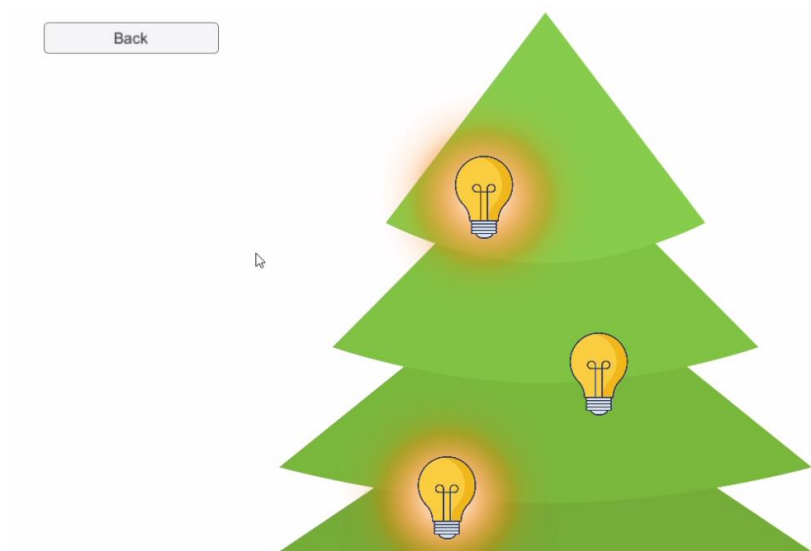


Рисунок 2.4 - Ліхтарі

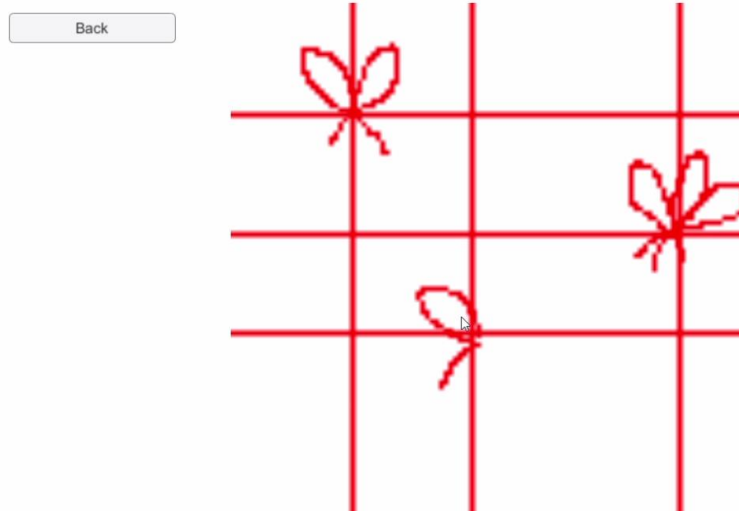


Рисунок 2.5 - Подарунок

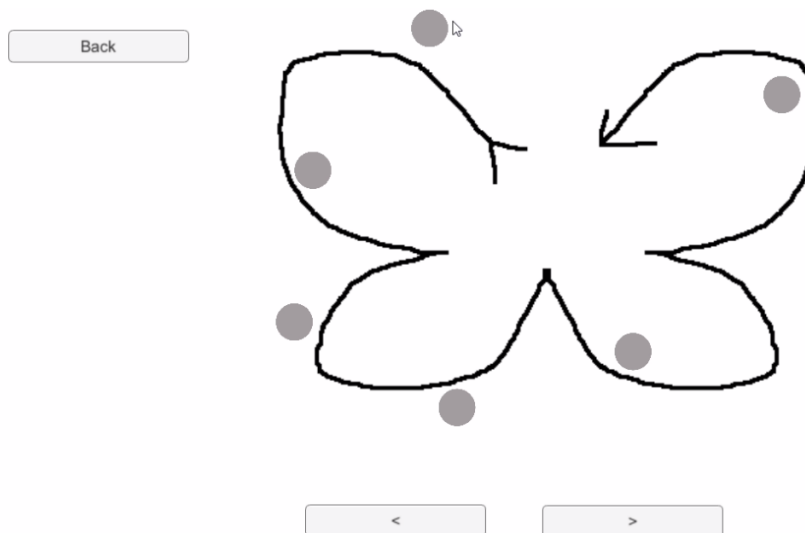


Рисунок 2.6 - Метелик



Рисунок 2.7 - Головоломка зі стаканами

- Рівень 3: кухня

1. Головоломка зі столовими приладами (рис.2.8). Зверху та зліва від полотна написані цифри, які показують, скільки точок цього ряду повинно бути закрито. Потрібно перемістити ложки, тарілки та інше так, щоб закрити усі потрібні точки.

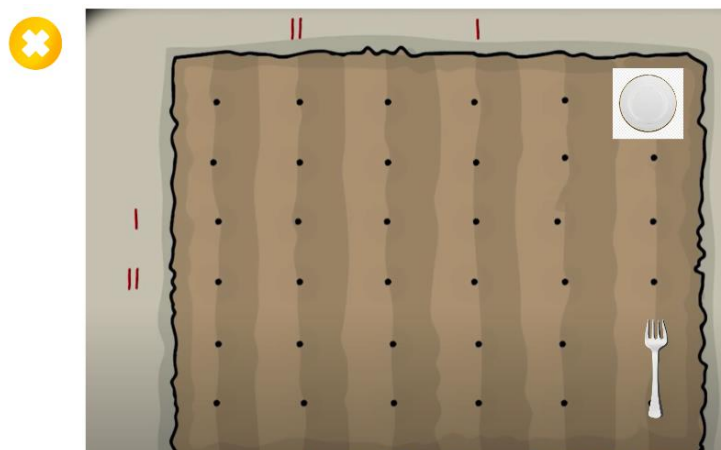


Рисунок 2.8 - Столові прилади

- Рівень 4: кімната без світла

1. Головоломка з розеткою (рис.2.9). Потрібно дотягнути дрід до розетки, після чого ввімкнеться лампа.
2. Різнобарвні ліхтарі (рис.2.10). Треба поєднати попарно лампочки однакового кольору.
3. Головоломка зі світляком (рис.2.11). На дошці є клітинки, по яким рухається світляк. При натисканні на одну з них, вона стає неактивною, на яку не може пересунутися жук. Він же в свою чергу робить хід на одну сусідню активну клітинку при кожному кліку гравця. Потрібно зробити ходи так, щоб оточити світляка і не дати йому можливості втекти.

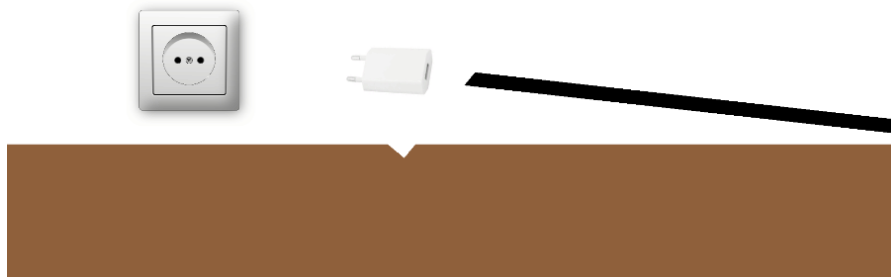


Рисунок - 2.9 Розетка

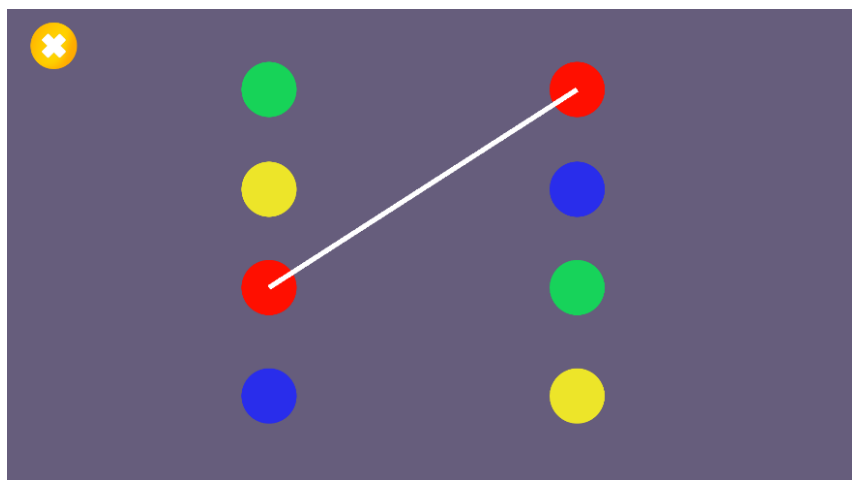


Рисунок 2.10 - Головоломка з різнокольоровими ліхтариками

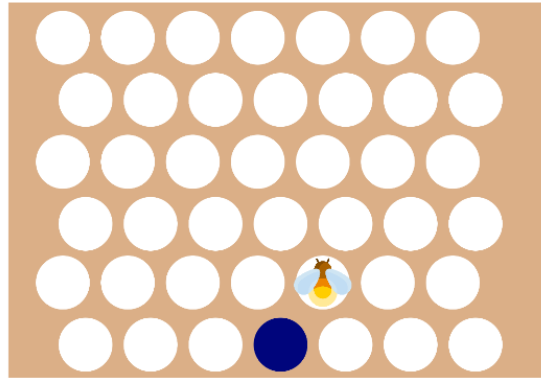


Рисунок 2.11 - Головоломка зі світляком

- Рівень 5: спальня.
- 1. Класичні пазли (рис.2.12).
- 2. Судоку (рис.2.13).
- 3. Головоломка із зірками (рис.2.14). Потрібно правильно поєднати точки. Над кожною надписано порядковий номер.
- 4. Головоломка з книжками (рис.2.15). Потрібно переставити книжки місцями так, щоб утворився візерунок.
- 5. Після виконання попередніх головоломок гравець дізнається цифри, які потрібно правильно скомбінувати у пароль (рис.2.16).

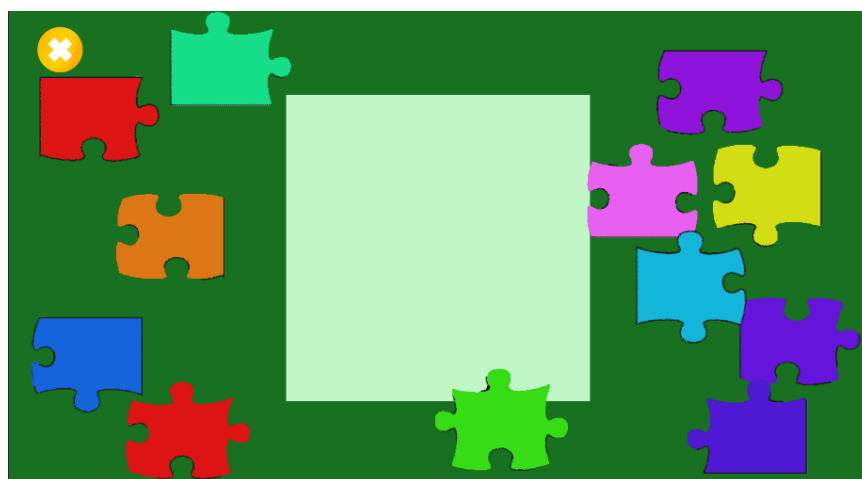


Рисунок 2.12 - Пазли

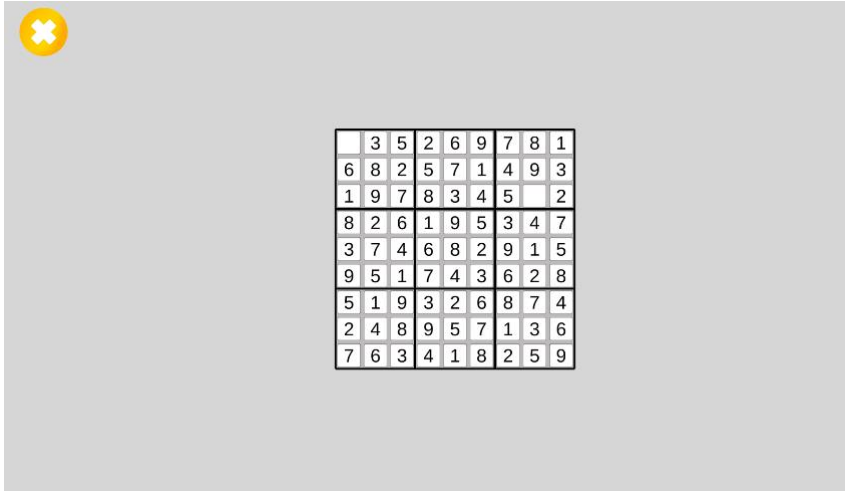


Рисунок 2.13 - Судоку

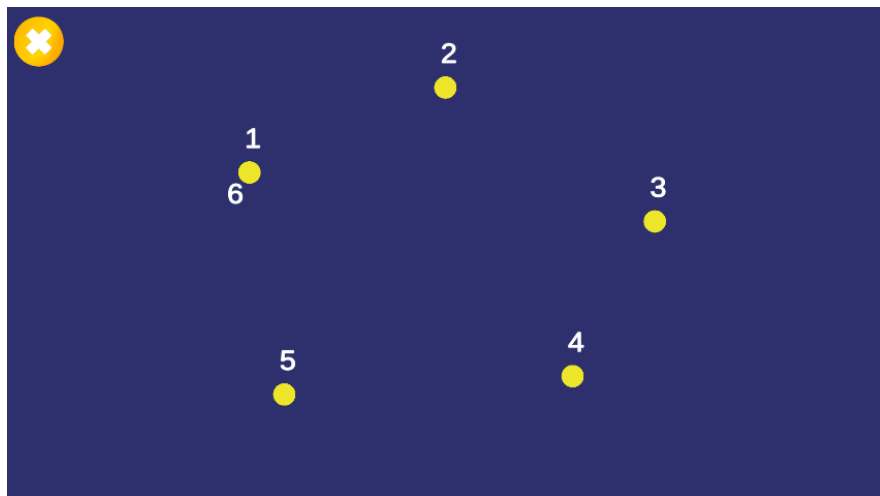


Рисунок 2.14 - Зірки

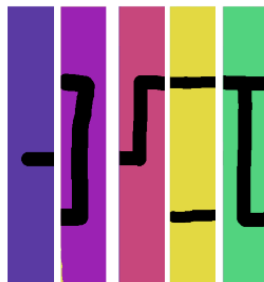


Рисунок 2.15 - Книжки з візерунками



0 0 0 0

Рисунок 2.16 - Пароль

- Рівень 6: комора
- 1. Ваги (рис.2.17). На них можна ставити пляшки з водою та дивитися, яка з двох легша.
- 2. Головоломка з відмірюванням води (рис.2.18). Потрібно перелити воду так, щоб в потрібній ємності була зазначена кількість води.
- 3. Сорткування води (рис.2.19). Класична головоломка, де потрібно переміщувати верхній шар між склянками так, щоб в результаті в кожній посудині був лише один колір.
- 4. Після усіх вище вказаних дій треба розставити склянки з водою по зростанню її ваги.

