ВСТУП

Багато версій «Tetris» були розроблені для різних платформ різними розробниками з самого початку, але тонка настройка, крім основних правил у звичайному «Tetris» (наприклад, відчуття точної роботи, відчуття точної роботи тощо) закон обертання Tetrimino , новий режим тощо) було залишено на винахідливість кожного розробника. Тому, залежно від програмного забезпечення, відчуття роботи, детальні правила, упорядковані правила та нові функції часто були абсолютно різними.

Тому в 2002 році Хенк Роджерс, президент The Tetris Company, встановив рекомендації щодо уніфікації цих деталей.

Більша частина цього вмісту заснована на правилах «Світів тетрісу», розроблених самим Роджерсом і випущених на кожній платформі в 2001 році (2002 рік у Японії). Офіційна назва та детальний зміст цієї інструкції не розголошуються звичайним користувачам, але вони зазвичай називаються «Світове правило», «Світовий стандарт», «Рекомендація TETRIS 2002», «Рекомендація TETRIS 2005» тощо, залежно від гри. або виробник. Називається по-різному.

Що стосується рекомендацій, то було підтверджено, що існують перші «Рекомендації TETRIS 2002», створені в 2002 році, і «Рекомендації TETRIS 2005», переглянуті в 2005 році, і Міхара, виробник серії TGM. У блозі під керуванням Ічіро, є опис, що припускає існування «Рекомендацій 2008 р.» та «Рекомендацій 2010 р.», що свідчить про те, що видавцям не обов’язково потрібно впроваджувати їх усі. Опис також було підтверджено.

«Тетріс», який народився після створення цього керівництва, має такі загальні специфікації (деякі специфікації керівництва не прийняті / деякі ігри можна змінити, тому не завжди потрібно впроваджувати їх усі) які зарекомендували себе як

проблематичні при розробці програмних систем, наприклад C# не підтримує множинне успадкування класів (на відміну від C++).

Я вирішив розробити гру «Tetris» на ігровому двигуні Unity та за допомогою мови програмування C#.

Unity (Unity3D) — ігровий движок з вбудованою IDE, розроблений і проданий Unity Technologies (японська корпорація — Unity Technologies Japan Co., Ltd.). Контент можна розробляти в основному шляхом програмування за допомогою C#. Він підтримує не тільки ПК (Windows, macOS), але й мобільні (iOS, Android), веб-браузер (WebGL), домашні ігрові консолі (PlayStation 4, Xbox One, Nintendo Switch тощо) та кросплатформні, VR / AR / Він також підтримує розробку контенту для обладнання MP.

Інструмент для розробки ігор, випущений у 2005 році компанією Over the Edge Entertainment (OTEE), яка була створена в Данії в 2004 році, а в 2007 році змінила назву на Unity Technologies. Кількість щомісячних активних користувачів контенту, розробленого або використовуваного за допомогою Unity, перевищує 2,8 мільярда, і хоча він займає велику частку в основному в мобільних іграх, він використовується в різних галузях і галузях, крім домашніх ігрових консолей та ігор.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Визначення Тетрісу

Тетріс (англ. Tetris) - культова комп'ютерна гра, винайдена в СРСР Олексієм Пажитновим і представлена громадськості 6 червня 1984 року. Ідею «Тетріса» йому підказала куплена ним гра в пентаміно, а сама назва сформувалася додаванням двох слів – «тетраміно» і «теніс». »(Видавництво «Мир», 1975 рік). Пажитновим у червні 1984 року на комп'ютері Електроніка-60. Працюючи у ВЦ Академії наук СРСР, він займався проблемами штучного інтелекту та розпізнавання мови, а для тестування ідей застосовував головоломки, у тому числі й класичне пентаміно. Пажитнов намагався автоматизувати укладання пентаміно у задані фігурки. Ці досліди і призвели до зародження основної ідеї «Тетріса»: фігурки мають падати, а заповнені ряди – зникати.

У рекомендаціях: «Кожні 10 рядків стираються, рівень збільшується на 1, а швидкість падіння тетриміно збільшується», «150 рядків (= 15 рівнів) стираються, щоб завершити гру», і «Рівень на початку може бути обраним довільно [Примітка] Режим «марафон», який прогресує за правилами «11]», є основним режимом гри. Крім того, існують ігрові режими, які не мають поняття рівня, наприклад «Спринт / 40 ліній», який змагається за час, щоб очистити вказану кількість ліній на основі «Tetris DX», і «Ultra», який змагається для оцінки протягом зазначеного часу.

В інших іграх нижче наведено основні умови для підвищення рівня, що вказує на швидкість падіння тетриміно.

Метод, який часто зустрічається з моменту створення керівництва до теперішнього часу, «він збільшується на 1 рівень на кожні 10 стертих рядків». У деяких випадках, наприклад «New Sega Tetris» і «Tetris Effect», рівень змінюється не кожні 10 рядків.

Метод підвищення рівня, використаний у «Світі тетрісу». Рівень підвищується, коли ви отримуєте вказані «очки» (в деяких іграх позначається «лінією»). Крім лінії стирання, до балів додаються бонуси від T-Spin і Back to Back, а вищі очки можна отримати, вирівнявши кілька ліній одночасно (наприклад, 3 бали, якщо дві лінії вирівняні одночасно) Кількість точок не означає кількість ліній. В основному, гра очищається, коли досягнута квота до 15 рівня.

