**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**„Київський політехнічний інститут імені Ігоря Сікорського”**

**Факультет прикладної математики**

“ЗАТВЕРДЖЕНО”

Керівник роботи

\_\_\_\_\_\_\_\_\_\_ Шкурат О.С.

“\_\_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2023 р.

**"Гра «Tetris» на основі Unity"**

**Пояснювальна записка**

|  |  |
| --- | --- |
| \_\_” \_\_\_\_\_\_\_\_\_\_\_\_ 2011 р. | Виконавець:  Науменко А.С. |

2023

**Зміст**

[**Список термінів, скорочень та позначень**](#_gjdgxs) **3**

[**Вступ**](#_1fob9te) **4**

[**Аналіз технологій розроблення**](#_3znysh7) 9

[**Розроблення браузерної гри**](#_yxd125fiz7ze) **10**

[2.1 Аналіз вимог до функціональності](#_ge9uywrhb4sw) 10

[2.2 Архітектура та вміст](#_26in1rg) 11

[**Аналіз розробленої гри**](#_lnxbz9) **15**

[**Висновки**](#_bhrpasj79fw1) **19**

[Список використаних літературних джерел](#_2jxsxqh) **20**

# **Список термінів, скорочень та позначень**

**Unity** — багатоплатформовий інструмент для розроблення відеоігор і застосунків, і рушій, на якому вони працюють. Створені за допомогою Unity програми працюють на настільних комп'ютерних системах, мобільних пристроях та гральних консолях у дво- та тривимірній графіці, та на пристроях віртуальної чи доповненої реальності. Застосунки, створені за допомогою Unity, підтримують DirectX та OpenGL.

C# (вимовляється Сі-шарп) — об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research (належить Microsoft).

Синтаксис C# близький до С++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов С++, Object Pascal, Модула і Smalltalk — С#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова С#, на відміну від C++, не передбачає множинне успадкування класів.

# **Вступ**

Гра «Tetris» на основі Unity являє собою гру яка була надихнута серією ігор «Tetris». Гравцям надано ігровий набір функцій, завдяки якому вони взаємодіють з тетрамінами.

# **Аналіз технологій розроблення**

## **1. Розбір технологій, використаних для створення гри**

## Коли ми говоримо про кросплатформену 2D груд, то єдина середа розробки,яка задовільнить розробника – це Unity, де у неї немає суперників.

## Вся логіка гри базується на скриптах, для написання скриптів була обрана мова програмування C#.

### 

### **1.1 Unity**

Unity - це крос-платформовий інтегрований ігровий двигун та середовище розробки, розроблене компанією Unity Technologies. Воно використовується для створення ігор, симуляцій, віртуальної та розширеної реальності, а також інтерактивних додатків. Ось деякі плюси Unity для розробки:

**Крос-платформовість:**

Unity дозволяє розробникам створювати ігри та додатки, які можуть працювати на різних платформах, таких як Windows, macOS, Linux, iOS, Android, WebGL, Xbox, PlayStation, VR- та AR-платформи.

**Інтеграція з Різними Технологіями:**

Unity легко інтегрується з різними технологіями, такими як комп'ютерна графіка, фізика, звук, штучний інтелект, комп'ютерний вид та інші, що полегшує створення високоякісних ігор та додатків.

**Велика Спільнота Розробників:**

Unity має велику та активну спільноту розробників, що сприяє обміну досвідом, навчанню та підтримці.

**Інтегроване Середовище Розробки:**

Unity постачається з інтегрованим середовищем розробки (IDE), яке надає зручний інтерфейс для створення, редагування та налагодження коду, а також розміщення об'єктів у сцені.

**Мова Програмування C#:**

Unity використовує мову програмування C#, яка є популярною, легкою для вивчення та дозволяє розробникам створювати потужні ігри та додатки.

**Asset Store:**

Unity має Asset Store, де розробники можуть придбати або продавати різноманітні ресурси, бібліотеки, плагіни та готові активи для прискорення процесу розробки.

**Широкі Можливості Графіки:**

Unity надає розробникам доступ до потужного двигуна графіки, що дозволяє створювати вражаючі графічні ефекти та реалістичні сцени.

