Міністерство освіти і науки України

Коледж інформаційних технологій та землевпорядкування

Національного авіаційного університету

**КУРСОВИЙ ПРОЕКТ**

**з дисципліни «Об’єктно орієнтоване програмування»**

**на тему: Програмна реалізація ігрового модуля «Тетріс»**

Студента ІV курсу РПЗ-48 групи

Спеціальності «Розробка програмного забезпечення»

Смілого Едуарда Руслановича

Керівник \_\_Краліна А. С.\_\_\_\_

Оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_ \_\_Апенько Н. В.\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_ \_\_Краліна А. С.\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_ \_\_Кириченко В. С.\_

(підпис) (прізвище та ініціали)

Київ 2019

Міністерство освіти і науки України

Коледж інформаційних технологій та землевпорядкування

Національного авіаційного університету

ЗАТВЕРДЖУЮ

Голова комісії

\_\_\_\_\_\_\_\_\_\_\_ \_\_Апенько Н. В.\_\_\_\_

(підпис) (прізвище та ініціали)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

На курсовий проект

Студента Смілого Едуарда Руслановича

спеціальності «Розробка програмного забезпечення» курсу ІV

ТЕМА: Програмна реалізація ігрового модуля «Тетріс»

Вихідні данні:

* Програмна реалізація ігрового модуля «Тетріс» для браузерів Firefox, Google Chrome, Edge, Opera та інших на базі Chromium.

Зміст текстової частини до курсового проекту:

Індивідуальне завдання

Анотація

Вступ

Сучасні технології розробки ігор

Розробка програми

Керівництво користувача

Висновок

Список літератури

Дата видачі «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 р. Керівник Краліна А. С.

Завдання отримав\_\_\_\_\_\_\_\_\_\_

**Тема: Програмна реалізація ігрового модулю «Тетріс»**

**Календарний план виконання проекту:**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Назва етапу курсового проекту | Термін виконання етапу | Примітка |
| 1. | Отримання завдання на курсовий проект. |  |  |
| 2. | Підбір, огляд, необхідної літератури за темою роботи |  |  |
| 3. | Розробка алгоритму роботи програми |  |  |
| 4. | Програмування розробленого алгоритму |  |  |
| 5 | Застосування розробленого алгоритму до програмного продукту |  |  |
| 6. | Написання керівництва користувача |  |  |
| 7. | Аналіз отриманих даних з керівництвом, написання доповіді та попередній захист курсового проекту |  |  |
| 8. | Корегування роботи за даними попереднього захисту |  |  |
| 9. | Остаточне оформлення розділів курсового проекту |  |  |
| 10. | Захист курсового проекту |  |  |

СтудентСмілий Едуард Русланович

КерівникКраліна Анна Сергіївна

**«\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**АНОТАЦІЯ**

Курсовий проект містить 31 сторінку, 14 рисунків.

Ключові слова: об’єктно-орієнтоване програмування, С#, MVC, ECS, ігровий модуль, Unity.

Предмет дослідження – програмний модуль гри «Тетріс».

Мета роботи – створення зручної модульної архітектури для гармонійної роботи всіх компонентів програмної реалізації «Тетрісу», гнучкого розширення функціональних можливостей, а також зручної модифікації уже створеного коду за допомогою патернів програмування та компонентного підходу.

Курсовий проект оформлений у текстовому редакторі Microsoft Word та виконаний в програмному середовищі ігрового двигуна Unity на мові програмування С# з використанням технології OpenGL.

ЗМІСТ

|  |  |
| --- | --- |
| ВСТУП…………………………………………………………………………… | 6 |
| РОЗДІЛ 1. СУЧІСНІ ТЕХНОЛОГІЇ РОЗРОБКИ ІГОР |  |
| 1.1. Мови та методології програмування ігор……………..………………….... | 7 |
| 1.2. Основні поняття Entity Component System…….…………………………... | 11 |
| 1.3. Ігровий двигун для розробки ігор – Unity………………...……………….. | 12 |
| РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ |  |
| 2.1. Призначення та область застосування …………………………………...... | 15 |
| 2.2. Покрокова розробка …………………………………………………............ | 15 |
| 2.3. Опис алгоритму програми……………………………………………...…... | 25 |
| РОЗДІЛ 3. КЕРІВНИЦТВО КОРИСТУВАЧА |  |
| 3.1. Програмне і апаратне забезпечення програми…………………………….. | 26 |
| 3.2. Керівництво користувача…………………………………….……………... | 26 |
| ВИСНОВОК...……………………………………………………………………. | 30 |
| СПИСОК ЛІТЕРАТУРИ………………………….……………………………... | 31 |