На додаток до "Old Sega Tetris", який стирає 4 рядки або підвищує рівень навіть на певний період часу, використовується унікальний метод залежно від гри, наприклад, поява 1 комплекту Tetrimino і серії TGM, яка підвищується на 1 для кожного рядка стирається і ділиться на кожні 100 рівнів. Можливо, це було зроблено.

1.2. Правила гри

Тетраміно - набір однозв'язних фігурок, що складаються з чотирьох квадратів (від грец. τετρα--чотири). Слово «однозв'язні» означає, що від одного квадрата до іншого можна дістатися ходом човна. Усього таких фігур сім, якщо вважати, що дві з них переходять одна в одну лише при відображенні, а не при обертанні. Вони є підмножиною поліміно.

У випадковому порядку фігури-тетраміно падають зверху у прямокутну склянку. Фігуру можна повертати або рухати по горизонталі. Фігурка летить, доки не натрапить на іншу фігуру або нижньої межі. Якщо рядок повністю заповнений, він зникає, і всі ряди, що знаходяться вище за нього, зсуваються на 1 клітинку вниз. Темп гри поступово збільшується. Гра закінчується, коли нова фігура не може поміститися.

Прискорити тетріміно

Якщо правила добре вивчені, гравець зможе продовжити гру напівпостійно.

Оскільки в реальній аркадній грі гра триває напівпостійно, то якщо гра триватиме тривалий час, швидкість падіння тетриміно буде поступово збільшуватися, а залежно від гри — сам час роботи (час гри) від приземлення тетріміно теж стане коротшим.

У результаті, якщо ви будете діяти, думаючи повільно, ви не зможете наздогнати падіння, тому вам потрібно буде зробити миттєве судження. Крім збільшення швидкості падіння тетриміно, зниження концентрації внаслідок тривалої гри збільшує кількість помилок судження та операцій, а тетріміно накопичується, що неминуче призводить до завершення гри. Однак, коли гра починається знову, це перша швидкість падіння. Здається, це мотивує гравця спробувати ще раз.

Одиниця швидкості Тетроміно

Зазвичай швидкість переміщення на ○ тетриміно в одному кадрі виражається як ○ G. Наприклад, якщо за 1 секунду намальовано 60 кадрів, 1 блок опускається за 1 секунду – це 1/60G, а 1 блок опускається за 0,5 секунди – це 1/30G.

Максимальна швидкість першого «Sega Tetris» (версії Sega System 16) становить 1G.

Однак у серії «Tetris The Grand Master (TGM)» ми запровадили прискорення 2G або більше, щоб вимагати подальшого прискорення. У першій половині гри він збільшиться до 5G як максимальна швидкість на етапі, де він може рухатися в повітрі, але з другої половини гри він раптово злетить до 20G (стан, коли Tetromino впаде до нижній частині поля в той же час, коли воно з’являється). Є певні сумніви щодо цього позначення, але, здається, зараз воно поширене.

1.3. Мова програмування

C # (Sea Sharp) — мова програмування, розроблена Андерсом Хейлсбергом. На синтаксис впливають мови C (C, C ++ тощо). Крім синтаксису, можна побачити вплив розробленого Borland Delphi, до якого раніше належав Хейлсберг.

Такі елементи, як багатопарадигмальна мова програмування, типізація, імперативний, декларативний, процедурний, функціональний, загальний та об'єктно-орієнтований, підкреслюються як гасла.

Окрім того, що була частиною фреймворку Microsoft ".NET Framework", включаючи периферійні технології, такі як Common Language Infrastructure (CLI), що була попередня Microsoft, яка намагалася перенести "несумісну Java" на Java за допомогою Visual J ++? На відміну від, багато специфікацій були активно розкриті та довірені механізму стандартизації, щоб дозволити вільне використання (ECMA-334, ISO / IEC 23270: 2003, JIS X 3015), і зміни у ставленні компанії змінилися. Це з’являється

У багатьох відносинах це мова, яка краще всього відображає функціональність базового інтерфейсу командної строки. Більшість вбудованих типів у C# відповідають типовим значенням, реалізованим у середовищі CLI. повинен підтримувати середу CLR або генерувати код у певному форматі, так як Common Intermediate Language (CIL). Слідомовно, теоретично можна генерувати залежні від середніх машинних мов, такі як C++ та FORTRAN.

C# має різні обмеження та покращення в порівнянні з C і C ++. Крім того, багато специфікацій спираються на саму .NET Framework, а не на мову C #. Багато обмежень і вдосконалень, запроваджених у Java, також були прийняті в C #, але деякі з нових удосконалень, запроваджених у C #, пізніше були прийняті і в Java. Нижче наведено приклад.