**Підтримка VR та AR:**

Unity надає вбудовану підтримку для віртуальної та розширеної реальності, що робить його ідеальним інструментом для розробки VR- та AR-додатків.

Ці переваги роблять Unity популярним вибором для багатьох розробників у галузі ігрової та інтерактивної розробки.

### **1.2 С#**

C# (вимовляється як "C-sharp") - це об'єктно-орієнтована мова програмування, розроблена компанією Microsoft. Ось деякі плюси мови програмування C#:

**Синтаксис:**

C# володіє чистим та зрозумілим синтаксисом, що дозволяє розробникам швидко вивчати та розуміти мову.

**Об'єктно-орієнтована:**

C# є об'єктно-орієнтованою мовою програмування, що дозволяє розробникам легко організовувати свій код за допомогою об'єктів та класів.

**Широке Застосування:**

C# використовується в основному для розробки програм на платформі Microsoft, таких як Windows-додатки, веб-додатки з використанням ASP.NET, ігри в середовищі Unity, додатки для платформи Xamarin для мобільних пристроїв, та інші.

**Інтеграція з .NET Framework:**

C# взаємодіє з .NET Framework, що забезпечує широкий набір бібліотек та функцій для розробників. Це полегшує розробку різноманітних застосувань.

**Безпека та Контроль Типів:**

C# підтримує систему контролю типів, що дозволяє ранню виявлення помилок та підвищує безпеку коду.

**Інтегрована Розробка в Microsoft Visual Studio:**

Розробка на C# зазвичай ведеться в інтегрованому середовищі розробки (IDE) Microsoft Visual Studio, яке надає розробникам широкий набір інструментів для підтримки коду, налагодження та відлагодження.

**Підтримка Асинхронного Програмування:**

C# надає зручні засоби для асинхронного програмування, що особливо важливо для роботи з інтерфейсами користувача, мережами та іншими завданнями, які можуть викликати затримки.

**Велика Спільнота та Підтримка:**

У C# існує велика та активна спільнота розробників, що сприяє обміну знань, публікації різноманітних бібліотек та інструментів.

Ці переваги роблять C# популярним вибором для розробників у багатьох областях програмування.

### **1.4 Висновки**

Отже, як бачимо з переліку вище, обрані технології дозволять реалізувати вимоги до розроблюваної гри, а саме інтерфейс, ігрову логіку, графіку.

# **Розроблення гри**

## **2.1 Аналіз вимог до функціональності**

Аналіз вимог - частина процесу розробки програмного забезпечення, що включає в себе збір вимог до програмного забезпечення (ПО), їх систематизацію, виявлення взаємозв'язків, а також документування.

Повнота і якість аналізу вимог грають ключову роль в успіху всього проекту. Вимоги до ПО повинні бути документовані, здійсненні, тестовані, з рівнем деталізації, достатнім для проектування системи. Вимоги можуть бути функціональними і не функціональними.

Аналіз вимог включає три типи діяльності:

1. Збір вимог - спілкування з клієнтами і користувачами, щоб визначити, які їхні вимоги; аналіз предметної області.
2. Аналіз вимог - визначення, чи є зібрані вимоги неясними, неповними, неоднозначними або такими, що суперечать одні з одним; рішення цих проблем; виявлення взаємозв'язку вимог.
3. Документування вимог - вимоги можуть бути задокументовані в різних формах, таких як простий опис, сценарії використання, призначені для користувача історії, або специфікації процесів.

З урахуванням особливостей жанру гри, та своїх ідей щодо нових механік та можливостей, було визначено наступні функціональні вимоги до гри:

* Гра повинна мати можливість взаємодії з гравцем, який може керувати тетромінами (геометричними фігурами) та їхнім обертанням та переміщенням.
* Коли гравець програє, тетраміно зтираються зі всього поля, і гравець починає гру знову.
* Гравці можуть взаємодіяти з тетраміно натискаючи на наступні комбінації на клавіатурі:
  + KeyQ – rotate left
  + KeyE – rotate right
  + KeyS – Soft drop
  + KeyA– left movement
  + KeyD – right movement
  + Space – HardDrop

## **Архітектура та вміст**

Вся гра відбувається на одному екрані.

На екрані гри зображена ігрова поверхня.