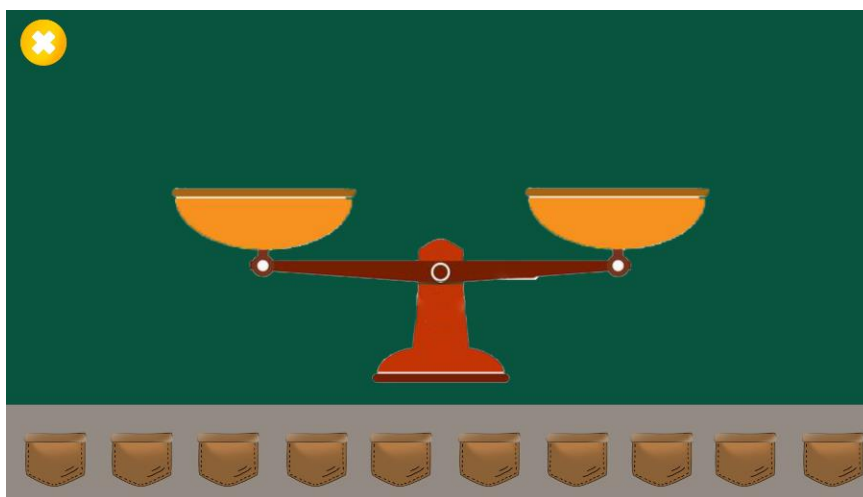


Рисунок 2.17 - Ваги

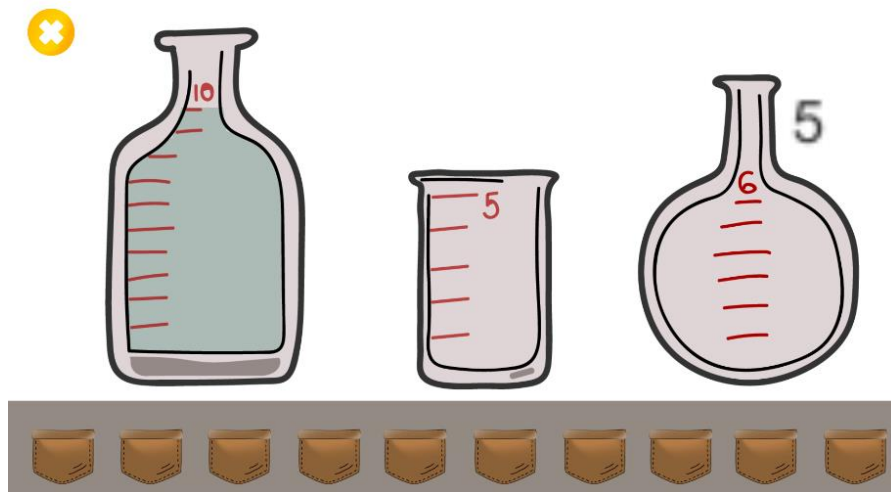


Рисунок 2.18 - Головоломка з відмірюванням води

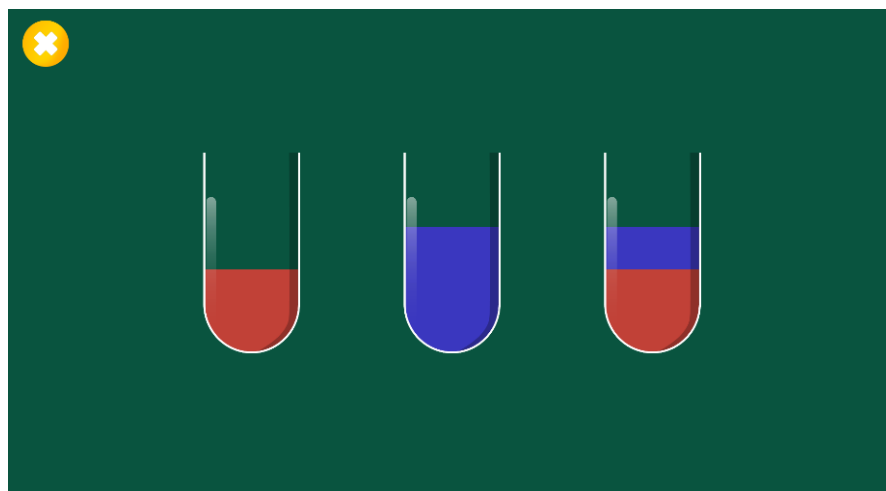


Рисунок 2.19 - Головоломка з сортуванням води

- Рівень 7: кімната з ігровими автоматами
 1. Класичний лабіринт (рис.2.20). Потрібно управляти кулькою та знайти вихід.
 2. Тетріс – усім знайома гра (рис.2.21).
 3. Головоломка з бобрами (рис.2.22). Спочатку підсвічуються комірки, після чого з'являються бобри. Гравець має натиснути на тих, що були виділені.
 4. Танцювальний апарат (рис.2.23). Стрілки підсвічуються по черзі, треба запам'ятати та відтворити правильний порядок.

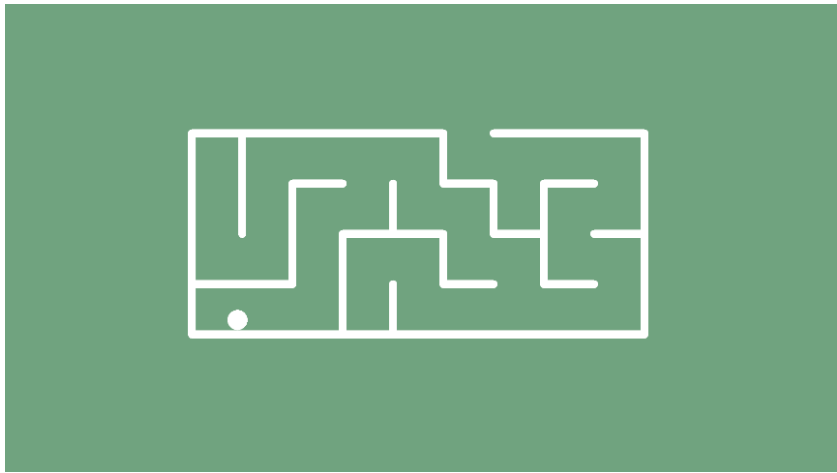


Рисунок 2.20 - Лабіринт

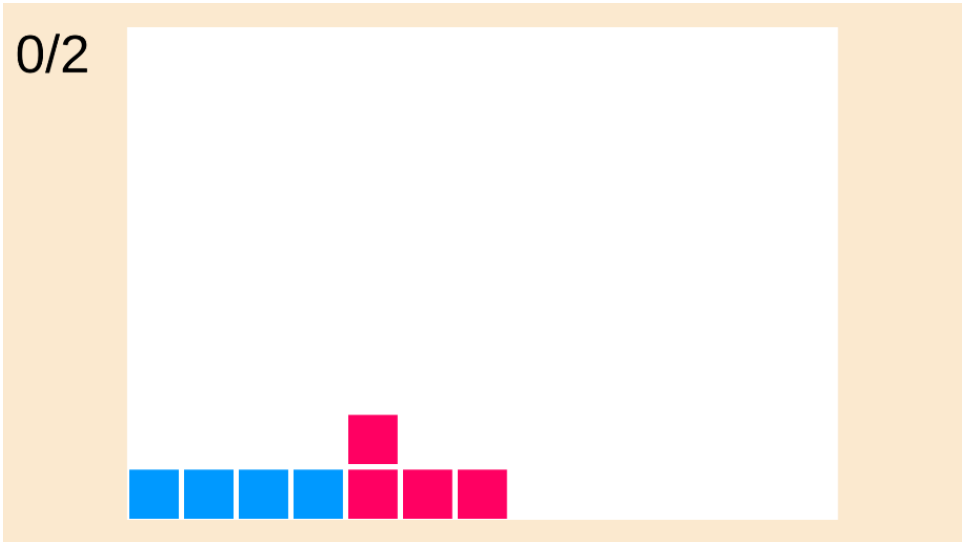


Рисунок 2.21 - Тетріс

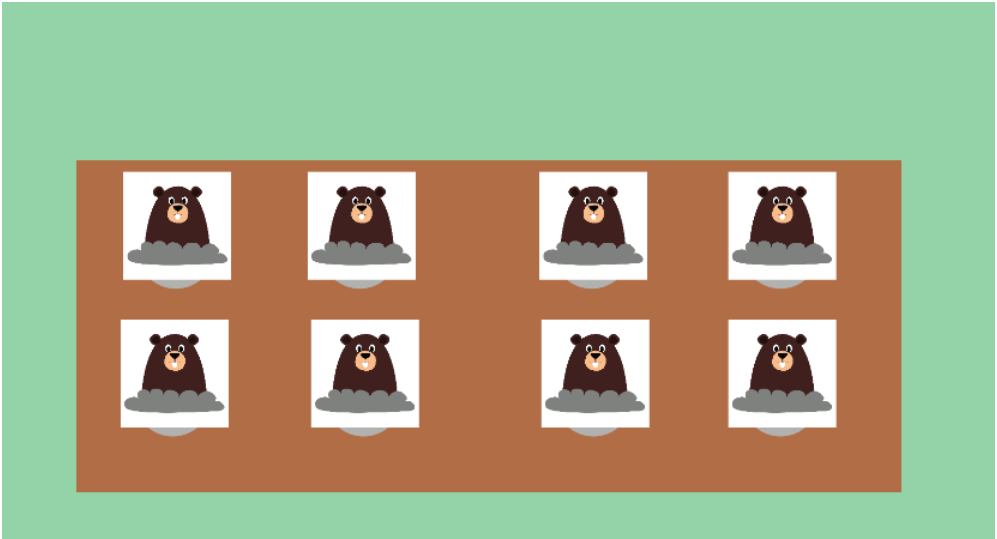


Рисунок 2.22 - Головоломка з бобрами

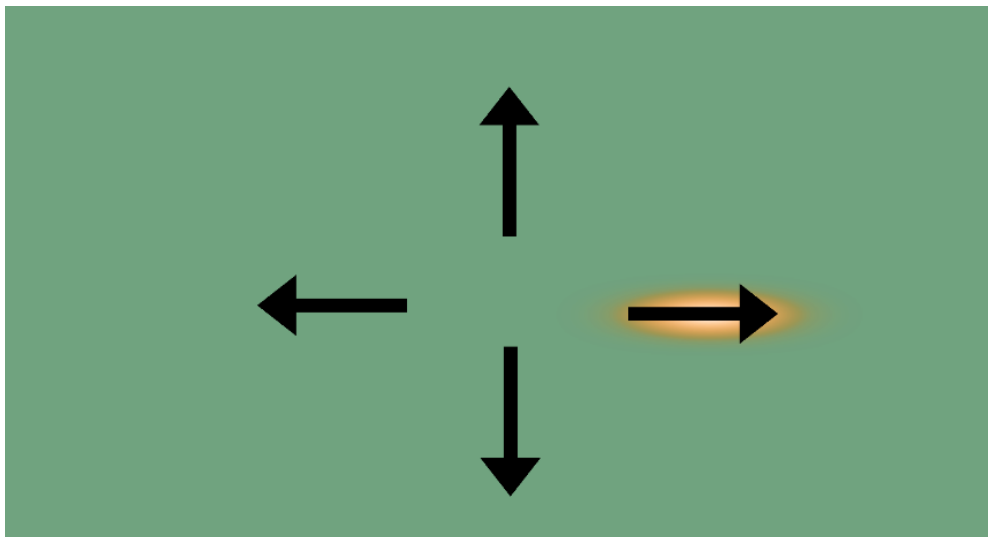


Рисунок 2.23 - Головоломка з танцювальним апаратом

- Рівень 8: кімната з музикою
- 1. Музикальна скринька (рис.2.24). На сцені потрібно знайти листок зі схемою, де показано, які комірки натиснути.
- 2. Головоломка з касетою (рис.2.25). Потрібно витягнути касету з шафи. Блоки рухаються або горизонтально, або вертикально, в залежності від їх форми.

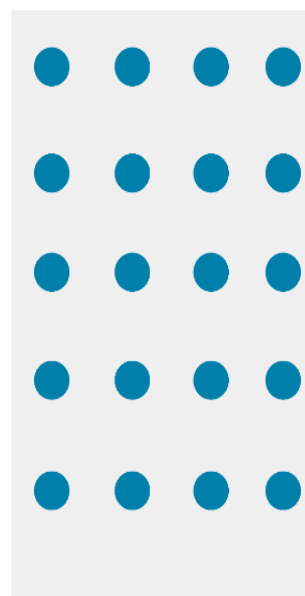


Рисунок 2.24 - Музикальна скринька

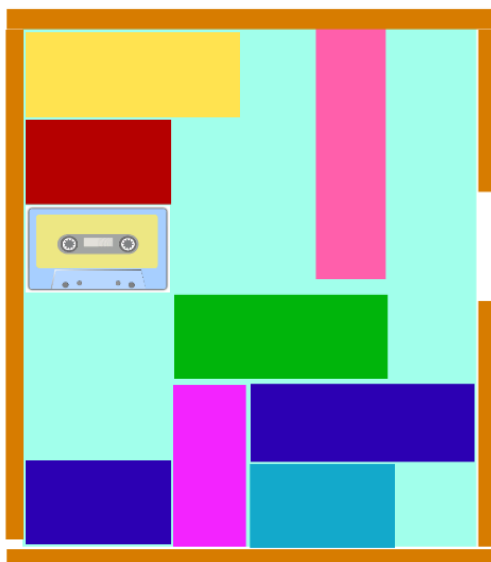


Рисунок 2.25 - Головоломка з касетою

Отже, продумано більше двадцяти головоломок, у кожній – унікальні механіка та завдання. В результаті гра вийде різноманітною, зацікавить багатьох гравців.