Синтаксичні та несинтаксичні покращення

- Булеві типи bool існують у C #, а умовним while операторам має бути надано вираз типу. У мові C немає логічного типу , і він використовується як тип (0 є хибним, а відмінний від нуля є істинним), і вказівник (оскільки було зручно вважати нульовий покажчик хибним). Однак специфікація була що це може бути речення або вираз, наданий реченням. Іноді це було зручно, але іноді не помічали помилок, наприклад той факт, що навіть якщо вираз присвоєння було помилково записано там, де має бути записано вираз порівняння, він був би сумісним з компіляцією. У C #, щоб запобігти помилкам, робить логічний тип незалежним замість такої специфікації, і є багато місць, де логічний тип є суворо обов'язковим. ifboolintwhileif
- switchstring Ви можете використовувати рядковий тип, а також цілочисельний тип або тип, подібний до цілого типу в інструкції . case Літерали рядків (рядкові константи) можна використовувати для міток, а також константи цілого чи цілого типу.
- Вбудований розмір і внутрішнє представлення вказані в специфікації та незалежні від платформи та реалізації. Числа з плаваючою комою відповідають стандарту IEEE 754 . Символи та рядки мають кодування UTF-16 [22] .
-

Керування вказівником і пам'яттю

- Підтримує покажчики. Вказівники можна unsafe використовувати лише в межах області, і лише програми з відповідними привілеями можуть unsafe виконувати код, позначений як. Більшість доступу до об’єктів здійснюється за допомогою керованих і захищених посилань, а більшість арифметичних операцій перевіряється на переповнення. unsafe Вказівники

можуть вказувати на типи значень або рядки. IntPtr Safecode дозволяє обмінюватися вказівниками за допомогою типів, хоча це не обов'язково.

- Немає явного способу звільнити керовану пам’ять, і пам’ять, на яку більше не посилаються, автоматично звільняється збирачем сміття . Збірник сміття усуває витік пам’яті, спричинений забуттям звільнення пам’яті. C # також надає спосіб явного керування некерованими ресурсами, такими як підключення до бази даних. Це робиться за допомогою IDisposable інтерфейсів і операторів. using

Простір імен і об'єктно-орієнтована система типів

- Немає глобальних змінних чи методів. Усі поля та методи мають бути оголошені як частина класу чи структури.

  Неможливо визначити вільні функції, які існують на рівні простору імен , наприклад функції C / C ++ printf(). У більшості випадків класи та структури належать до просторів імен , щоб уникнути колізій імен .

  - Простір імен має ієрархічну структуру. Тобто простори імен можуть бути оголошені в інших просторах імен.
  - Усі типи, включаючи вбудовані типи значень, є похідними від object класу (System.Object). Іншими словами object, він успадковує всі властивості та методи класу. Наприклад, кожен тип має ToString() метод.
  - Класи (class) є типами посилань, а структури (struct) і перерахуваннями (enum) є типами значень. Структури легші за класи і взаємодіють із C / C ++, але вони не можуть визначати похідні типи.
  - Класи та структури можуть реалізовувати декілька інтерфейсів , але множинне успадкування не підтримується.
  - C # є типобезпечним у порівнянні з C ++. Неявні перетворення за замовчуванням обмежуються безпечними перетвореннями, такими як перетворення, які розширюють діапазон цілих чисел, і перетворення від похідних класів до базових. Це застосовується під час компіляції, JIT-

компіляції та в деяких динамічних випадках під час виконання. Неявне перетворення неможливе між булевим типом і цілочисельним типом, а також типом перерахування і цілочисельним типом. Коли ви визначаєте неявне перетворення, ви повинні явно вказати його таким чином. Це відмінна специфікація від конструктора C ++.

- Члени переліку поміщаються в область дії переліку. Ви також можете отримати ім'я константи типу перерахування. Крім того, значення констант можна динамічно отримувати з перерахованих імен констант.
- Синтаксис властивостей можна використовувати для спрощення визначення та використання засобів доступу . Інкапсуляція в C++ і Java зазвичай визначає та використовує функції-члени або методи, які є методами доступу для отримання/налаштування, але в C# функція властивості робить її такою ж інтуїтивною, як читання та запис інкапсуляцію., зберігаючиполів Властивості можна використовувати для контролю доступу до учасників та перевірки достовірності даних. Синтаксис події ( ) надається для інкапсуляції делегата, який використовується для обробника подій .

.1.4. Висновок

У цьомі розділі був опис предметної області. Визначення гри Тетріс та опис правил гри.

РОЗДІЛ 2

ПРОЕКТУВАННЯ ТЕТРІСУ

2.1. Постановка задачі

Розробити додаток «Тетріс», призначений для генерації ігрового поля (склянки) і заповнення його фігурами-тетраміно.

Тетріс повинен:

1) Створювати ігрове поле.

2) Випадковим чином генерувати фігури. осі та горизонтальних зрушень.

3) Дозволяти користувачеві змінювати положення фігур у просторі шляхом обертання навколо своєї осі та горизонтальних зрушень.

4)Слідкувати за заповненням рядів і очищати ті, які повністю заповнені клітинами фігур.

5) Надавати зручний та інтуїтивно зрозумілий інтерфейс для користувача відповідно до вимог ергономічності.

2.2. Блок-схема

Запустивши програму оголошується масив, з наступним заповненням порожніми значеннями. Потім генератор чисел вибирає фігуру. Далі включається цикл падіння фігури. Тіло циклу є зсув фігури на один рядок нижче і перевірка на натискання кнопок "Вправо", "Вліво", "Поворот" (при натисканні цих кнопок відбувається зсув фігури вправо, вліво і поворот навколо своєї осі відповідно). Умова закінчення циклу – відсутність вільного місця під фігурою. Після виходу із циклу починається перевірка повністю заповнених рядків. Якщо такі є, то відбувається їх очищення та зсув усіх рядків, що знаходяться вище. Далі відбувається новий вибір фігури та повторення алгоритму. Якщо ж заповнених рядків немає, відбувається перевірка на вільне місце у третьому рядку. Якщо третій рядок вільний, то відбувається вибір фігури та виконання алгоритму циклу. При

заповненні третього рядка відбувається зупинка таймера та виведення результатів гри. Потім відбувається вихід із гри. (див рис. 2.1).