Після завершення гри всі тетраміно зтираються та з’являється повідомлення про завершення гри

**2.3 Дизайн гри**

Рис. 1. Головний екран гри

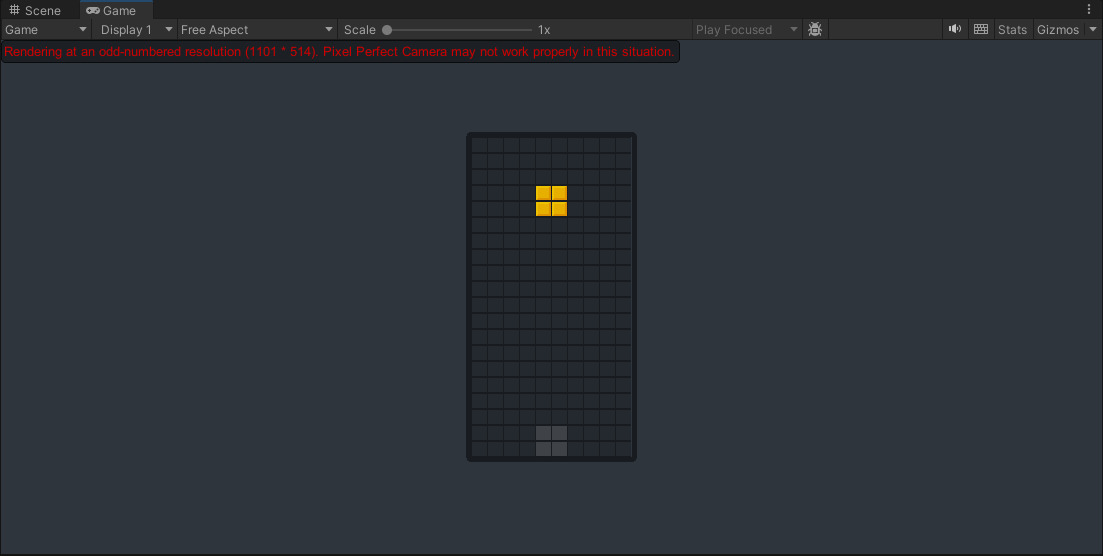


Рис. 2. Поле заповнена тетарміно

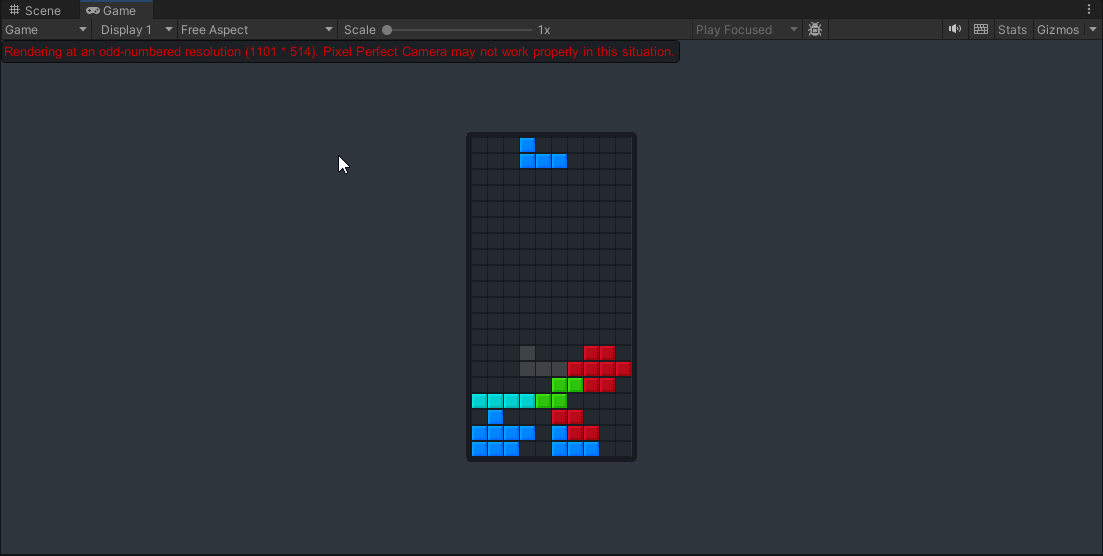
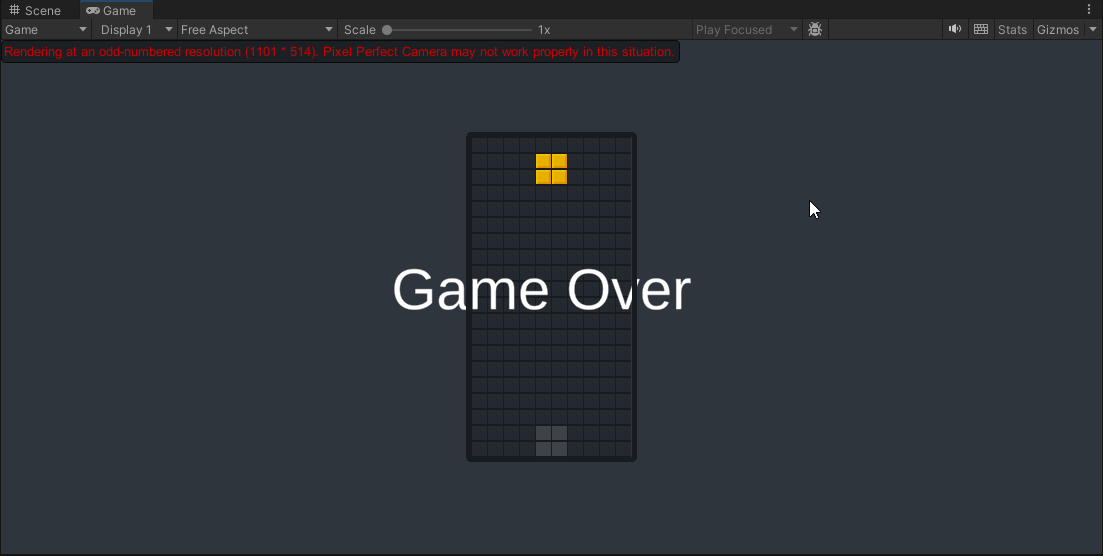


Рис. 3. Завершення гри



# **Аналіз розробленої гри**

**3.1 Тестування гри**

Тестування гри проводились на всіх етапах розробки, як розробником гри, так і сторонніми зацікавленими особами, що виявили бажання тестувати гру.

При тестуванні Були знайдені та виправлені деякі помилки, такі як:

* Орфографічні помилки;
* Баги з відбиванням від стін
* Затримка очищення ліній
* Баг з неочищенням ліній, через неправильну логіку очищення

**3.2 Опис використаних алгоритмів**

Для додавання тетарміно була створена така функція

|  |
| --- |
| public void SpawnPiece()  {  int random = Random.Range(0, tetrominoes.Length);  TetrominoData data = tetrominoes[random];  activePiece.Initialize(this, spawnPosition, data);  if (IsValidPosition(activePiece, spawnPosition)) {  Set(activePiece);  } else {  GameOver();  Invoke(nameof(DisableText), 3);  }  } |

Реалізація повернення тетарміно

|  |
| --- |
| private void ApplyRotationMatrix(int direction)  {  float[] matrix = Data.RotationMatrix;   // Rotate all of the cells using the rotation matrix  for (int i = 0; i < cells.Length; i++)  {  Vector3 cell = cells[i];   int x, y;   switch (data.tetromino)  {  case Tetromino.I:  case Tetromino.O:  // "I" and "O" are rotated from an offset center point  cell.x -= 0.5f;  cell.y -= 