2.2 Розробка архітектури головоломок в ігровому додатку

Діаграма варіантів використання (Use Case Diagram) (рис.2.26).

На схемі можна побачити, що у кожній головоломці гравець може робити.

Звичайно, можна вийти на загальну сцену, тобто закрити інтерфейс головоломки. Основна механіка – це взаємодія з предметами. При цьому, можна підібрати предмет, перетягнути, використати з інвентаря або поєднати два в один.

Також, якщо у головоломці є окремий персонаж (наприклад, кулька, що рухається по лабіринту), ним можна управляти.



Рисунок 2.26 - Діаграма варіантів використання

Діаграми активності описують, як діяльність координується для виконання дії, яка може бути на різних рівнях абстракції.

Створені діаграми можна переглянути на рисунках 2.27, 2.28, 2.29 і 2.30. Так як описані дії лише окремо в головоломках, схеми вийшли роздібнені та поділені на маленькі.



Рисунок 2.27 - Діаграма активності для перетягування предмета

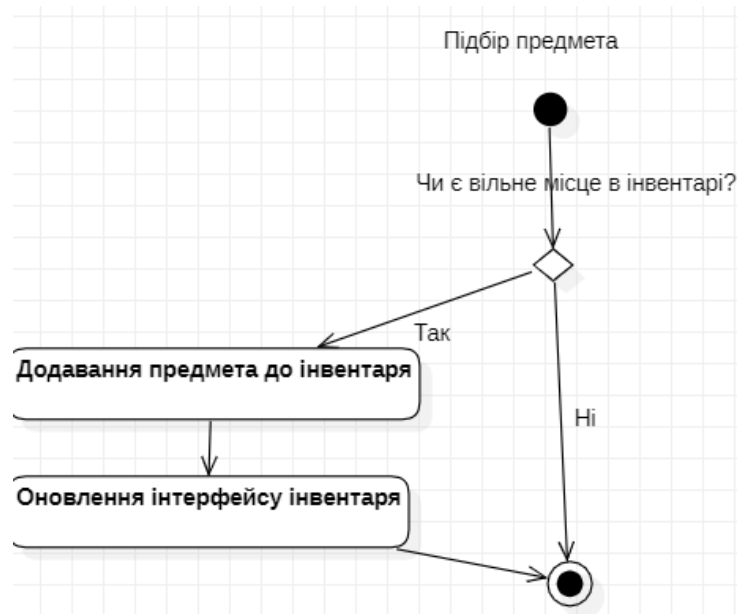


Рисунок 2.28 - Діаграма активності для підбору предмета

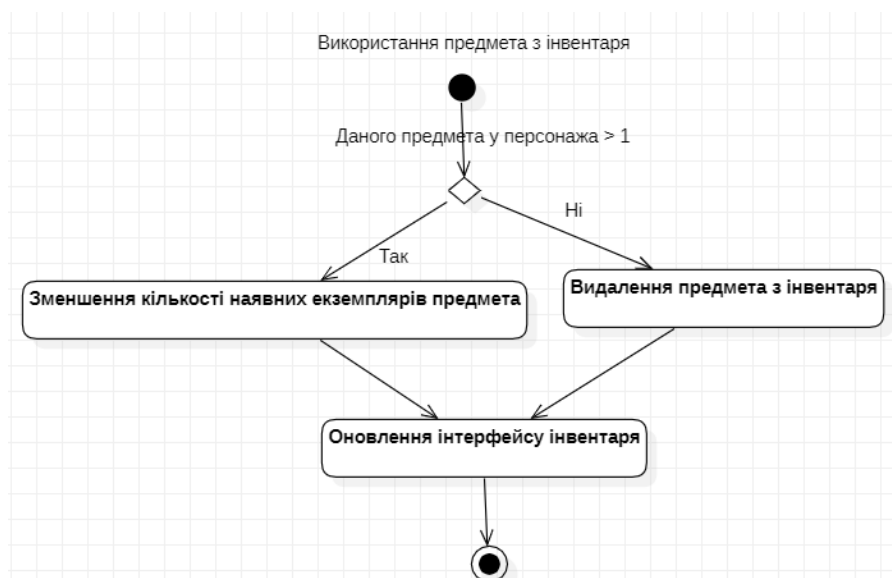


Рисунок 2.29 - Діаграма активності для використання предмета з інвентаря

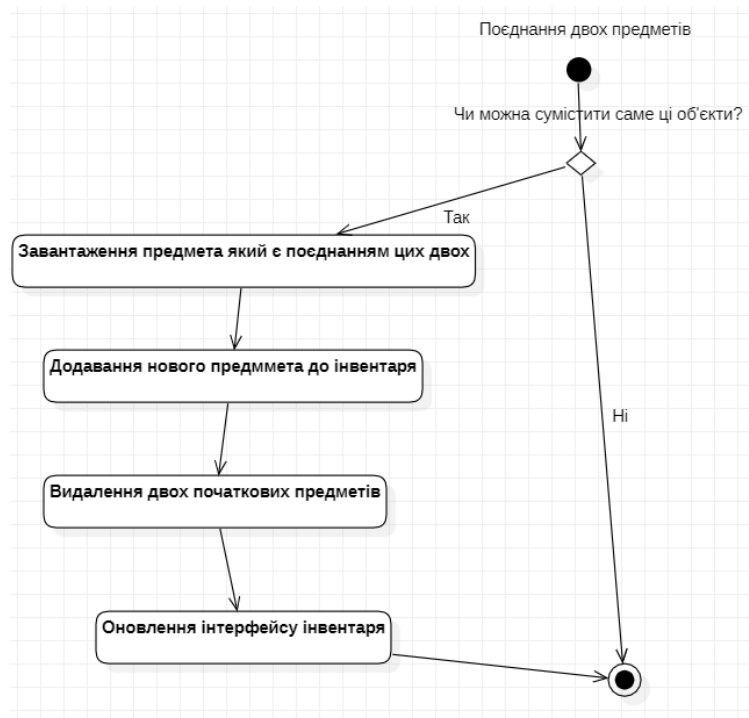


Рисунок 2.30 - Діаграма активності для поєднання двох предметів

Для перетягування предмета (див. рис.2.27), коли гравець перемістив предмет, проводиться перевірка, чи є задовільними нові координати. Якщо так, вони зберігаються, в іншому випадку зміни скасовуються і об'єкт залишається на попередній позиції.

Коли персонаж підбирає предмет (див. рис.2.28), спочатку йде перевірка на наявність вільного місця в інвентарі. Якщо ні, то далі нічого не відбувається, якщо так, то предмет додається до інвентаря, після чого інтерфейс оновлюється, щоб відображати новий предмет також.

Якщо ж предмет намагаються використати (див. рис.2.29), йде перевірка на його кількість у персонажа. Коли його є більше ніж один екземпляр, то зменшується кількість, інакше предмет повністю видаляється.

У грі є можливість поєднувати два предмета (див. рис.2.30), для цього проводиться перевірка, чи можна взагалі сумістити дані речі, якщо так, то з ресурсів завантажуються предмет, який має вийти в результаті, додається в інвентар, потім видаляються дві початкових деталі, і зрештою йде оновлення інтерфейсу.

Діаграму класів можна побачити на рисунку 2.31. Представлена частина класів відповідає за основні головоломки на перших двох рівнях.

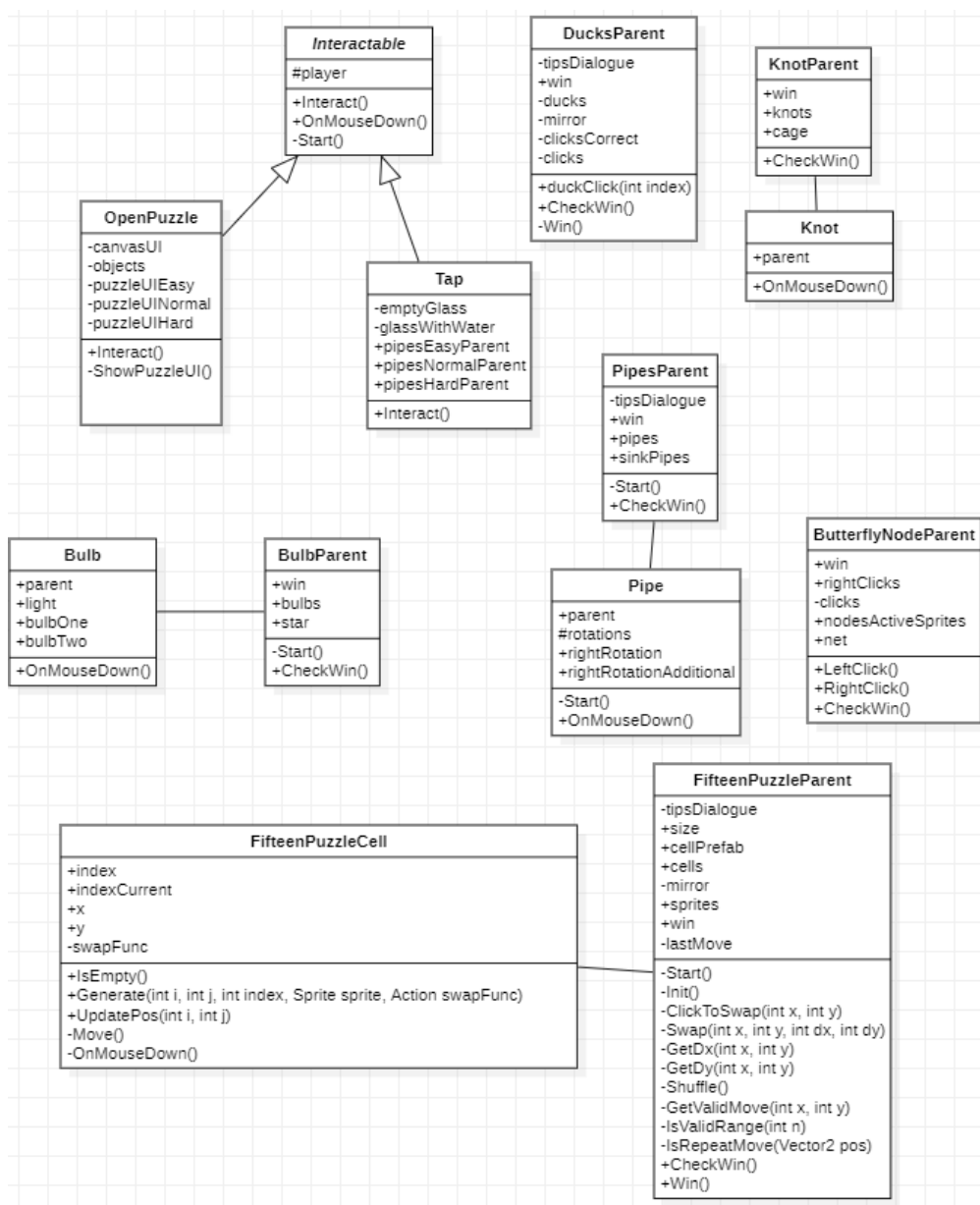


Рисунок 2.31 - Діаграма класів

Опис класів, а також їх полів та методів:

- клас **Interactable** – абстрактний клас, який має шаблон для виконання методу після натиску на об'єкт;
- клас **OpenPuzzle** – клас, який відповідає за відкриття головоломки. Він має поля з об'єктами інтерфейсу, які треба буда сховати або ж, навпаки, показати, та методи для показу головоломки;
- інші класи стосуються напряму головоломок (**DucksParent** – головоломки з качками, **KnotParent** та **Knot** – головоломки з

вузлами тощо) та переважно в параметрах мають поля, що відповідають за об'єкти, які треба сховати або показати в певний момент, посилання на інтерфейс головоломок, а в методах – методи генерування (наприклад, Generate в Fifteen PuzzleCell – метод генерування комірок для п'ятнашок), та методи перевірки на виграш.

Варто зазначити, що у проектуванні було використано патерн. Патерн проектування — це типовий спосіб вирішення певної проблеми, що часто зустрічається при проектуванні архітектури програм.

Шаблонний метод — це поведінковий патерн проектування, який визначає кістяк алгоритму, перекладаючи відповідальність за деякі його кроки на підкласи. Патерн дозволяє підкласам перевизначати кроки алгоритму, не змінюючи його загальної структури.

Для початку певний клас треба зробити абстрактними. З цієї причини усі підкласи повинні будуть реалізувати кожен з кроків по-своєму. Таким чином, є основний метод Interact(), який викликається по кліку мишею на об'єкт з цим класом, але сам клас Interactable не можна використовувати сам по собі, а лише класи, що його наслідують, і в них даний метод може бути перевизначений.

Діаграма компонентів (рис.3.32) описує організацію та підключення фізичних компонентів у системі.

Як видно з діаграми, гра складається з основних модулів, таких як: графічний, модулі налаштувань, підказок, керування персонажем та інші. Графічний модуль включає в себе графічні бібліотеки, що дозволяють взагалі відображати сцену та дають змогу додавати візуальні ефекти.