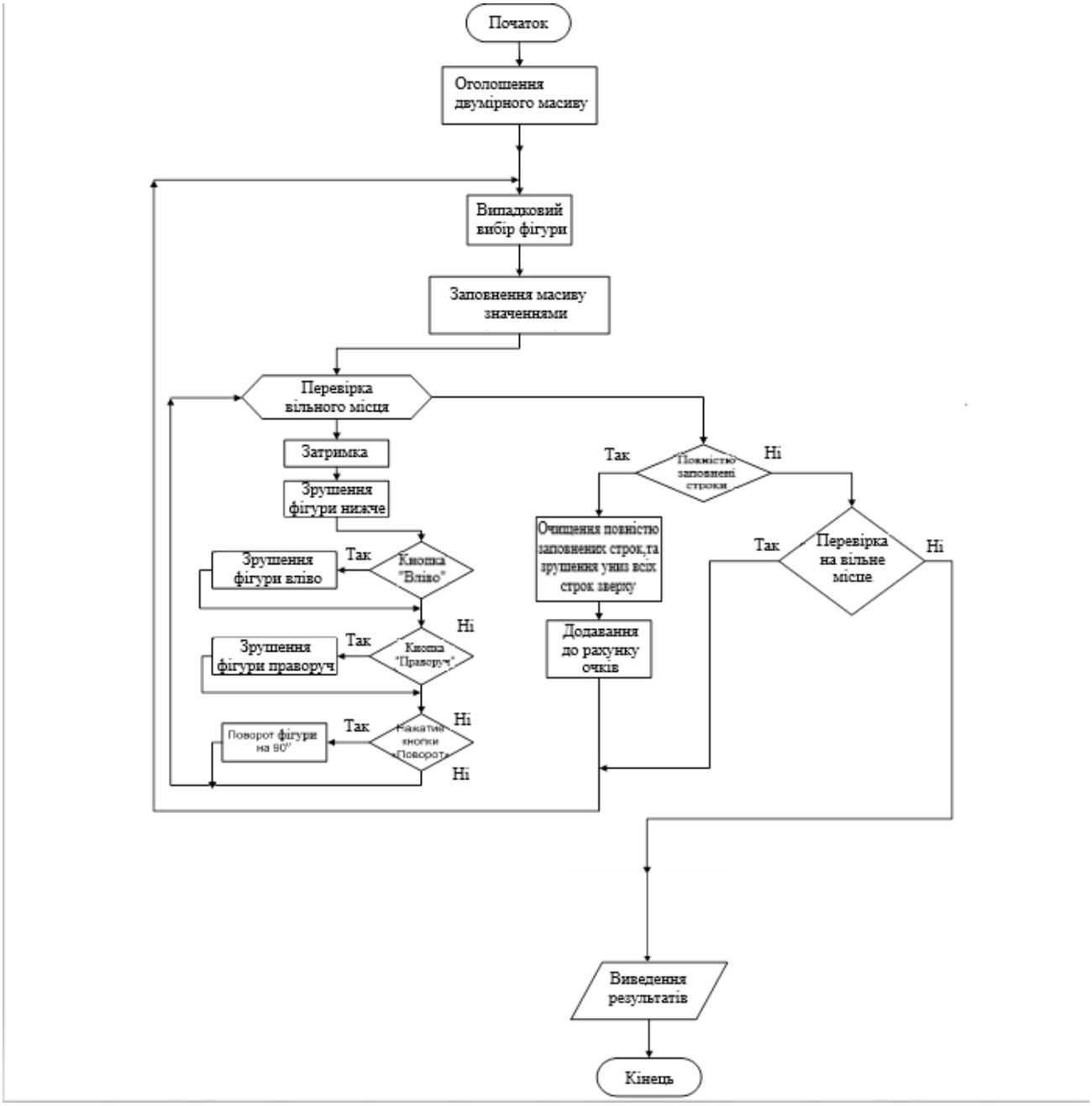


Рис. 2.1 Блок-схема

2. 3 Заготівля блоку

Щоб створити фігури, мені потребувалося зробити заготівку блоку, щоб кожен раз не змінювати його характеристики(розмір, тег та інші). (Рис.2.2)