0.5f;  x = Mathf.CeilToInt((cell.x \* matrix[0] \* direction) + (cell.y \* matrix[1] \* direction));  y = Mathf.CeilToInt((cell.x \* matrix[2] \* direction) + (cell.y \* matrix[3] \* direction));  break;   default:  x = Mathf.RoundToInt((cell.x \* matrix[0] \* direction) + (cell.y \* matrix[1] \* direction));  y = Mathf.RoundToInt((cell.x \* matrix[2] \* direction) + (cell.y \* matrix[3] \* direction));  break;  }   cells[i] = new Vector3Int(x, y, 0);  }  }  attacker.blockCriticalHit = true;  setTimeout(() => {  attacker.blockCriticalHit = false;  }, 10000);  } else {  playSound('hit');  showIconHit(defender, 'hit');  damage = getDamage(attacker.fighter, defender.fighter);  }  defender.health -= damage;  setFighterHealthBar(defender);  if (defender.health <= 0) resolve(attacker.fighter);  }  });  }  export function getDamage(attacker, defender) {  const hitPower = getHitPower(attacker);  const blockPower = getBlockPower(defender);  let damage = hitPower - blockPower;  damage = damage > 0 ? damage : 0;  return damage;  }  export function getCriticalDamage(attacker) {  const damage = attacker.attack \* 2;  return damage;  }  export function getHitPower(fighter) {  const criticalHitChance = Math.random() + 1;  const power = fighter.attack \* criticalHitChance;  return power;  }  export function getBlockPower(fighter) {  const dodgeChance = Math.random() + 1;  const power = fighter.defense \* dodgeChance;  return power;  }  function checkKeysCriticalHit(keys, pressed) {  for (let key of keys) {  if (!pressed.has(key)) {  return false;  }  }  return true;  } |