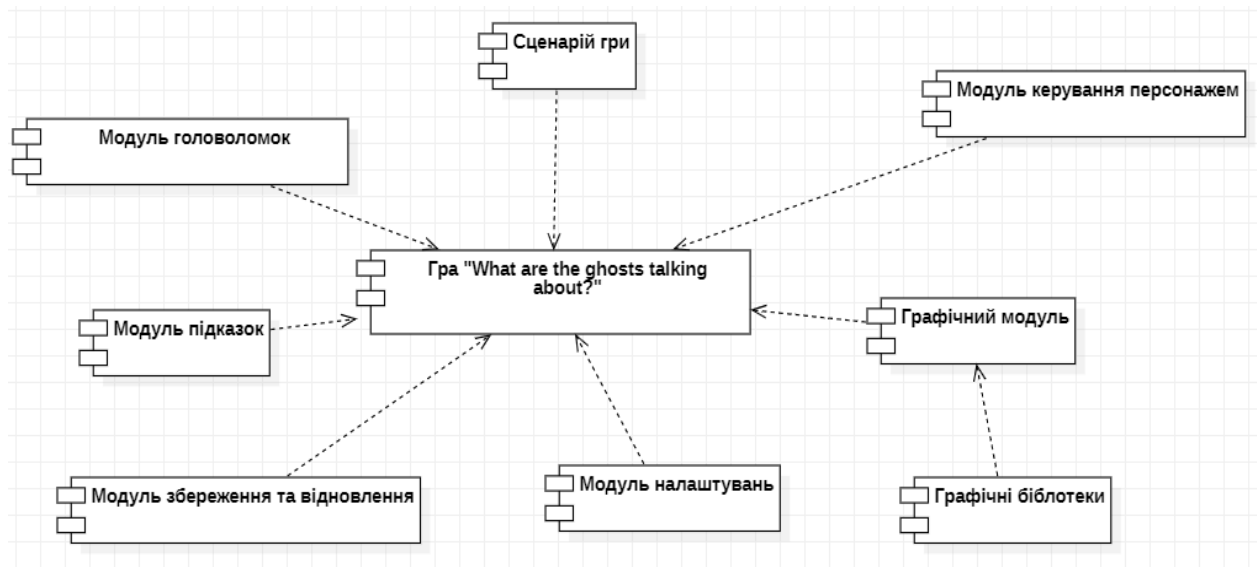


Рисунок 2.32 - Діаграма компонентів

Діаграми розгортання (рис.3.33) зазвичай використовуються для візуалізації фізичного обладнання та програмного забезпечення системи. Використовуючи його, ви можете зрозуміти, як система буде фізично розгорнута на апаратному забезпеченні.

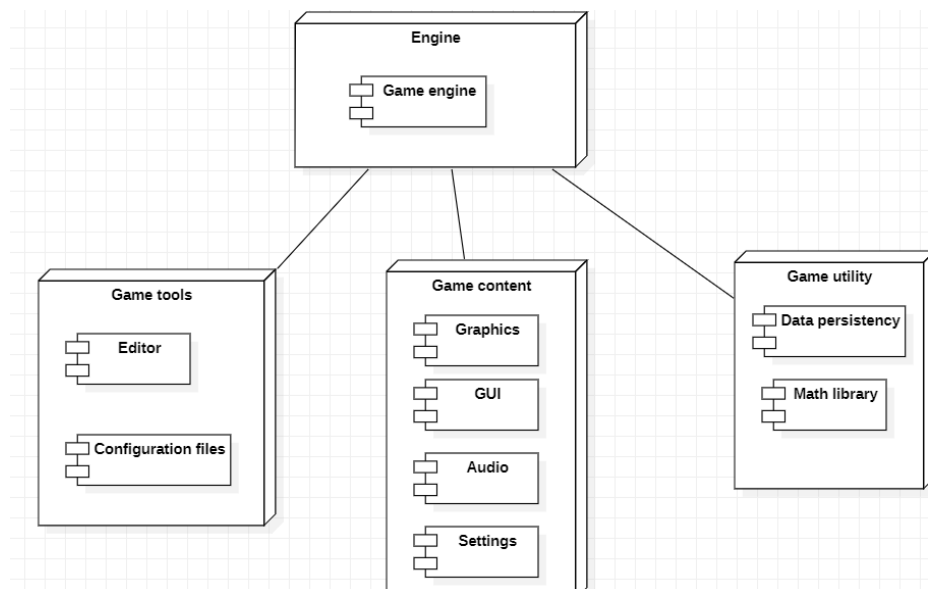


Рисунок 2.33 - Діаграма розгортання

2.3 Основні етапи реалізації гри

Спочатку було розроблено основний абстрактний клас `Interactable` для взаємодії з оточенням. У ньому є методи, що забезпечують зчитування натиску кнопкою миші по даному об'єкту:

```

public virtual void Interact()
{
    Debug.Log("Interact");
}

public void OnMouseDown()
{
    Interact();
}

```

Крім цього, однією з механік є перетягування предмета. За це відповідає DragAndDrop. При кліку зчитуються координати курсора, потім предмету постійно присвоюється позиція миші аж доти, доки гравець не відпускає кнопку:

```

protected Vector3 GetMouseWorldPosition()
{
    return Camera.main.ScreenToWorldPoint(Input.mousePosition);
}

protected virtual void OnMouseDown()
{
    mousePositionOffset = transform.position - GetMouseWorldPosition();
}

protected virtual void OnMouseDrag()
{
    Vector2 position = GetMouseWorldPosition() + mousePositionOffset;
    rb.MovePosition(position);
}

```

Звичайно, найбільше уваги було приділено саме логіці головоломок. Найскладніші механіки було спроектовано у таких головоломках, як п'ятнашки, sudoku та процедурна генерація лабіринту.

Для п'ятнашок створено два класи: FifteenPuzzleParent та FifteenPuzzleCell. Спочатку у циклі відбувається генерація клітинок: додається префаб, на нього накладається спрайт з відповідним числом від 1 до 15 при грі на простій та середній складності, або від 1 до 24 на високій. Оновлюється позиція.

Після того, як усі клітинки наявні на сцені у порядку зростання, вони перемішуються. Для цього 500 разів запускається функція перестановки місцями пустої клітинки з будь-якою сусідньою:

```

void Shuffle()
{
    for (int i = 0; i < size; i++)

```

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		34

```

{
    for (int j = 0; j < size; j++)
    {
        if (cells[i, j].IsEmpty())
        {
            Vector2 pos = GetValidMove(i, j);
            Swap(i, j, (int)pos.x, (int)pos.y);
        }
    }
}
}

```

Потім розпочинається гра: гравець може натискати на будь-яку клітинку. Якщо поряд є пуста комірка, вони міняються місцями. Інакше нічого не відбувається. Після кожного ходу гравця йде перевірка на перемогу.

Цікава механіка наявна у sudoku. Спочатку гри уже є задані числа у вигляді матриці – рис.2.34. При завантаженні головоломки кожна клітинка заповнюється відповідним числом. Якщо ж у матриці стоїть нуль, клітинка залишається порожньою.

```

private int[,] matrixEasy = new int[9, 9]
{
    { 0, 3, 5, 2, 6, 9, 7, 8, 1 },
    { 6, 8, 2, 5, 7, 1, 4, 9, 3 },
    { 1, 9, 7, 8, 3, 4, 5, 0, 2 },
    { 8, 2, 6, 1, 9, 5, 3, 4, 7 },
    { 3, 7, 4, 6, 8, 2, 9, 1, 5 },
    { 9, 5, 1, 7, 4, 3, 6, 2, 8 },
    { 5, 1, 9, 3, 2, 6, 8, 7, 4 },
    { 2, 4, 8, 9, 5, 7, 1, 3, 6 },
    { 7, 6, 3, 4, 1, 8, 2, 5, 9 }
};

```

Рисунок 2.34 - Приклад матриці для sudoku

Гравець може записувати пусті клітинки числа, мотім змінювати або видаляти їх. Після кожної зміни відбувається перевірка на правильність рішення – найскладніша частина коду. На повторення цифри спочатку перевіряється кожний горизонтальний ряд:

```

void CheckHorizontalLines()
{
    List<int> horizontalForCheck = new List<int>();

    for (int i = 0; i < height; i++)
    {
        horizontalForCheck.Clear();
        for (int j = 0; j < width; j++)
        {

```

```

        if (cells[i, j].value > 0 && cells[i, j].value < 10)
        {
            horizontalForCheck.Add(cells[i, j].value);
        }
        else
        {
            Debug.Log("cells[" + i + ", " + j].value = " + cells[i, j].value);
            win = false;
            break;
        }
    }
    if (horizontalForCheck.Count() != horizontalForCheck.Distinct().Count())
    {
        Debug.Log(i);
        win = false;
    }
}
}

```

Потім те саме робиться із вертикальними стовпчиками:

bool CheckVerticalLines()

```

{
    List<int> verticalForCheck = new List<int>();

    for (int j = 0; j < width; j++)
    {
        verticalForCheck.Clear();
        for (int i = 0; i < height; i++)
        {
            if (cells[i, j].value > 0 && cells[i, j].value < 10)
                verticalForCheck.Add(cells[i, j].value);
            else return false;
        }
        if (verticalForCheck.Count() != verticalForCheck.Distinct().Count())
        {
            return false;
        }
    }

    return true;
}

```

І зрештою перевірку проходять окремі квадрати розмірами 3х3:

```

void CheckSquares()
{
    List<int> squareForCheck = new List<int>();
    for (int squareI = 0; squareI < height - 2; squareI += 3)
    {
        for (int squareJ = 0; squareJ < width - 2; squareJ += 3)
        {
            //one square
            squareForCheck.Clear();

            int index = 0;
            for (int i = squareI; i < squareI + 3; i++)
            {
                for (int j = squareJ; j < squareJ + 3; j++)

```

```

        {
            if (cells[i, j].value > 0 && cells[i, j].value < 10)
                squareForCheck.Add(cells[i, j].value);
            else win = false;
            index++;
        }
    }
    if (squareForCheck.Count() != squareForCheck.Distinct().Count())
    {
        win = false;
    }
}
}
}

```

Якщо кожна перевірка виявляється успішною, дається нагорода.

Для побудови лабіринту було використано процедурну генерацію. Для цього створено п'ять класів: MazeSpawner, MazePlayerController, MazeGeneratorCell, MazeGenerator та MazeCell.

Префаб клітинки лабіринту – це дві лінії, що відіграють роль стінок.

У MazeCell зберігаються лише посилання на ліву та нижню стіну, а у MazeGeneratorCell – чи дані стіни видимі, чи ні, індекси та відстань до старту (у кількості клітинок).

MazePlayerController відповідає за управління кулею, яким потрібно знайти вихід.

MazeGenerator – найважливіший клас, він генерує лабіринт. Спочатку просто створюється двовимірний масив клітинок. Потім з країв ховаються зайві стіни, які не будуть закриватися іншими клітинками:

```

public MazeGeneratorCell[,] GenerateMaze(int width, int height)
{
    MazeGeneratorCell[,] maze = new MazeGeneratorCell[width, height];

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

    for (int x = 0; x < maze.GetLength(0); x++)
    {
        maze[x, height - 1].wallLeft = false;
    }
    for (int y = 0; y < maze.GetLength(1); y++)
    {

```

```

        maze[width - 1, y].wallBottom = false;
    }

    RemoveWalls(maze, width, height);
    PlaceMazeExit(maze, width, height);

    return maze;
}

```

Метод RemoveWalls відповідає за те, щоб сформувати прохід від початку до виходу. Прохід починається з першої комірки, вона позначається відвіданою. Далі створюється стек із невідведаних сусідніх комірок, серед яких навмання обирається одна. Вона також позначається як відвідана та між нею та попередньою клітинкою забирається стіна.

Обрана клітинка додається до стеку. Ці дії повторюються до тих пір, поки не перестануть знаходитися сусідні непройдені комірки. Тоді зі стеку виймається остання клітинка, в якій знову перевіряється наявність сусідів. Так відбувається, поки у стеку є елементи. Коли він стає пустим – це означає, що лабіринт повністю сформовано.

```

private void RemoveWalls(MazeGeneratorCell[,] maze, int width, int height)
{
    MazeGeneratorCell current = maze[0, 0];
    current.visited = true;
    current.DistanceFromStart = 0;

    Stack<MazeGeneratorCell> stack = new Stack<MazeGeneratorCell>();
    do
    {
        List<MazeGeneratorCell> unvisitedNeighbours = new List<MazeGeneratorCell>();

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

```

Йде перевірка усіх чотирьох сусідів на те, чи вони були відвідані.

```

//check all 4 neighbours
if (x > 0 && !maze[x - 1, y].visited)
    unvisitedNeighbours.Add(maze[x - 1, y]);
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]);

//if there are unvisited neighbours - pick random and go there
if (unvisitedNeighbours.Count > 0)
{

```

```

        MazeGeneratorCell chosen =
            unvisitedNeighbours[Random.Range(0, unvisitedNeighbours.Count)];
        RemoveWalls(current, chosen);
        chosen.visited = true;
        current = chosen;
        stack.Push(current);
        current.DistanceFromStart = stack.Count;
    }
    else
    {
        current = stack.Pop();
    }
} while (stack.Count > 0);
}

```

Залишається знайти, куди помістити вихід:

```

private void PlaceMazeExit(MazeGeneratorCell[,] maze, int width, int height)
{
    MazeGeneratorCell exit = maze[0, 0];

    for (int x = 0; x < maze.GetLength(0); x++)
    {
        if (maze[x, height - 2].DistanceFromStart > exit.DistanceFromStart)
            exit = maze[x, height - 2];

        if (maze[x, 0].DistanceFromStart > exit.DistanceFromStart)
            exit = maze[x, 0];
    }

    for (int y = 0; y < maze.GetLength(1); y++)
    {
        if (maze[width - 2, y].DistanceFromStart > exit.DistanceFromStart)
            exit = maze[width - 2, y];

        if (maze[0, y].DistanceFromStart > exit.DistanceFromStart)
            exit = maze[0, y];
    }
}

```

І, нарешті, розміщується вихід шляхом видалення відповідної лінії:

```

if (exit.X == 0)
    exit.wallLeft = false;
else if (exit.Y == 0)
    exit.wallBottom = false;
else if (exit.X == width - 2)
    maze[exit.X + 1, exit.Y].wallLeft = false;
else if (exit.Y == height - 2)
    maze[exit.X, exit.Y + 1].wallBottom = false;
}

```

Висновки до другого розділу

Перераховано усі головоломки та показано на малюнках прототипи.

Описано архітектуру за допомогою схем варіантів використання, активності, класів, компонентів та розгортання.

Розписано головні етапи реалізації гри з описом найбільш вагомих або складних за виконанням методів коду.

РОЗДІЛ 3. ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

3.1 Порядок встановлення та налаштування системи

«What are the ghosts talking about?» - це додаток, який було створено за допомогою рушія Unity2D. Після побудови релізної версії, гру можна буде викладати на сайтах, де гравці зможуть завантажити файл та встановити на свій комп'ютер.