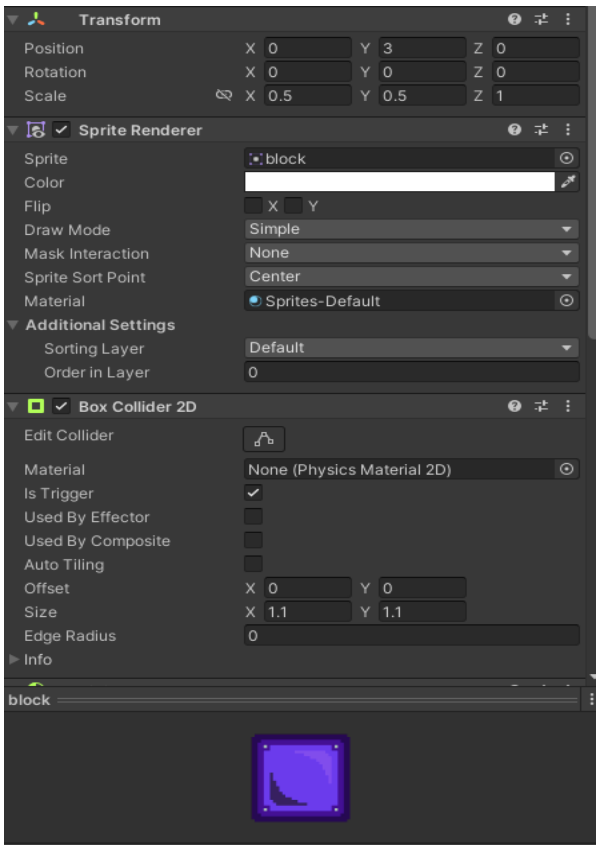


Рис 2.2. Блок

Гаразд, настав час створити групи. Точніше, Prefab для кожного.
Я створив порожній GameObject, вибравши GameObject -> Create Empty у верхньому меню. Це додає порожній GameObject до нашої ієрархії.
Потім перетягнув зображення блоку в порожній GameObject 4 рази, тому 4 блоки є його дочірніми: (Рис.2.3)

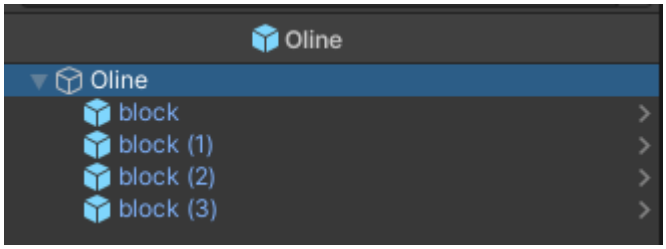


Рис.2.3 Структура фігури

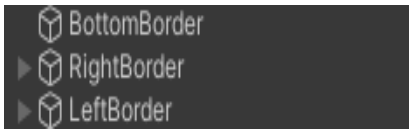
Тепер, щоб розташувати блоки так, щоб вони стали групою O: (Рис.2.4)

Рис.2.4 Приклад фігури

І так з кожною фігурою.

2. 4 Додавання меж

Після створення фігур, я додав межі, за які, фігура не повинна виходити.
Щоб це реалізувати мені знадобилось три різних блоків(один для нижньої межі, а інші для бокових). Я створив 3 пустих GameObject та наповнив їх потрібними блоками, як це було із фігурами. (Рис.2.5-2.7)



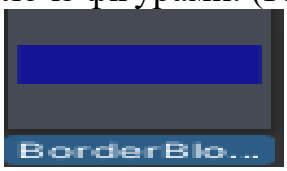


Рис.2.5 Блоки для меж

Рис.2.6 Зображення блоків меж

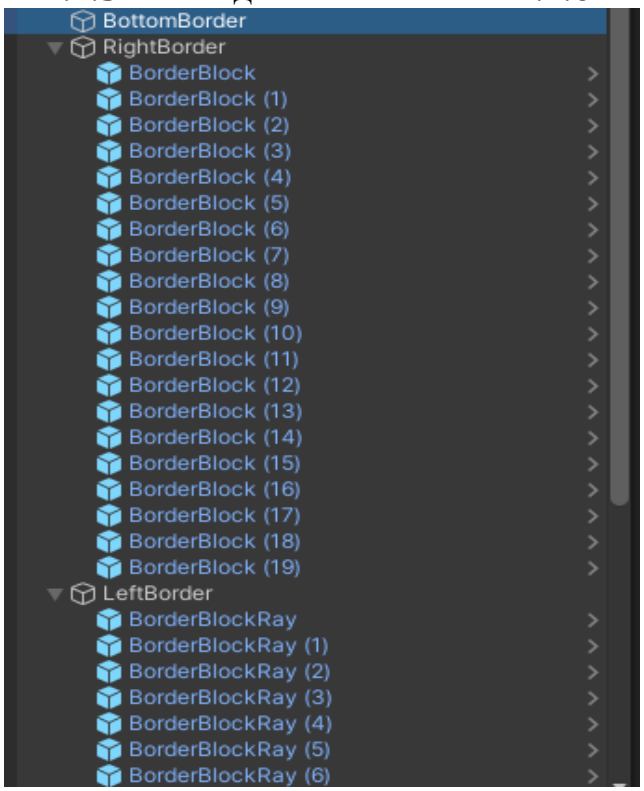


Рис.2.7 Склад блоків меж

2.5 Спавнер фігур

Далі я зробив ще один порожній GameObject, та назвав його SpawnPoint і розмістив у верхній частині сцени: (Рис.2.8)

Рис.2.8 Точка з’яення фінур

Саме з цієї точки будуть випадковим чином будуть з’являтися фігури.