Алгоритм для тестування ударів об стіни

|  |
| --- |
| private bool TestWallKicks(int rotationIndex, int rotationDirection)  {  int wallKickIndex = GetWallKickIndex(rotationIndex, rotationDirection);   for (int i = 0; i < data.wallKicks.GetLength(1); i++)  {  Vector2Int translation = data.wallKicks[wallKickIndex, i];   if (Move(translation)) {  return true;  }  }   return false;  }   private int GetWallKickIndex(int rotationIndex, int rotationDirection)  {  int wallKickIndex = rotationIndex \* 2;   if (rotationDirection < 0) {  wallKickIndex--;  }   return Wrap(wallKickIndex, 0, data.wallKicks.GetLength(0));  }   private int Wrap(int input, int min, int max)  {  if (input < min) {  return max - (min - input) % (max - min);  } else {  return min + (input - min) % (max - min);  }  } |

Необхідні данні для обчислень та перевірок

|  |
| --- |
| using System.Collections.Generic; using UnityEngine;  public static class Data {  public static readonly float cos = Mathf.Cos(Mathf.PI / 2f);  public static readonly float sin = Mathf.Sin(Mathf.PI / 2f);  public static readonly float[] RotationMatrix = new float[] { cos, sin, -sin, cos };   public static readonly Dictionary<Tetromino, Vector2Int[]> Cells = new Dictionary<Tetromino, Vector2Int[]>()  {  { Tetromino.I, new Vector2Int[] { new Vector2Int(-1, 1), new Vector2Int( 0, 1), new Vector2Int( 1, 1), new Vector2Int( 2, 1) } },  { Tetromino.J, new Vector2Int[] { new Vector2Int(-1, 1), new Vector2Int(-1, 0), new Vector2Int( 0, 0), new Vector2Int( 1, 0) } },  { Tetromino.L, new Vector2Int[] { new Vector2Int( 1, 1), new Vector2Int(-1, 0), new Vector2Int( 0, 0), new Vector2Int( 1, 0) } },  { Tetromino.O, new Vector2Int[] { new Vector2Int( 0, 1), new Vector2Int( 1, 1), new Vector2Int( 0, 0), new Vector2Int( 1, 0) } },  { Tetromino.S, new Vector2Int[] { new Vector2Int( 0, 1), new Vector2Int( 1, 1), new Vector2Int(-1, 0), new Vector2Int( 0, 0) } },  { Tetromino.T, new Vector2Int[] { new Vector2Int( 0, 1), new Vector2Int(-1, 0), new Vector2Int( 0, 0), new Vector2Int( 1, 0) } },  { Tetromino.Z, new Vector2Int[] { new Vector2Int(-1, 1), new Vector2Int( 0, 1), new Vector2Int( 0, 0), new Vector2Int( 1, 0) } },  };   private static readonly Vector2Int[,] WallKicksI = new Vector2Int[,] {  { new Vector2Int(0, 0), new Vector2Int(-2, 0), new Vector2Int( 1, 0), new Vector2Int(-2,-1), new Vector2Int( 1, 2) },  { new Vector2Int(0, 0), new Vector2Int( 2, 0), new Vector2Int(-1, 0), new Vector2Int( 2, 1), new Vector2Int(-1,-2) },  { new Vector2Int(0, 0), new Vector2Int(-1, 0), new Vector2Int( 2, 0), new Vector2Int(-1, 2), new Vector2Int( 2,-1) },  { new Vector2Int(0, 0), new Vector2Int( 1, 0), new Vector2Int(-2, 0), new Vector2Int( 1,-2), new Vector2Int(-2, 1) },  { new Vector2Int(0, 0), new Vector2Int( 2, 0), new Vector2Int(-1, 0), new