Збірку виконано на такі операційні системи, як Windows, Mac та Linux.

Щоб додаток коректно працював та зберігав достатню частоту кадрів у секунду, є такі мінімальні вимоги до системи:

- Операційна система: Windows 7
- 4 Гб оперативної пам'яті
- Графічний процесор з підтримкою DirectX 9
- 3 Гб вільного місця на пристрої

Рекомендовані вимоги до системи:

- Операційна система: Windows 10
- 8 Гб оперативної пам'яті
- Графічний процесор з підтримкою DirectX 10
- 5 Гб вільного місця на пристрої

3.2 Інтерфейс та порядок роботи з програмним додатком

Представлено можливі дії гравця на трьох рівнях (першому, другому та шостому).

При завантаженні будь-якого рівня можна ходити персонажем, натискаючи кнопки A та D. Esc викличе меню паузи. Зліва над інвентарем є кнопка, що викликає підказку.

На першому рівні знаходиться ванна кімната (рис.3.1). Одразу можна взяти пусту склянку. Якщо натиснути на умивальник, відкриється головоломка з трубами (рис.3.2), які потрібно повернути в правильні

положення. Після цього можна набрати води у склянку та примінити її на кактус. На ньому розцвіте квітка, у якій буде фрагмент дзеркала.

Ванна викликає міні-гру з качками (рис.3.3), де на них потрібно натиснути в правильному порядку та певну визначену кількість разів. Якщо роздивитися кімнату, то можна помітити підказку – картини на стіні. Кольори рамки та квіток вказують на колір пір'я та дзьоба відповідно, а кількість бутонів – на те, скільки разів треба натиснути на цю качку. Якщо виконати усе правильно, знаходиться ще один шматочок дзеркала.

На шафі стоять п'ятнашки, які викликають усім знайому гру (рис.3.4.), яка додає останній фрагмент.

Після того, як гравець ремонтує дзеркало, випадає ключ, що відчиняє шафу з картиною.



Рисунок 3.1 - Ванна кімната

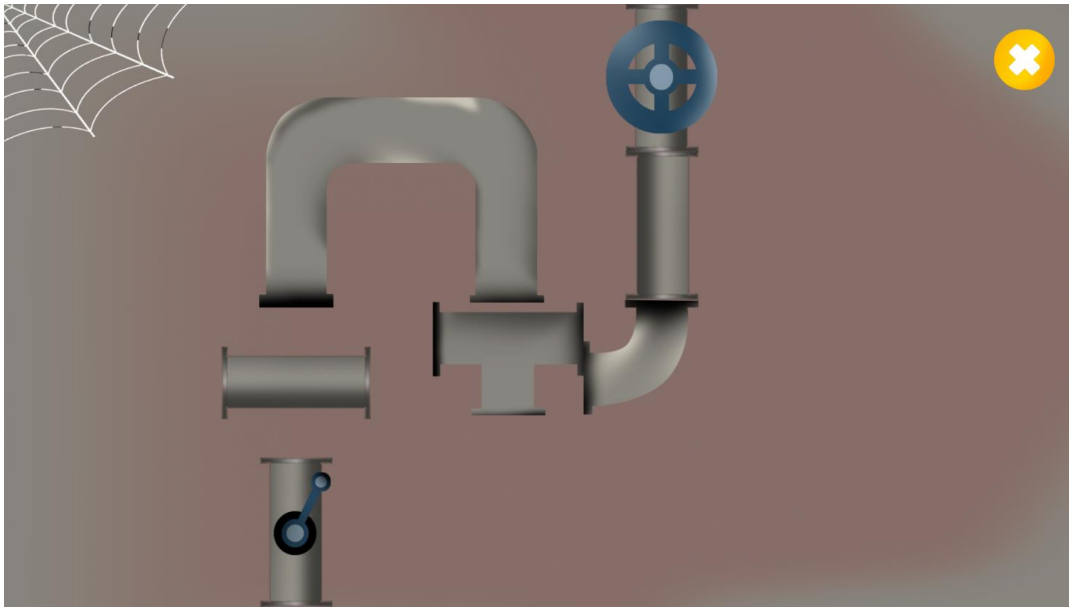


Рисунок 3.2 - Труби

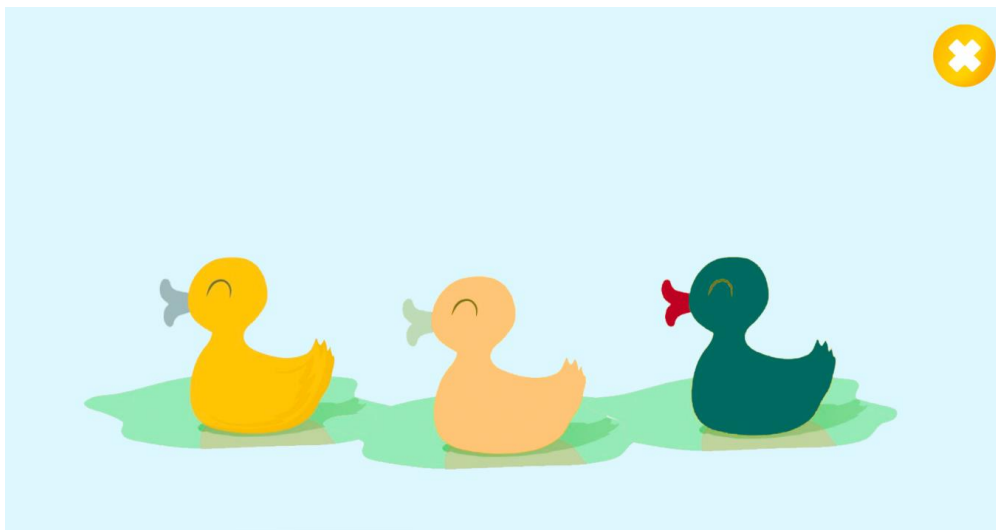


Рисунок 3.3 - Качки

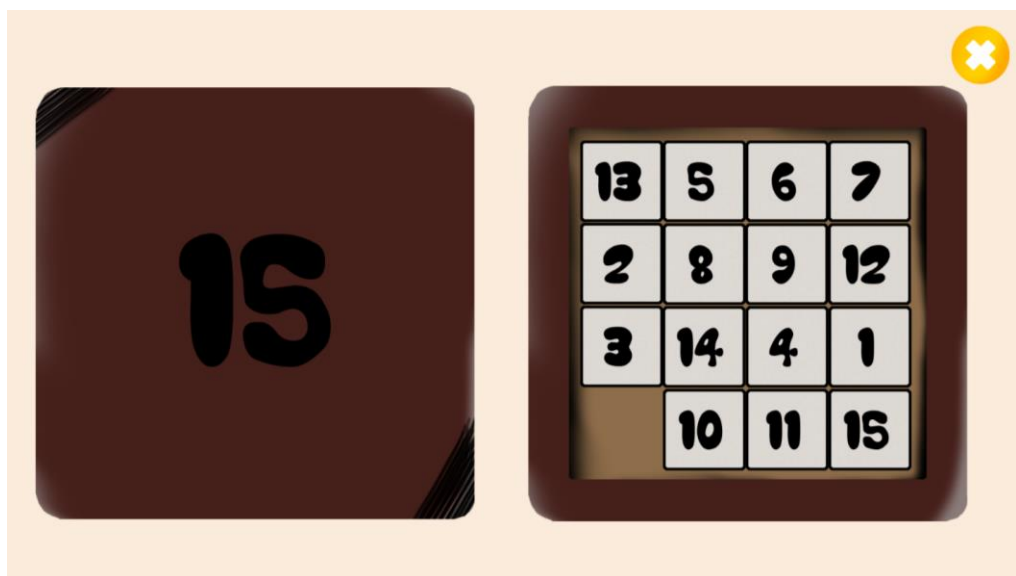


Рисунок 3.4 - П'ятнашки

На другому рівні можна спочатку підібрати стрілки від годинника, клікнувши на тумбочку, потім, обравши їх у інвентарі, додати їх на годинник. Після вищеописаних дій можна змінювати пору року у кімнаті, клікнувши на годинник.

Щоб пройти дану кімнату, потрібно зібрати чотири предмета (метелик, кістка, листок, зірка) та вставити їх у шафу, де знайдеться фрагмент картини.

Осінь (рис.3.5) можна підібрати листок з дерева та зіграти зі скелетом у міні-гру зі стаканами, але лише якщо дати йому капелюх з пером. У головоломці (рис.3.6) потрібно обрати потрібну склянку, під якою схований предмет – кістка.



Рисунок 3.5 - Другий рівень, осінь



Рисунок 3.6 - Предметом під склянкою

Зимою (рис.3.7), натиснувши на дерево, отримується гілка. Ялинка викликає головоломку (рис.3.8), де, натискаючи на ліхтарі, потрібно увімкнути світло в усіх них, в якості винагороди дається зірка. Подарунок – також логічна міні-гра (рис.3.9), де гравець має натиснути на бантики у порядку зростання їх вузлів, приз - клітка.



Рисунок 3.7 - Другий рівень, зима



Рисунок 3.8 - Ліхтарі на ялинці

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
						45
Змн.	Арк.	№ докум.	Підпис	Дата		



Рисунок 3.9 - Головоломка з подарунком

Весною (рис.3.10) за допомогою гілки можна зняти з манекену капелюх, а сачком – зловити метелика.



Рисунок 3.10 - Другий рівень, весна

Літом (рис.3.11) кліткою можна спіймати сову, з якої випаде пір'їнка. Якщо поєднати капелюх та перо, то можна пройти гру у скелета зимою.

Кліком на коробку викликається головоломка з метеликом (рис.3.12), де потрібно натискати на ліву або праву кнопку, в залежності від того, з якої сторони від малюнку знаходиться наступний ліхтар. В нагороду дається сачок.



Рисунок 3.11 - Другий рівень, літо

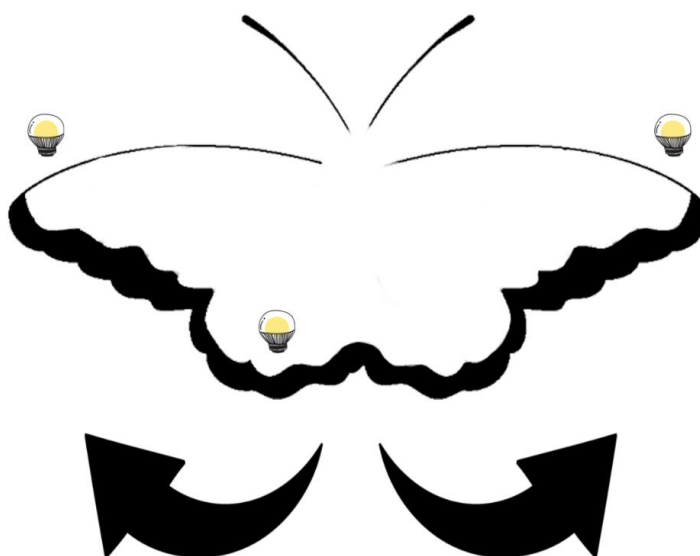


Рисунок 3.12 - Головоломка з метеликом

На шостому рівні (рис.3.13) розташована комора.

При кліку на пробірки викликається головоломка з сортуванням води (рис.3.14), де потрібно переливати різнокольорову рідину так, щоб кожна посудина була або порожня, або містила лише один колір. В нагороду даються колби з водою різного кольору.

При кліку на скляні пляшки відкривається задачка з переливанням води, де потрібно відміряти певну кількість у одну посудину (рис.3.15).

Після цього з'являється п'єдестал, на який потрібно виставити колби у порядку зростання їх ваги, зваживши попередньо на вагах (рис.3.16). Потім з'являється фрагмент картини, що означає, що рівень пройдений.



Рисунок 3.13 - Шостий рівень

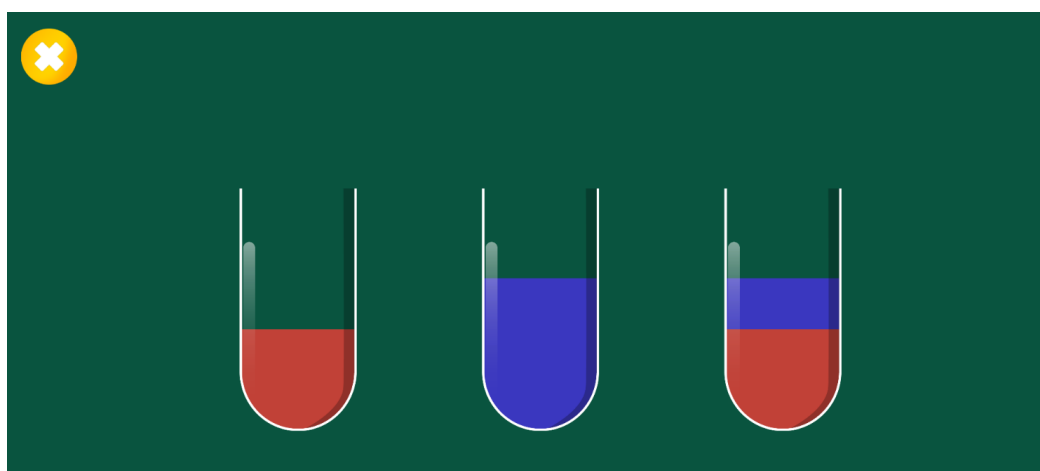


Рисунок 3.14 - Головоломка з сортуванням води



Рисунок 3.15 - Головоломка з переливанням води

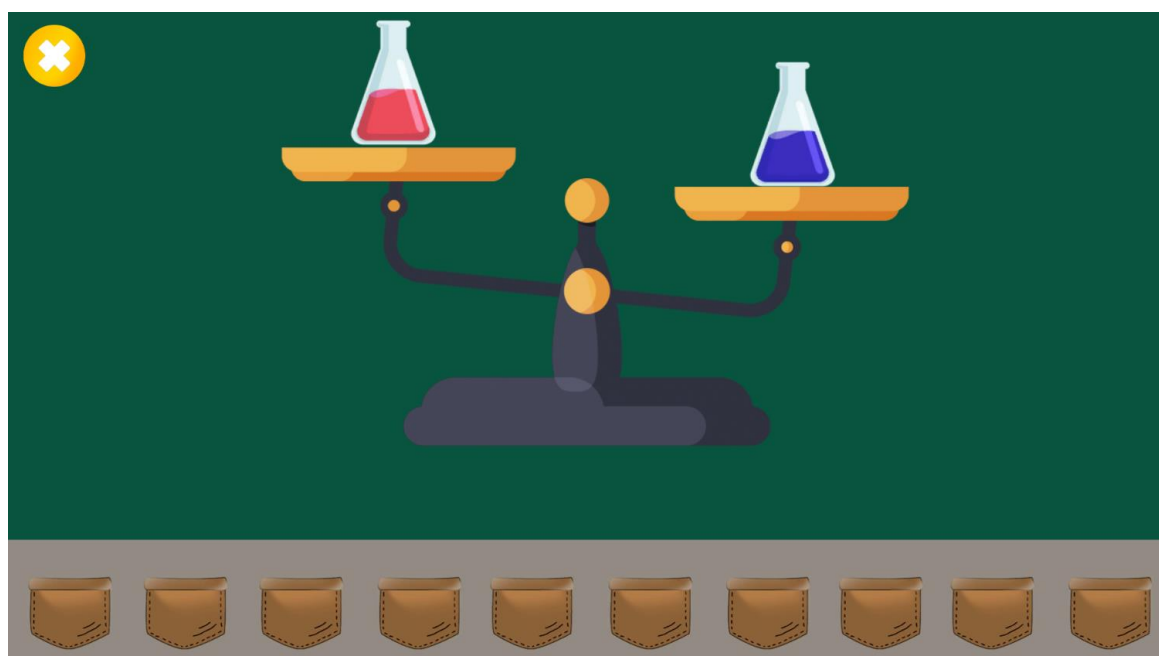


Рисунок 3.16 - Ваги

Окрім вище зазначених дій, головна героїня може казати певні фрази при кліку на деякі предмети.