Spawner забезпечить функцію SpawnNextShape, яка створює випадкову групу, коли це необхідно. (Рис.2.9)

```
public void SpawnNextShape()
{
    int index = Random.Range(0, shapePrefabs.Length);

    GameObject shape = Instantiate(shapePrefabs[index], spawnPoint.position, Quaternion.identity);

    index = Random.Range(0, 4);
    shape.transform.eulerAngles = new Vector3(0, 0, 90 * index);
}
```

Рис.2.9 Реалізація спавнера

2.6 Падіння фігур

Після того, як вже готові фігури та межі, я приступив до написання коду, для падіння фігур: (Рис.2.10)

```
IEnumerator MoveDown ()
{
    while(true)
    {
        time = 0;
        transform.Translate(0, -0.5f, 0, Space.World);
        yield return new WaitForSeconds(1f / speedIndex);
    }
}
```

Рис.2.10 Реалізація падіння фігур

2.7 Виявлення зіткнення

Є кілька речей, які було справді нелегко зробити з моїми нинішніми навичками та знаннями як мови, так і програми, але одна була справді понад усе: виявлення зіткнень.

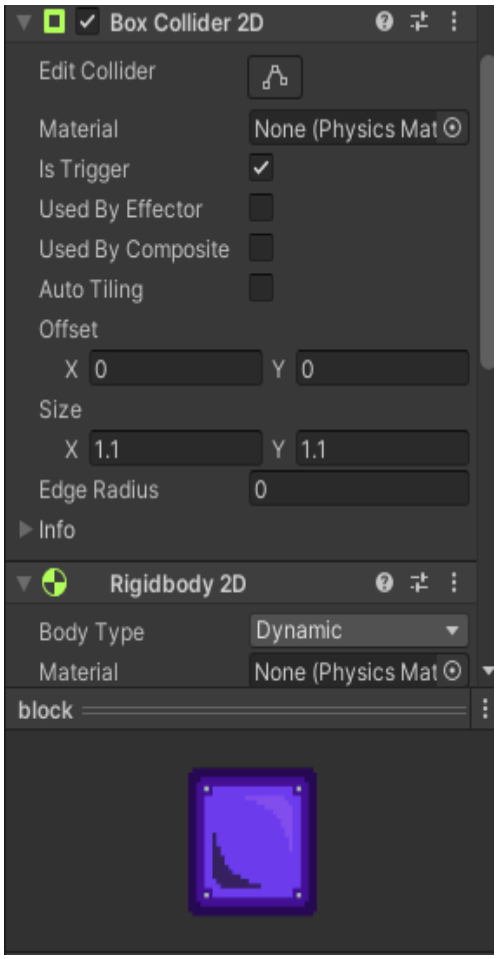
В основному я хотів:

- не допускати перетину цеглини лівого і правого країв каркаса
- запобігти перетин цегли з іншою
- зупинити цеглу при ударі об підлогу або іншу цеглу

На перший погляд це здається легким, особливо якщо ви знаєте, що двигун Unity реалізує систему виявлення зіткнень. Простіше кажучи, модуль виявлення зіткнень від Unity постійно перевіряє, чи щось торкається вашого поточного предмета. Якщо є, він повертає елемент, який торкається його, інакше повертає false.

І тут у мене виникла проблема: як відрізнити зіткнення знизу від бічних? Бо це не одне й те саме. Якщо виявити зіткнення знизу, треба змусити цеглину зупинитися, а якщо виявити зіткнення збоку, ви повинні перешкодити користувачеві перемістити цеглу з цього боку.

.Для того щоб індефікувати зіткнення блоків з межами та іншими блоками, я додав колайдери: (Рис. 2.11-2.13)