**ВСТУП**

**Актуальність теми:** При швидкому розвитку світу інформаційних технологій та вимогам суспільства, програмування, десь більше десь менше, стало невід’ємною частиною нашого життя. З появою перших обчислювальних машин з’явилися різні типи мультимедіа, в тому числі й ігри. У наш час ігри стали подібними до реальності та вирішують настільки складний і широкий спектр задач, що спеціалістів ігро-індустрії охоче приймають на роботу у такі передові компанії світу, як Google, Microsoft, Apple або ж SpaceX. І це не говорячи вже про саму індустрію розваг, яка налічує тисячі компаній по всьому світу.

**Мета:** практикування розробки складних систем з використанням принципово нових підходів та методологій програмування для подальшого їх вивчення.

**Завдання:** реалізувати програмний модуль старої гри «Тетріс» з використанням новітніх середовищ розробки та методологій програмування.

Для досягнення мети необхідно:

1. Набути навички моделювання програм та систем за допомогою принципів компонентно-орієнтованого програмування.

2. Ознайомитися з сучасними фреймворками, бібліотеками, методологіями та патернами, що використовуються у ігрових двигунах для швидкої та гнучкої розробки.

3. Ознайомитися з побудовою передових ігрових двигунів та визначити їх ключові особливості серед конкурентів.

4. Дослідити та реалізувати стару концепцію за допомогою новітніх методів моделювання, проектування та програмування.

**РОЗДІЛ 1. СУЧАСНІ ТЕХНОЛОГІЇ РОЗРОБКИ ІГОР**

**1.1 Мови та методології програмування ігор**

Розробка відеогри – це процес створення відеогри, яким займається розробник відеоігор, котрий може бути як однією людиною так і компанією з сотнями співробітників. Гра може розроблятися як силами кількох людей з обмеженим бюджетом, так і спираючись на фінансування від видавця. Тривалість та вартість розробки залежить від складності проекту.

Перш за все хтось подає ідею створення нової гри, як правило він і стає керівником проекту. Поступово чи мозковим штурмом вирішується про що вона оповідатиме, яка її особливість може привернути увагу гравців; гра принесе щось кардинально нове або ж використовуватиме вже відомі та перевірені технології, тематику. Виходячи із запланованих переважних дій гравця, обирається жанр, наприклад, стратегія в реальному часі або шутер від першої особи. В ході роботи жанр може змінитися і вся концепція зазнати суттєвої переробки. Геймдизайнер вирішує якими стануть образи персонажів, ігрового світу, виконує попередні малюнки – концепт-арти. Гра отримує попередню назву, яка з часом може уточнитися чи зовсім змінитися.

Збирається команда розробників з програмістів, сценаристів, художників, композиторів, менеджерів і так далі. Визначаються строки виконання кожного виду роботи і орієнтовна дата виходу готової гри, яке обладнання, кошти для цього знадобляться. Та сама людина може займатися кількома справами. Наприклад, бути менеджером, координуючи виконанням роботи інших, і дизайнером, розроблячи образи персонажів.

На цьому етапі складається концепт-документ, в якому обґрунтовується чому буде вигідно створити дану гру. Дизайн-документ описує гру загалом, її початковий сюжет, ігровий процес, містить концепт-арти. Також створюється прототип, початкова проста версія гри, з якої можна скласти уявлення яким буде ігровий процес.

Оскільки відеогра є комп'ютерною програмою, її робота, технічні можливості, контент та ігровий процес, забезпечується програмним кодом. Розробка гри включає ті ж етапи, що і розробка програманого забезпечення, але передбачає більше роботи над контентом і створення ігрових механік.

Сучасні ігри здебільшого засновані на готових програмних модулях –ігрових рушіях, де вже реалізовані базові функції, здатні зв'язувати воєдино графіку, звук, об'єкти і їх рухи. Щоб налаштувати рушій для реалізації конкретного задуму програмісти доопрацьовують його, додаючи потрібні функції. Існують як вільні ігрові рушії, доступні будь-кому, так і ті, що вимагають отримання ліцензії на їх використання. Крім того рушії різняться за ліцензіями. Для незалежних розробників їх використання може бути значно дешевшим.