На першому рівні є діалоги при натиску на картини на стіні (рис.3.17), порожньою склянкою на умивальник, якщо ще не була пройдена головоломка з трубами (рис.3.18) та рушник (рис.3.19).



Рисунок 3.17 - Фраза при взаємодії із картинами



Рисунок 3.18 - Фраза при взаємодії зі зламаним умивальником



Рисунок 3.19 - Фраза при взаємодії із рушником

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		50

На шостому рівні є фрази при кліку на мишу (рис.3.20), на полицки з припасами (рис.3.21) та пляшку (рис.3.22), на полотно (рис.3.23) та на підлогу (рис.3.24).



Рисунок 3.20 - Фраза при взаємодії з мишею



Рисунок 3.21 - Фраза при взаємодії із полицками



Рисунок 3.22 - Фраза при взаємодії з пляшкою



Рисунок 3.23 - Фраза при взаємодії із полотном



Рисунок 3.24 - Фраза при взаємодії з підлогою

3.3 Тестування роботи програмного забезпечення

Проведено тестування системи головоломок для виявлення та усунення недоліків. Представлено пункти, які було протестовано – таблиця 3.1.

Таблиця 3.1

Головоломка або кімната	Вхідні дані	Очікувані дані	Спостережені дані
Перший рівень	Натиск кнопки «Esc»	Відкривається меню паузи	Як і очікувалося
Перший рівень	Натиск на кнопку підказок	З'являється привид, який каже підказку	Як і очікувалося
Перший рівень	Клік на предмет, який можна підібрати	Предмет додається до інвентаря, інтерфейс оновлюється	Як і очікувалося
Перший рівень	Клік на предмет, який відкриває головоломку	Відкривається інтерфейс головоломки	Як і очікувалося
Перший рівень	Клік на предмет, який викликає фразу головної героїні	З'являється інтерфейс діалогу, де пишеться фраза, і після деякого часу зникає	Як і очікувалося
Головоломка з трубами	Клік на трубу	Труба повертається на 90 градусів проти годинникової стрілки	Як і очікувалося
	Клік на кнопку з хрестиком	Головоломка закривається	Як і очікувалося
Головоломка з качками	Натиск на качок в правильному порядку	З'являється нагорода – шматок дзеркала, кнопка закриття пропадає	Як і очікувалося
	Клік на кнопку з хрестиком	Головоломка закривається	
	Клік на дзеркало	Шматок дзеркала додається до інвентаря, головоломка закривається	Як і очікувалося

П'ятнашки	Відкриття головоломки	Комірки генеруються правильно	Як і очікувалося
	Клік на комірку, біля якої немає порожньої	Нічого не відбувається	Як і очікувалося
	Клік на комірку, біля якої є порожня	Обрана комірка переміщується на порожнє місце	Як і очікувалося
	Вирішення головоломки	З'являється шматок дзеркала	Як і очікувалося
	Клік на дзеркало	Шматок дзеркала додається до інвентаря, головоломка закривається	Як і очікувалося
Другий рівень	Натиск кнопки «Esc»	Відкривається меню паузи	Як і очікувалося
Другий рівень	Натиск на кнопку підказок	З'являється привид, який каже підказку	Як і очікувалося
Другий рівень	Клік на предмет, який можна підібрати	Предмет додається до інвентаря, інтерфейс оновлюється	Як і очікувалося
Другий рівень	Клік на предмет, який відкриває головоломку	Відкривається інтерфейс головоломки	Як і очікувалося
Другий рівень	Клік на предмет, який викликає фразу головної героїні	З'являється інтерфейс діалогу, де пишеться фраза, і після деякого часу зникає	Як і очікувалося
Ліхтарики на ялинці	Клік на ліхтарик	Обраний ліхтарик та його сусіди міняють стан: якщо світилися – вимикаються, та навпаки	Як і очікувалося
	Включення усіх ліхтариків	Відключається можливість клікати на ліхтарики. З'являється	Як і очікувалося

		нагорода - зірка	
	Клік на зірку	Зірка додається до інвентаря, головоломка закривається	Як і очікувалося
	Клік на кнопку з хрестиком	Головоломка закривається	Як і очікувалося
Головоломка з подарунком	Клік на бантик в правильному порядку	Бантик пропадає	Як і очікувалося
	Клік на бантик в неправильному порядку	Усі бантики знову стають видимі	Як і очікувалося
	Вирішення головоломки	З'являється нагорода - клітка	Як і очікувалося
	Натиск на клітку	Клітка додається в інвентар, головоломка закривається	Як і очікувалося
	Клік на кнопку з хрестиком	Головоломка закривається	Як і очікувалося
Головоломка з метеликом	Клік на кнопку зліва	Якщо наступний ліхтарик зліва, він вмикається. Якщо ж ні, то усі вимикаються	Як і очікувалося
	Клік на кнопку справа	Якщо наступний ліхтарик справа, він вмикається. Якщо ж ні, то усі вимикаються	Як і очікувалося
	Правильно вирішено головоломку	З'являється нагорода - сачок	Як і очікувалося
	Клік на сачок	Сачок додається в інвентар, головоломка закривається	Як і очікувалося
	Клік на кнопку з хрестиком	Головоломка закривається	Як і очікувалося
	Класична головоломка з	Клік на неправильну	Гра починається заново

предметом під склянкою	склянку		
	Клік на правильну склянку	З'являється нагорода - кістка	Як і очікувалося
	Клік на кістку	Кістка додається в інвентар, головоломка закривається	Як і очікувалося
Третій рівень	Натиск кнопки «Esc»	Відкривається меню паузи	Як і очікувалося
Третій рівень	Натиск на кнопку підказок	З'являється привид, який каже підказку	Як і очікувалося
Третій рівень	Клік на предмет, який можна підібрати	Предмет додається до інвентаря, інтерфейс оновлюється	Як і очікувалося
Третій рівень	Клік на предмет, який відкриває головоломку	Відкривається інтерфейс головоломки	Як і очікувалося
Третій рівень	Клік на предмет, який викликає фразу головної героїні	З'являється інтерфейс діалогу, де пишеться фраза, і після деякого часу зникає	Як і очікувалося
Головоломка з відмірюванням води	Клік на неактивну пляшку	Якщо була обрана певна пляшку, вода з неї переливається. В іншому випадку обрана пляшка стає активною	Як і очікувалося
	Клік на активну пляшку	Пляшка стає неактивною	Як і очікувалося
	Клік на кнопку з хрестиком	Головоломка закривається	Як і очікувалося
Сортування води	Клік на неактивну ємність	Якщо була обрана певна ємність, вода з неї переливається. В іншому випадку обрана ємність стає активною	Як і очікувалося
	Клік на активну ємність	Ємність стає неактивною	Як і очікувалося

	Клік на кнопку з хрестиком	Головоломка закривається	Як і очікувалося
--	----------------------------	--------------------------	------------------

При тестуванні було виявлено багато помилок, незручностей та недоліків гри. Наприклад, деякі об'єкти проходили крізь стіни. Це було виправлено шляхом додавання відповідних колайдерів.

Було багато помилок з UI, певні елементи перекривали інші. Це було вирішено виправленням налаштувань шарів інтерфейсу.

В багатьох головоломках було змінено час виконання дій. Наприклад, в п'ятнашках спочатку занадто повільно перемішувалися комірки, а в танцювальному апараті – навпаки, занадто швидко миготіли стрілки. В тетрісі підкорегована швидкість падіння фігури.

В деяких міні-іграх було зменшено або підвищено складність. Так, в грі з сортуванням води спочатку були занадто легкі завдання, а в судоку – занадто складні, на які треба було витратити дуже багато часу.

Після цього було проведено фінальне тестування, яке не виявило проблем або недоліків.

Висновки до третього розділу:

У третьому розділі описано порядок встановлення, мінімальні та рекомендовані налаштування системи.

Також представлено інтерфейс та порядок роботи, розписано які дії може виконати гравець на трьох рівнях.

Описано процес тестування: які дії повинні до чого призвести, та що відбувалося насправді. Виявлено деякі помилки, неточності та недоліки. Їх було проаналізовано та виправлено.

ВИСНОВКИ

Було досягнуто мети та створено працюючу систему головоломок для гри «What are the ghosts talking about?».

Переглянуто три популярні гри жанрів, у яких було заплановано проект, визначено їх переваги та недоліки. Переглянуто приклади головоломок та міні-ігор та прийнято до уваги нюанси. Проаналізовано потреби гравця та поставлено задачу проекту, а саме який в загальному має бути функціонал.

Створено концепт-документ, в якому визначено жанр та аудиторію, особливості, геймплей, сюжет та платформу.

Складено перелік усіх головоломок у грі та наведено невеликий опис та скріншоти на стадії проектування прототипу.

Описано архітектуру за допомогою схем варіантів використання, активності, класів, компонентів та розгортання.

Розписано головні етапи реалізації гри з описом найбільш вагомих або складних за виконанням методів коду.

Показано інтерфейс та пояснено порядок роботи, детально представлено дії, які може виконати гравець на трьох рівнях.

Описано процес тестування: розказано, що саме було протестовано і на що звернуто увагу, що було виправлено та вдосконалено.

В додатках приведено лістинг важливих функцій.

Створено звіт та презентацію, і зрештою проект був готовий. У результаті отримано робочу програму, яка справно виконує усі бажані дії та коректно працює при різному поводженні персонажа.

					КН.КР.Б – 122 – 23 – ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		58

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Патерни проектування [Електронний ресурс] – Режим доступу до ресурсу: <https://refactoring.guru/uk/design-patterns>
2. Робота з шейдером [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=zO2q4BHkGQU&list=PLdE10202Nk1MgtkCKysZQE5B1vXHBRbGp&index=15>
3. Виникнення та психологія логічних ігор [Електронний ресурс] – Режим доступу до ресурсу: <https://plato.stanford.edu/entries/logic-games/>
4. Як правильно впорядковувати файли в Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://unity.com/how-to/organizing-your-project>
5. Розробка основної механіки для п'ятнашок [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=b4pJ8nbQO0Q&list=PLdE10202Nk1MgtkCKysZQE5B1vXHBRbGp&index=6>
6. Розробка основної механіки тетрісу [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=T5P8ohdxDjo&list=PLdE10202Nk1MgtkCKysZQE5B1vXHBRbGp&index=9>
7. Як можна малювати лінію за допомогою Line Renderer у Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=5ZBynjAsfwI&list=PLdE10202Nk1MgtkCKysZQE5B1vXHBRbGp&index=2>
8. Типи логічних ігор [Електронний ресурс] – Режим доступу до ресурсу: <https://clontz.org/puzzles/types/>
9. Робота з UI у Unity2D [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=IuuKUaZQiSU>
10. Як тестувати гру [Електронний ресурс] – Режим доступу до ресурсу: <https://gamedevacademy.org/game-playtest-tutorial/>
11. Пересування персонажа у 2D [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=K1xZ-rycYY8>

12. Створення пазлу у Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=KHZFpRL3Xzc>
13. Робота із зіткненням предметів [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=gAB64vfbrhI>
14. Як впорядковувати слої зображень у 2D графіці [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=9vBbg1-Bxcw>
15. Як зберігати прогрес, використовуючи PlayerPrefs [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=y7RPVvwjrsA>
16. Як зберігати прогрес, використовуючи текстові файли [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=XOjd_qU2Ido
17. Механіка drag&drop у Unity2D [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=xFAT-R2_aXM
18. Використання корутин у Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=5L9ksCs6MbE>
19. Наслідування класів у Unity [Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=F7Wu6_uzD1I
20. Як зберігати об'єкти як Scriptable Object у Unity [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=aPXvoWVabPY>

ДОДАТКИ

Найбільш важливі функції та класи

Файл Interactable:

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

public abstract class Interactable : MonoBehaviour
{
    protected Transform player;

    void Start()
    {
        player = PlayerManager.instance.player.transform;
    }

    public virtual void Interact()
    {
        Debug.Log("Interact");
    }

    public void OnMouseDown()
    {
        Interact();
    }
}
```

Файл DragAndDrop:

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

public class DragAndDrop : MonoBehaviour
{
    protected Vector3 mousePositionOffset;
    protected Rigidbody2D rb;

    protected virtual void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }

    protected Vector3 GetMouseWorldPosition()
    {
        return Camera.main.ScreenToWorldPoint(Input.mousePosition);
    }

    protected virtual void OnMouseDown()
    {
        mousePositionOffset = transform.position - GetMouseWorldPosition();
    }

    protected virtual void OnMouseDrag()
    {
        Vector2 position = GetMouseWorldPosition() + mousePositionOffset;
        rb.MovePosition(position);
    }
}
```