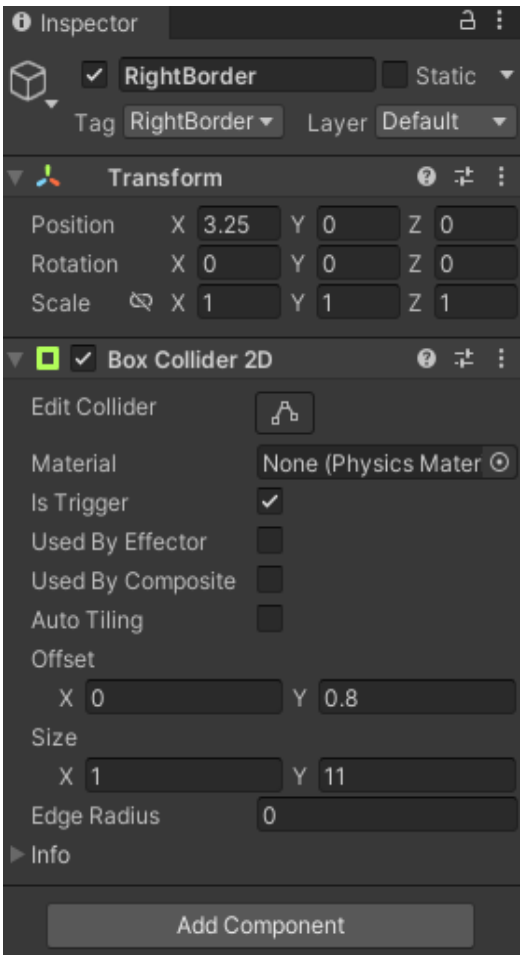


Рис. 2.11Тригер для блоку Рис. 2.12 Тригер для правої межі

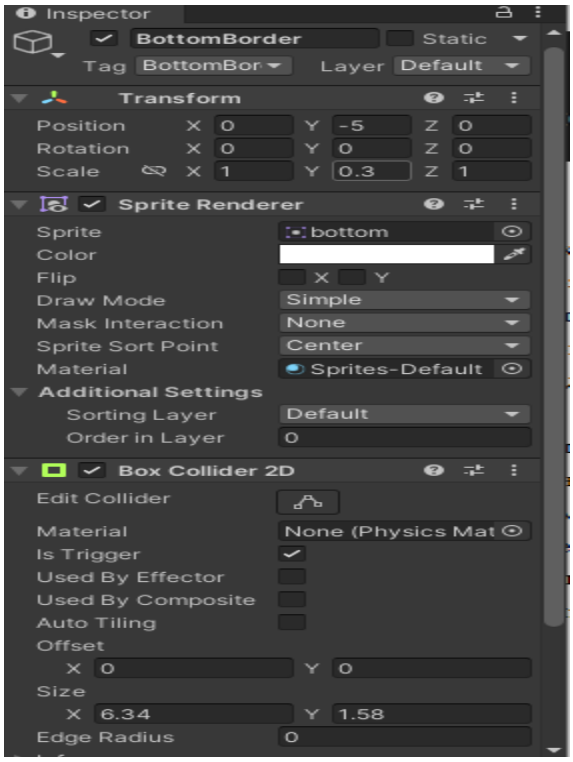


Рис.2.13 Тригер для лівої межі

Та написав таку функцію: (Рис.2.14)

```
internal void BorderCollided(string tag)
{
    switch(tag)
    {
        case "LeftBorder":
            canLeft = false;
            break;

        case "RightBorder":
            canRight = false;
            break;
        case "BottomBorder":
        case "Block":
            StopMovement();
            break;
        default:
                canLeft = true;
                canRight = true;
                break;
    }
}
```

Рис. 2.14 Реалізація тригеру

2.8 Обертання цегли

Коли я почав кодувати, я очікував, що обертання цегли буде дуже простим. У певному сенсі я не помилився: змусити цеглину обертатися сам по собі – найпростіша частина. При натисканні верхньої стрілки цегла обертається на 90° за годинниковою стрілкою або проти годинникової стрілки, без проблем. Реалізував це я таким способом: (Рис.2.15)

```
void RotateShape()
{
    if (Input.GetKeyDown(KeyCode.UpArrow))
    {
        transform.eulerAngles = transform.eulerAngles + new Vector3(0, 0, -90);
    }
}
```

Рис. 2.15 Реалізація обертання

Але набагато важче було запобігти обертанню, коли надто близько до стіни. Обертання червоної смуги біля стіни може означати вклинювання планки в стіну. Мені вдалося запобігти обертанню, коли він занадто близько до стіни, проте все ще є помилка з фіолетовою L-цеглою, яка не може обертатися, якщо розміщена на відстані 1 одиниці від стіни. На жаль, я не знаю, як це виправити.
Іншою великою проблемою, яку я мав і все ще маю з обертанням цегли, є обертання біля іншої цегли. На відміну від стін, немає обмежень для обертання цегли біля іншого, тобто таким способом можна перекривати цеглу. У більшості випадків ця помилка не буде помічена, оскільки звичайна поведінка під час гри в тетріс — обертання та переміщення цеглини під час падіння, а не переміщення цеглини в самий останній момент її падіння. Проте є помилка, яку я не зміг виправити.

2.9 Знищення заповнених рядків

На цьому етапі в мене були труднощі. Я довго не міг знайти спосіб, як виконати процес знищення рядів. Але з часом я вирішив застосувати промені.(Рис. 15) У кожному рядку є промінь, який розпізнає наявність блоків. Я порахував скільки блоків у ряду, та потім добавив ще одну функцію, яка буде перевіряти заповнений ряд, чи ні. (Рис. 16)

```
public void RunRays()
{
    for( int i = 0; i < lines.Length; i++)
    {
        lines[i].RunRay();
    }

    CheckLines();
}
```

```
void CheckLines()
{
    for(int i = 0; i < lines.Length; i++)
    {
        if(lines[i].blocks.Count == 11)
        {
            // print("ряд полный");
            lines[i].RemoveLine();


            DropDown(i);
            break;
        }
    }
}
```

Рис. 2.16 Створення проміння

Рис. 2.17 Перевірка наповненості рядку

Рядки зникають, усе працює, але потрібно ще, щоб блоки які були зверху цього ряду, падали вниз. Реалізував це я, таким чином: (Рис. 17)

```
void DropDown(int startIndex)
{
    for(int i = startIndex + 1; i < lines.Length; i++)
    {
        BlockRay line = lines[i];
        for (int k = 0; k < line.blocks.Count; k++)
        {
            line.blocks[k].gameObject.transform.position += offset;
        }
    }

    RunRays();

}
```

Рис. 2.18 Падіння блоків над зникшим рядком

2.10 Тестування програми

Тестування програми - це етап, у якому перевіряється, як веде себе програма якомога більшої кількості вхідних наборів даних, зокрема і заздалегідь неправильних.

Основні засади організації тестування:

1) слід якомога уникати тестування програми її автором, т.к. крім вже зазначеної об'єктивної складності тестування для програмістів тут є і той фактор, що виявлення недоліків у своїй діяльності суперечить людській психології (проте налагодження програми найефективніше виконується саме автором програми);

2) з тих самих міркувань організація - розробник програмного забезпечення має “одноосібно ” його тестувати;

3) під час аналізу результатів кожного тесту необхідно перевіряти, чи робить програма те, що вона має робити;

4) слід враховувати так званий "принцип накопичення помилок": ймовірність наявності не виявлених помилок у деякій частині програми прямо пропорційна числу помилок, вже виявлених у цій частині;

5) слід завжди пам'ятати, що тестування - творчий процес, а не ставитись до нього нехтуючи;

.