Vector2Int( 2, 1), new Vector2Int(-1,-2) },  { new Vector2Int(0, 0), new Vector2Int(-2, 0), new Vector2Int( 1, 0), new Vector2Int(-2,-1), new Vector2Int( 1, 2) },  { new Vector2Int(0, 0), new Vector2Int( 1, 0), new Vector2Int(-2, 0), new Vector2Int( 1,-2), new Vector2Int(-2, 1) },  { new Vector2Int(0, 0), new Vector2Int(-1, 0), new Vector2Int( 2, 0), new Vector2Int(-1, 2), new Vector2Int( 2,-1) },  };   private static readonly Vector2Int[,] WallKicksJLOSTZ = new Vector2Int[,] {  { new Vector2Int(0, 0), new Vector2Int(-1, 0), new Vector2Int(-1, 1), new Vector2Int(0,-2), new Vector2Int(-1,-2) },  { new Vector2Int(0, 0), new Vector2Int( 1, 0), new Vector2Int( 1,-1), new Vector2Int(0, 2), new Vector2Int( 1, 2) },  { new Vector2Int(0, 0), new Vector2Int( 1, 0), new Vector2Int( 1,-1), new Vector2Int(0, 2), new Vector2Int( 1, 2) },  { new Vector2Int(0, 0), new Vector2Int(-1, 0), new Vector2Int(-1, 1), new Vector2Int(0,-2), new Vector2Int(-1,-2) },  { new Vector2Int(0, 0), new Vector2Int( 1, 0), new Vector2Int( 1, 1), new Vector2Int(0,-2), new Vector2Int( 1,-2) },  { new Vector2Int(0, 0), new Vector2Int(-1, 0), new Vector2Int(-1,-1), new Vector2Int(0, 2), new Vector2Int(-1, 2) },  { new Vector2Int(0, 0), new Vector2Int(-1, 0), new Vector2Int(-1,-1), new Vector2Int(0, 2), new Vector2Int(-1, 2) },  { new Vector2Int(0, 0), new Vector2Int( 1, 0), new Vector2Int( 1, 1), new Vector2Int(0,-2), new Vector2Int( 1,-2) },  };   public static readonly Dictionary<Tetromino, Vector2Int[,]> WallKicks = new Dictionary<Tetromino, Vector2Int[,]>()  {  { Tetromino.I, WallKicksI },  { Tetromino.J, WallKicksJLOSTZ },  { Tetromino.L, WallKicksJLOSTZ },  { Tetromino.O, WallKicksJLOSTZ },  { Tetromino.S, WallKicksJLOSTZ },  { Tetromino.T, WallKicksJLOSTZ },  { Tetromino.Z, WallKicksJLOSTZ },  };  } |

# **Висновки**

Під час виконання курсової роботи було застосовано більшість набутих знань і навичок з курсів Основи програмування. Використано можливості мови C#, такі як функції, експорт імпорт, умовні оператори, замикання, цикли, методи масивів.

Також застосовано можливості Unity як середи для програмною розробки, такі як створення сцен та елементів, додання логіки до них.

Було реалізовано всі функціональні вимоги згідно технічного завдання, проте через брак часу на розробку ще є потенціал розвитку гри. Серед можливих нововведень в майбутньому: внесення унікальних фігур, прискорення гри з плином часу, встановлення таймера для гри, створення різних режимів гри.

# Список використаних літературних джерел

1. Інструменти розробки Unity [Електронний ресурс]. — Режим доступу:  [https://unity.com/ru/developer-tools](https://javascript.info/)
2. C# Documentation [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide
3. Stack Overflow [Електронний ресурс]. — Режим доступу: <https://stackoverflow.com/>