Файл OpenPuzzle:

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

public class OpenPuzzle : Interactable
{
    [SerializeField] private GameObject canvasUI;
    [SerializeField] private GameObject objects;

    [SerializeField] private GameObject puzzleUIEasy;
    [SerializeField] private GameObject puzzleUINormal;
    [SerializeField] private GameObject puzzleUIHard;

    public bool win = false;

    public override void Interact()
    {
        base.Interact();
        ShowPuzzleUI();
        FindObjectOfType<PauseMenu>().SetCanBePaused(false);
    }

    void ShowPuzzleUI()
    {
        if (!win)
        {
            if (puzzleUIEasy != null && puzzleUINormal != null && puzzleUIHard != null)
            {
                if (FindObjectOfType<CameraMovement>() != null)
                    FindObjectOfType<CameraMovement>().DisableMovement();
                if (canvasUI != null)
                    canvasUI.SetActive(false);
                objects.SetActive(false);

                string difficulty = PlayerManager.instance.difficultyLevel;

                if (difficulty == "Easy")
                {
                    puzzleUIEasy.SetActive(true);
                }
                else if (difficulty == "Normal")
                {
                    puzzleUINormal.SetActive(true);
                }
                else if (difficulty == "Hard")
                {
                    puzzleUIHard.SetActive(true);
                }
            }
        }
    }
}
```

Файл FifteenPuzzleParent:

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

public class FifteenPuzzleParent : MonoBehaviour
{
    [SerializeField]
    private OpenPuzzle openPuzzle;

    [SerializeField]
    private InstantiateDialogueTip tipsDialogue;

    public int size;
    public FifteenPuzzleCell cellPrefab;
    public FifteenPuzzleCell[,] cells;

    [SerializeField]
    private GameObject mirror;

    public Sprite[] sprites;

    public bool win;

    private Vector2 lastMove = new Vector2();

    private void Start()
    {
        cells = new FifteenPuzzleCell[size, size];

        Init();
        for (int i = 0; i <= 500; i++)
            Shuffle();
    }

    void Init()
    {
        int index = 0;
        for (int y = size - 1; y >= 0; y--)
        {
            for (int x = 0; x < size; x++)
            {
                FifteenPuzzleCell cell = Instantiate(cellPrefab, new Vector2(x, y), Quaternion.identity, transform);
                cell.Generate(x, y, index + 1, sprites[index], ClickToSwap);
                cells[x, y] = cell;
                index++;
            }
        }
    }

    void ClickToSwap(int x, int y)
    {
        int dx = GetDx(x, y);
        int dy = GetDy(x, y);

        Swap(x, y, dx, dy);
    }

    void Swap(int x, int y, int dx, int dy)
    {
        var from = cells[x, y];
```

```
var target = cells[x + dx, y + dy];
```

```
//swap these 2 cells  
cells[x, y] = target;  
cells[x + dx, y + dy] = from;
```

```
int tmpIndex = from.indexCurrent;  
from.indexCurrent = target.indexCurrent;  
target.indexCurrent = tmpIndex;
```

```
//update position  
from.UpdatePos(x + dx, y + dy);  
target.UpdatePos(x, y);
```

```
CheckWin();  
if (win)  
    Win();  
}
```

```
int GetDx(int x, int y)  
{  
    //is right empty  
    if (x < size - 1 && cells[x + 1, y].IsEmpty())  
    {  
        return 1;  
    }  
}
```

```
//is left empty  
if (x > 0 && cells[x - 1, y].IsEmpty())  
{  
    return -1;  
}
```

```
return 0;  
}
```

```
int GetDy(int x, int y)  
{  
    //is top empty  
    if (y < size - 1 && cells[x, y + 1].IsEmpty())  
    {  
        return 1;  
    }  
}
```

```
//is bottom empty  
if (y > 0 && cells[x, y - 1].IsEmpty())  
{  
    return -1;  
}
```

```
return 0;  
}
```

```
void Shuffle()  
{  
    for (int i = 0; i < size; i++)  
    {  
        for (int j = 0; j < size; j++)  
        {  
            if (cells[i, j].IsEmpty())  
            {  
                Vector2 pos = GetValidMove(i, j);  
                Swap(i, j, (int)pos.x, (int)pos.y);  
            }  
        }  
    }  
}
```

```

    }
  }
}

```

```

Vector2 GetValidMove(int x, int y)
{
    Vector2 pos = new Vector2();
    do
    {
        int n = Random.Range(0, size);
        if (n == 0)
            pos = Vector2.left;
        else if (n == 1)
            pos = Vector2.right;
        else if (n == 2)
            pos = Vector2.up;
        else
            pos = Vector2.down;
    } while (!(IsValidRange(x + (int)pos.x) && IsValidRange(y + (int)pos.y)) || IsRepeatMove(pos));
}

```

```

    lastMove = pos;

```

```

    return pos;
}

```

```

bool IsValidRange(int n)
{
    return n >= 0 && n <= size - 1;
}

```

```

bool IsRepeatMove(Vector2 pos)
{
    return pos * (-1) == lastMove;
}

```

```

public void CheckWin()
{
    win = true;
    foreach (FifteenPuzzleCell cell in cells)
    {
        if (cell.index != cell.indexCurrent)
            win = false;
    }
}

```

```

public void Win()
{
    Debug.Log("Win");
    openPuzzle.win = true;
}

```

```

tipsDialogue.unacceptableNodes.Add(5);
tipsDialogue.unacceptableNodes.Add(15);

```

```

    mirror.SetActive(true);
    Destroy(gameObject);
}
}

```

Файл JigsawPuzzlePiece:

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

public class JigsawPuzzlePiece : DragAndDrop
{
    [SerializeField] private GameObject puzzlePieceCheck;
    public bool isCorrect = false;

    private JigsawPuzzleParent parent;

    protected override void Start()
    {
        base.Start();
        parent = FindObjectOfType<JigsawPuzzleParent>();
    }

    protected override void OnMouseDown()
    {
        if (!isCorrect)
            base.OnMouseDown();
    }

    protected override void OnMouseDrag()
    {
        if (!isCorrect)
            base.OnMouseDrag();
    }

    private void OnMouseUp()
    {
        float distance = Vector3.Distance(puzzlePieceCheck.transform.position, transform.position);
        if (distance < 1)
        {
```

```

        transform.position = puzzlePieceCheck.transform.position;

        isCorrect = true;

        parent.CheckWin();

    }

}

}

```

Файл SudokuParent:

```

using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class SudokuParent : MonoBehaviour
{
    [SerializeField] GameObject showPrizeWin;

    private bool isAbleToCheckWin = false;
    private bool win = false;

    private int[,] matrixEasy = new int[9, 9]
    {
        { 0, 3, 5, 2, 6, 9, 7, 8, 1 },
        { 6, 8, 2, 5, 7, 1, 4, 9, 3 },
        { 1, 9, 7, 8, 3, 4, 5, 0, 2 },
        { 8, 2, 6, 1, 9, 5, 3, 4, 7 },
        { 3, 7, 4, 6, 8, 2, 9, 1, 5 },
        { 9, 5, 1, 7, 4, 3, 6, 2, 8 },
        { 5, 1, 9, 3, 2, 6, 8, 7, 4 },
        { 2, 4, 8, 9, 5, 7, 1, 3, 6 },
        { 7, 6, 3, 4, 1, 8, 2, 5, 9 }
    };

    private int[,] matrixNormal = new int[9, 9]
    {

```

```

        { 4, 3, 5, 2, 6, 9, 7, 8, 1 },
        { 6, 8, 2, 5, 7, 1, 4, 9, 3 },
        { 1, 9, 7, 8, 3, 4, 5, 6, 2 },
        { 8, 2, 6, 1, 9, 5, 3, 4, 7 },
        { 3, 7, 4, 6, 8, 2, 9, 1, 5 },
        { 9, 5, 1, 7, 4, 3, 6, 2, 8 },
        { 5, 1, 9, 3, 2, 6, 8, 7, 4 },
        { 2, 4, 8, 9, 5, 7, 1, 3, 6 },
        { 7, 6, 3, 4, 1, 8, 2, 5, 9 }
    };

```

```

    private int[,] matrixHard = new int[9, 9]
    {
        { 4, 3, 5, 2, 6, 9, 7, 8, 1 },
        { 6, 8, 2, 5, 7, 1, 4, 9, 3 },
        { 1, 9, 7, 8, 3, 4, 5, 6, 2 },
        { 8, 2, 6, 1, 9, 5, 3, 4, 7 },
        { 3, 7, 4, 6, 8, 2, 9, 1, 5 },
        { 9, 5, 1, 7, 4, 3, 6, 2, 8 },
        { 5, 1, 9, 3, 2, 6, 8, 7, 4 },
        { 2, 4, 8, 9, 5, 7, 1, 3, 6 },
        { 7, 6, 3, 4, 1, 8, 2, 5, 9 }
    };

```

```

    private int[,] matrix;

```

```

    [SerializeField] private GameObject parentPanel;

```

```

    [SerializeField] private GameObject cellPrefab;

```

```

    [SerializeField] int width = 9;

```

```

    [SerializeField] int height = 9;

```

```

    SudokuCell[,] cells;

```

```

    void Start()

```

```

    {

```

```

        string difficulty = PlayerManager.instance.difficultyLevel;

```

```
    if (difficulty == "Easy")
    {
        matrix = matrixEasy;
    }
    else if (difficulty == "Normal")
    {
        matrix = matrixNormal;
    }
    else if (difficulty == "Hard")
    {
        matrix = matrixHard;
    }
```

```
    cells = new SudokuCell[height, width];
```

```
    for (int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            SudokuCell cell = Instantiate(cellPrefab, parentPanel.transform).GetComponent<SudokuCell>();
            cells[i, j] = cell;
            cell.indexI = i;
            cell.indexJ = j;
            if (matrix[i, j] != 0)
            {
                cell.value = matrix[i, j];
                cell.GetComponentInChildren<TMP_InputField>().textComponent.color = Color.black;
                cell.GetComponentInChildren<TMP_InputField>().text = cell.value.ToString();

                cell.GetComponentInChildren<TMP_InputField>().enabled = false;
            }
            else
            {
```

```
        cell.GetComponentInChildren<TMP_InputField>().textComponent.color = Color.red;
    }
    cell.parent = this;
}
}
```

```
    isAbleToCheckWin = true;
}
```

```
public void CheckWin()
{
    if (isAbleToCheckWin)
    {
        Debug.Log("CheckWin");
        win = true;

        //check horizontal lines
        CheckHorizontalLines();

        if (win)
        {
            Debug.Log("CheckHorizontalLines passed");
            //check vertical lines
            win = CheckVerticalLines();
        }

        if (win)
        {
            Debug.Log("CheckVerticalLines passed");
            //check squares
            CheckSquares();
        }

        if (win)
```

```

    {
        Win();
    }
}

}

void CheckHorizontalLines()
{
    List<int> horizontalForCheck = new List<int>();

    for (int i = 0; i < height; i++)
    {
        horizontalForCheck.Clear();
        for (int j = 0; j < width; j++)
        {
            if (cells[i, j].value > 0 && cells[i, j].value < 10)
            {
                horizontalForCheck.Add(cells[i, j].value);
            }
            else
            {
                Debug.Log("cells[" + i + ", " + "j"].value = " + cells[i, j].value);
                win = false;
                break;
            }
        }

        if (horizontalForCheck.Count() != horizontalForCheck.Distinct().Count())
        {
            Debug.Log(i);
            win = false;
        }
    }
}

bool CheckVerticalLines()

```

```

{
    List<int> verticalForCheck = new List<int>();

    for (int j = 0; j < width; j++)
    {
        verticalForCheck.Clear();
        for (int i = 0; i < height; i++)
        {
            if (cells[i, j].value > 0 && cells[i, j].value < 10)
                verticalForCheck.Add(cells[i, j].value);
            else return false;
        }
        if (verticalForCheck.Count() != verticalForCheck.Distinct().Count())
        {
            return false;
        }
    }

    return true;
}

```

```

void CheckSquares()
{
    List<int> squareForCheck = new List<int>();
    for (int squareI = 0; squareI < height - 2; squareI += 3)
    {
        for (int squareJ = 0; squareJ < width - 2; squareJ += 3)
        {
            //one square
            squareForCheck.Clear();

            int index = 0;
            for (int i = squareI; i < squareI + 3; i++)
            {
                for (int j = squareJ; j < squareJ + 3; j++)
                {

```

```

        if (cells[i, j].value > 0 && cells[i, j].value < 10)
            squareForCheck.Add(cells[i, j].value);
        else win = false;
        index++;
    }
}

if (squareForCheck.Count() != squareForCheck.Distinct().Count())
{
    win = false;
}
}
}

void Win()
{
    bool hasWinChecked = false;
    foreach (SudokuCell cell in cells)
    {
        if (cell.value == 3 && !hasWinChecked)
        {
            cell.GetComponentInChildren<TMP_InputField>().textComponent.color = Color.red;
            hasWinChecked = true;
        }
        else
        {
            cell.GetComponentInChildren<TMP_InputField>().textComponent.color = Color.black;
        }
        cell.GetComponentInChildren<TMP_InputField>().enabled = false;
    }
    showPrizeWin.SetActive(true);
}
}

```

Файл MazeGenerator:

```
using System.Collections;
```

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

```
using UnityEngine;
```

```
public class MazeGeneratorCell
```

```
{
```

```
    public int X;
```

```
    public int Y;
```

```
    public bool wallLeft = true;
```

```
    public bool wallBottom = true;
```

```
    public bool visited = false;
```

```
    public int DistanceFromStart;
```

```
}
```

```
public class MazeGenerator
```

```
{
```

```
    public MazeGeneratorCell[,] GenerateMaze(int width, int height)
```

```
    {
```

```
        MazeGeneratorCell[,] maze = new MazeGeneratorCell[width, height];
```