2.12 Висновок

У цьому розділі були потреби до програми, інструменти реалізації та їх детальний опис.

РОЗДІЛ 3

ОПИС ОСНОВНИХ ФУНКЦІЙ СИСТЕМИ

3.1. Інтерфейс

Рис 3.1

3.2 Управління

Клавіша	Дія
→	Рух праворуч
←	Рух ліворуч
↑	Обертання фігури
↓	Прискорення фігури

Таб. 3.1

3.3.Основні функції

Специфікації для роботи тетриміно відразу після заземлення та блокування тетриміно

У реалізації керівництва операція може виконуватися як є навіть після того, як тетріміно приземлиться, і фіксується, якщо жодна операція не виконується протягом 0,5 секунди. Крім того, якщо ви виконаєте будь-яку операцію (поворот або переміщення хоча б на одну клітинку) над фігурою протягом цього часу «гри», час «гри» буде скинуто, і фігура продовжуватиме обертатися або рухатися вбік одразу після приземлення. як воно є, це не буде виправлено назавжди. Операція обертання також відноситься до O-tetrimino, який не змінює форму.

Ця специфікація була встановлена як «Блокування розміщення нескінченності» і вперше була реалізована у «Світі тетрісу», але в тій самій роботі кнопку не натискали повторно, а продовжували натискати. Вона просто обертається утримуючи його, так що ви можете практично зупинити гру, просто утримуючи кнопку. Це було великим хітом серед гравців і критиків, тому пізніші ігри більше не використовують «специфікацію, яка обертається просто утримуванням кнопки».

Оскільки хід гри можна зупинити напівназавжди, це буде значно заважати ігровому процесу, тому поточні рекомендації обмежують те, що якщо ви переміщаєте або обертаєте загалом 15 разів за тетріміно, це буде негайно виправлено. «Розширене блокування розміщення» прийнято як базову специфікацію. Крім того, у «Sega Ages 2500 Series Vol.28 Tetris Collection» є такі, як «Tetris: New Century», які дозволяють довільно встановлювати, чи обмежувати кількість обертів і час відтворення Tetrimino.

Крім того, реалізація, яка була масовою до керівництва: «Час відтворення після приземлення скидається лише тоді, коли тетріміно природним чином випадає з кроку тощо», у настанові визначається як «Класичне блокування розміщення».

Є також такі роботи, як «Tetris The Absolute The Grand Master 2», час відтворення яких скорочується за рахунок підвищення рівня.

Вільний час після закріплення тетриміно

Більшість звичайних тетріс мають інтервал близько 0,5 секунди навіть після фіксації.

У ці дні інтервали змінюються залежно від реалізації, а в деяких випадках інтервал поступово скорочується за рахунок підвищення рівня, або наступний фрагмент може відображатися в полі в момент, коли тетріміно фіксується незалежно від рівня.

Швидкість переміщення Tetromino, коли ви утримуєте бічні клавіші зі стрілками

Якщо ви утримуєте бічну частину клавіші зі стрілкою, тетріміно буде рухатися через регулярні проміжки часу, але швидкість все ще різна залежно від гри, і вона незначно коливається відповідно до швидкості падіння тетриміно, а рух при тривалому натисканні реалізується. Можливо, це не було зроблено.

Інструкція передбачає, що тетріміно рухається з інтервалом 0,3 секунди на початку натискання, а потім зі швидкістю досягнення протилежної сторони від кінця за 0,5 секунди.

Виправлено зміщення порядку появи тетриміно

У ранньому «Тетрісі» алгоритм визначення наступної появи Tetrimino був повністю випадковим, який визначався монотонним випадковим числом, або існувала специфікація, заснована на шаблоні живлення, тому якщо вам не пощастило, випадкове число У деяких випадках 3 або 4 тетриміно одного типу падали підряд, і навіть якщо було оброблено 15 або більше паличок (I), необхідних для тетрісу, вони не впали (тоді утримання не було введено і палички). Я навіть не витримав).

Тому в серії «Tetris The Grand Master», яка діяла до створення керівних принципів, вперше була внесена поправка, в якій випадкова лотерея Тетріміно, що з'явилася, була повторно розіграна за певних умов.

Інструкція також вирішила ввести корекцію тетриміно, але специфікації відрізняються від методу лотереї, і повторюється перегрупування «7 типів тетриміно, які з'являються у випадковому порядку», прийнято «закон одного раунду».

ВИСНОВОК

Завданням дипломної роботи було розробити програму Тетріс

У ході виконання дипломної роботи розроблена програма Тетріс на мові C#, знайдено багато переваг та недоліків середовища Unity, а також розглянута мова програмування C#.

Програма складається з:

- Графічний інтерфейс користувача
- Функції взаємодії з даними;
- Працюючий функціонал програми;

Граючи у Тетріс можна скоротити час та розвивати логічне мислення. Програма не буде займати багато місця, а також її системні вимоги підійдуть до будь-якого комп’ютера