```
        for (int x = 0; x < maze.GetLength(0); x++)
```

```
        {
```

```
            for (int y = 0; y < maze.GetLength(1); y++)
```

```
            {
```

```
                maze[x, y] = new MazeGeneratorCell { X = x, Y = y };
```

```
            }
```

```
        }
```

```
        for (int x = 0; x < maze.GetLength(0); x++)
```

```
        {
```

```
            maze[x, height - 1].wallLeft = false;
```

```
        }
```

```
        for (int y = 0; y < maze.GetLength(1); y++)
```

```
        {
```

```
    maze[width - 1, y].wallBottom = false;
```

```
}
```

```
RemoveWalls(maze, width, height);
```

```
PlaceMazeExit(maze, width, height);
```

```
return maze;
```

```
}
```

```
private void RemoveWalls(MazeGeneratorCell[,] maze, int width, int height)
```

```
{
```

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

```
    current.visited = true;
```

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

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

```
    do
```

```
    {
```

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

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

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

```
        //check all 4 neighbours
```

```
        if (x > 0 && !maze[x - 1, y].visited)
```

```
            unvisitedNeighbours.Add(maze[x - 1, y]);
```

```
        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]);
```

```
        //if there are unvisited neighbours - pick random and go there
```

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

```

    {
        MazeGeneratorCell chosen =
            unvisitedNeighbours[Random.Range(0, unvisitedNeighbours.Count)];
        RemoveWalls(current, chosen);
        chosen.visited = true;
        current = chosen;
        stack.Push(current);
        current.DistanceFromStart = stack.Count;
    }
    else
    {
        current = stack.Pop();
    }

} while (stack.Count > 0);
}

```

```

private void RemoveWalls(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;
    }
}

```

```

private void PlaceMazeExit(MazeGeneratorCell[,] maze, int width, int height)
{
    MazeGeneratorCell exit = maze[0, 0];
}

```

```

        for (int x = 0; x < maze.GetLength(0); x++)
        {
            if (maze[x, height - 2].DistanceFromStart > exit.DistanceFromStart)
                exit = maze[x, height - 2];

            if (maze[x, 0].DistanceFromStart > exit.DistanceFromStart)
                exit = maze[x, 0];
        }

        for (int y = 0; y < maze.GetLength(1); y++)
        {
            if (maze[width - 2, y].DistanceFromStart > exit.DistanceFromStart)
                exit = maze[width - 2, y];

            if (maze[0, y].DistanceFromStart > exit.DistanceFromStart)
                exit = maze[0, y];
        }

        if (exit.X == 0)
            exit.wallLeft = false;
        else if (exit.Y == 0)
            exit.wallBottom = false;
        else if (exit.X == width - 2)
            maze[exit.X + 1, exit.Y].wallLeft = false;
        else if (exit.Y == height - 2)
            maze[exit.X, exit.Y + 1].wallBottom = false;
        }
    }
}

```

Файл SpawnTetromino:

```

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

```

```

public class SpawnTetromino : MonoBehaviour
{
    [SerializeField] private GameObject[] tetrominoes;
    [SerializeField] public TMP_Text textRounds;
    [SerializeField] public GameObject winAward;

    [SerializeField] private OpenPuzzle openPuzzleTetris;

    void Start()
    {
        NewTetromino();
    }

    public void NewTetromino()
    {
        if(!openPuzzleTetris.win)
        {
            Instantiate(tetrominoes[Random.Range(0, tetrominoes.Length)],
                transform.position, Quaternion.identity);
        }
    }

    public void Win()
    {
        openPuzzleTetris.win = true;
        winAward.SetActive(true);

        TetrisBlock[] blocks = FindObjectsOfType<TetrisBlock>();
        foreach (TetrisBlock block in blocks)
        {
            Destroy(block.gameObject);
        }
    }
}

```

Файл BottleController:

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

public class BottleController : Interactable
{
    public Color[] bottleColors;
    public SpriteRenderer bottleMaskSR;

    public float timeToRotate = 1;
    public AnimationCurve scaleAndRotationMultiplierCurve;
    public AnimationCurve fillAmountCurve;
    public AnimationCurve rotationSpeedMultiplier;

    public float[] fillAmounts;
    public float[] rotationValues;

    private int rotationIndex = 0;

    [Range(0, 4)]
    public int numberOfColorsInBottle = 4;

    public Color topColor;
    public int numberOfTopColorLayers = 1;

    public BottleController bottleRef;
    private int numberOfColorsToTransfer = 0;

    public Transform leftRotationPoint;
    public Transform rightRotationPoint;
    private Transform chosenRotationPoint;

    private float directionMultiplier = 1.0f;

    Vector3 originalPosition;
    Vector3 startPosition;
```

```

    Vector3 endPosition;

    [SerializeField]
    LineRenderer lineRenderer;

    private BottleParent parent;

    void Start()
    {
        parent = FindObjectOfType<BottleParent>();

        bottleMaskSR.material.SetFloat("_FillAmount", fillAmounts[numberOfColorsInBottle]);

        originalPosition = transform.position;

        UpdateColorsOnShader();
        UpdateTopColorValues();
    }

    public override void Interact()
    {
        if (parent.clickable)
        {
            base.Interact();
            parent.OnBottleClick(this);
        }
    }

    public void StartColorTransfer()
    {
        ChooseRotationPointAndDirection();

        numberOfColorsToTransfer = Mathf.Min(numberOfTopColorLayers, 4 -
bottleRef.numberOfColorsInBottle);

        for (int i = 0; i < numberOfColorsToTransfer; i++)

```

```

{
    bottleRef.bottleColors[bottleRef.numberOfColorsInBottle + i] = topColor;
}

bottleRef.UpdateColorsOnShader();

CalculateRotationIndex(4 - bottleRef.numberOfColorsInBottle);

transform.GetComponent<SpriteRenderer>().sortingOrder += 2;
bottleMaskSR.sortingOrder += 2;

StartCoroutine(Movebottle());
}

```

```

IEnumerator Movebottle()
{
    startPosition = transform.position;
    if (chosenRotationPoint == leftRotationPoint)
    {
        endPosition = bottleRef.rightRotationPoint.position;
    }
    else
    {
        endPosition = bottleRef.leftRotationPoint.position;
    }

    float t = 0;

    while (t <= 1)
    {
        transform.position = Vector3.Lerp(startPosition, endPosition, t);
        t += Time.deltaTime * 2;

        yield return new WaitForEndOfFrame();
    }

    transform.position = endPosition;
}

```

```
StartCoroutine(RotateBottle());
```

```
}
```

```
IEnumerator MoveBottleBack()
```

```
{
```

```
    startPosition = transform.position;
```

```
    endPosition = originalPosition;
```

```
    float t = 0;
```

```
    while (t <= 1)
```

```
    {
```

```
        transform.position = Vector3.Lerp(startPosition, endPosition, t);
```

```
        t += Time.deltaTime * 2;
```

```
        yield return new WaitForEndOfFrame();
```

```
    }
```

```
    transform.position = endPosition;
```

```
    transform.GetComponent<SpriteRenderer>().sortingOrder -= 2;
```

```
    bottleMaskSR.sortingOrder -= 2;
```

```
    parent.clickable = true;
```

```
    parent.CheckWin();
```

```
}
```

```
void UpdateColorsOnShader()
```

```
{
```

```
    bottleMaskSR.material.SetColor("_Color01", bottleColors[0]);
```

```
    bottleMaskSR.material.SetColor("_Color02", bottleColors[1]);
```

```
    bottleMaskSR.material.SetColor("_Color03", bottleColors[2]);
```

```
    bottleMaskSR.material.SetColor("_Color04", bottleColors[3]);
```

```
}
```

```

IEnumerator RotateBottle()
{
    float t = 0;
    float lerpValue;
    float angleValue;

    float lastAngleValue = 0;

    while (t < timeToRotate)
    {
        lerpValue = t / timeToRotate;
        angleValue = Mathf.Lerp(0.0f, directionMultiplier * rotationValues[rotationIndex], lerpValue);

        //transform.eulerAngles = new Vector3(0, 0, angleValue);
        transform.RotateAround(chosenRotationPoint.position, Vector3.forward, lastAngleValue -
angleValue);

        bottleMaskSR.material.SetFloat("_ScaleAndRotationMultiplier",
            scaleAndRotationMultiplierCurve.Evaluate(angleValue));

        if (fillAmounts[numberOfColorsInBottle] > fillAmountCurve.Evaluate(angleValue) + 0.005f)
        {
            if (!lineRenderer.enabled)
            {
                float lineY = 5.814713f;

                if (bottleRef.numberOfColorsInBottle == 1) lineY = 5.114713f;
                if (bottleRef.numberOfColorsInBottle == 2) lineY = 4.234713f;
                if (bottleRef.numberOfColorsInBottle == 3) lineY = 3.394713f;

                lineRenderer.material = new Material(Shader.Find("Sprites/Default"));
                lineRenderer.startColor = topColor;
                lineRenderer.endColor = topColor;
                lineRenderer.SetPosition(0, chosenRotationPoint.position);
                lineRenderer.SetPosition(1, new Vector3(chosenRotationPoint.position.x,
                    chosenRotationPoint.position.y - lineY, chosenRotationPoint.position.z));
            }
        }
    }
}

```

```

        lineRenderer.enabled = true;

    }

    bottleMaskSR.material.SetFloat("_FillAmount", fillAmountCurve.Evaluate(angleValue));

    bottleRef.FillUp(fillAmountCurve.Evaluate(lastAngleValue) -
fillAmountCurve.Evaluate(angleValue));

    }

    t += Time.deltaTime * rotationSpeedMultiplier.Evaluate(angleValue);
    lastAngleValue = angleValue;
    yield return new WaitForEndOfFrame();
    }

    angleValue = directionMultiplier * rotationValues[rotationIndex];
    //transform.eulerAngles = new Vector3(0, 0, angleValue);
    bottleMaskSR.material.SetFloat("_ScaleAndRotationMultiplier",
    scaleAndRotationMultiplierCurve.Evaluate(angleValue));
    bottleMaskSR.material.SetFloat("_FillAmount", fillAmountCurve.Evaluate(angleValue));

    numberOfColorsInBottle -= numberOfColorsToTransfer;
    bottleRef.numberOfColorsInBottle += numberOfColorsToTransfer;
    Debug.Log("numberOfColorsToTransfer = " + numberOfColorsToTransfer);
    Debug.Log("numberOfColorsInBottle = " + numberOfColorsInBottle);

    lineRenderer.enabled = false;

    StartCoroutine(RotatePottleBack());
    }

    IEnumerator RotatePottleBack()
    {
        float t = 0;
        float lerpValue;
        float angleValue;

```

```

float lastAngleValue = directionMultiplier * rotationValues[rotationIndex];

while (t < timeToRotate)
{
    lerpValue = t / timeToRotate;
    angleValue = Mathf.Lerp(directionMultiplier * rotationValues[rotationIndex], 0.0f, lerpValue);

    //transform.eulerAngles = new Vector3(0, 0, angleValue);

    transform.RotateAround(chosenRotationPoint.position, Vector3.forward, lastAngleValue -
angleValue);

    bottleMaskSR.material.SetFloat("_ScaleAndRotationMultiplier",
        scaleAndRotationMultiplierCurve.Evaluate(angleValue));

    lastAngleValue = angleValue;

    t += Time.deltaTime;

    yield return new WaitForEndOfFrame();
}

UpdateTopColorValues();
bottleRef.UpdateTopColorValues();

angleValue = 0.0f;
transform.eulerAngles = new Vector3(0, 0, angleValue);
bottleMaskSR.material.SetFloat("_ScaleAndRotationMultiplier",
    scaleAndRotationMultiplierCurve.Evaluate(angleValue));

StartCoroutine(MoveBottleBack());
}

public void UpdateTopColorValues()
{

```

```
if (numberOfColorsInBottle != 0)
```

```
{
```

```
    numberOfTopColorLayers = 1;
```

```
    topColor = bottleColors[numberOfColorsInBottle - 1];
```

```
    if (numberOfColorsInBottle == 4)
```

```
    {
```

```
        if (bottleColors[3].Equals(bottleColors[2]))
```

```
        {
```

```
            numberOfTopColorLayers = 2;
```

```
            if (bottleColors[2].Equals(bottleColors[1]))
```

```
            {
```

```
                numberOfTopColorLayers = 3;
```

```
                if (bottleColors[1].Equals(bottleColors[0]))
```

```
                {
```

```
                    numberOfTopColorLayers = 4;
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

```
    else if (numberOfColorsInBottle == 3)
```

```
    {
```

```
        if (bottleColors[2].Equals(bottleColors[1]))
```

```
        {
```

```
            numberOfTopColorLayers = 2;
```

```
            if (bottleColors[1].Equals(bottleColors[0]))
```

```
            {
```

```
                numberOfTopColorLayers = 3;
```

```
            }
```

```
        }
```

```
    }
```

```
    else if (numberOfColorsInBottle == 2)
```

```
    {
```

```
        if (bottleColors[1].Equals(bottleColors[0]))
```

```
        {
```

```

        numberOfTopColorLayers = 2;
    }
}

    rotationIndex = 3 - (numberOfColorsInBottle - numberOfTopColorLayers);
}
else numberOfTopColorLayers = 0;
}

public bool FillBottleCheck()
{
    if (numberOfColorsInBottle == 4)
    {
        return false;
    }
    else
    {
        return true;
    }
}

private void CalculateRotationIndex(int numberOfEmptySpacesInSecondBottle)
{
    rotationIndex = 3 - (numberOfColorsInBottle -
        Mathf.Min(numberOfEmptySpacesInSecondBottle, numberOfTopColorLayers));
}

private void FillUp(float fillAmountToAdd)
{
    bottleMaskSR.material.SetFloat("_FillAmount", bottleMaskSR.material.GetFloat("_FillAmount") +
fillAmountToAdd);
}

private void ChooseRotationPointAndDirection()
{
    if (transform.position.x > bottleRef.transform.position.x)

```

```
{  
    chosenRotationPoint = leftRotationPoint;  
    directionMultiplier = -1.0f;  
}  
else  
{  
    chosenRotationPoint = rightRotationPoint;  
    directionMultiplier = 1.0f;  
}  
}
```