Деякі рушії розраховані на створення ігор конкретного жанру, інші –універсальні. Не всі рушії можуть забезпечити однакові внутрішньо-ігрові можливості та рівень графіки. Частина рушіїв дозволяють створювати ігри для різних платформ, так Unreal Development Kit підтримує розробку інтерактивних творів для PC, Xbox 360, PlayStation 3, Wii та Android.

Деякі ігри створюються в спеціальних програмах, які вже мають початкові ресурси, дії, та не вимагають знання мов програмування. Прикладами таких програм є Game Maker, Construct, RPG Maker.

Відеогра передбачає створення графіки, звуків та внутрішньо-ігрових текстів. Концепт-арти виконуються на папері або комп'ютері, зазвичай в кількох варіантах. На основі концепт-артів художниками затверджуються і створюються двовимірні або тривимірні моделі персонажів, предмети та декорації. Для цього художники працюють в програмах, призначених для роботи із графікою.

Щоб моделі рухалися, вони анімуються в інших спеціальних програмах. Створюються різні набори рухів, які відтворюватимуться залежно від конкретних дій гравця з допомогою програмного коду. У випадку двовимірної графіки це набори спрайтів, де кожна картинка є окремими кадром. Для реалізації реалістичних рухів чи емоцій може застосовуватися захоплення руху живих акторів. Після фіксування руху датчиками вони переносяться на комп'ютерного персонажа, як людина або чудовисько.

Візуальні ефекти роблять гру видовищнішою і задають стиль. Серед них деякі додають реалістичності, як відкидання тіней, заломлення світла, постріли і вибухи. Інші позначають стани і дії персонажа, які визначають стиль виконання гри. Деякі ігри цілком виконуються в стилі коміксу або кіноплівки. За реалізацію картинки і звуку відповідають графічний і звуковий рушії.

Для звукового оформлення пишеться музика і відбувається озвучування персонажів. Крім того для повноцінного звукового оформлення потрібні ефекти, як кроки, звуки пострілів. Вони можуть обиратися з вільних бібліотек чи записуватися окремо. Декотрі композитори спеціалізуються на створенні музики до ігор. Музика може виконуватися цілими професійними оркестрами, мати пісенний супровід. Діалоги персонажів часто озвучуються спеціальними акторами на студіях озвучування.

Часом ігри містять відео вставки, створені в програмах двовимірної чи тривимірної анімації. Інколи для відео вставок знімаються живі актори і будуються декорації. Є актори, які спеціалізуються саме на зйомках в таких відео вставках або озвучуванні персонажів. Сюжет, діалоги, додаткові тексти пишуться сценаристами і відповідальними за це письменниками.

Ігрова механіка визначає насиченість ігрового процесу, правила, за якими грається відеогра. Основою механіки є ігрові об'єкти, такі як персонажі, об'єкти, з якими вони можуть маніпулювати, декорації. Частиною ігрової механіки є управління, якими чином гравець керує персонажем та ігровим світом. Наприклад, як задається напрям руху, як активізується взаємодія з віртуальними предметами. Крім того на етапі розробки механіки створюється користувацький інтерфейс, який інформує гравця і дозволяє взаємодіяти зі світом гри.

Ігри зазвичай поділяються на рівні (локації), щоб комп'ютер не навантажувався обчисленням всього світу гри. В новітніх іграх світ часто моделюється так, щоб не мати чітких поділів на локації. Дизайнер рівнів розміщує готові об'єкти в ігровому світі та продумує їх рухи. Компонування рівнів визначає наскільки цікавою буде гра, які можливості будуть у гравця вирішити конкретну ситуацію.

За взаємодію об'єктів, яка відбувається без контрою гравця, відповідає фізичний рушій. До прикладу, він реалізує закони інерції, гравітацію, поведінку рідин, властивості предметів. Штучний інтелект (ШІ) відповідає за поведінку персонажів, як вони реагуватимуть на дії гравця. Багато подій в грі відбуваються за скриптами. Самі події придумуються сценаристами, а скрипти реалізуються програмістами.

Після завершення праці над кодом, контентом і механікою, за яких гра може функціонувати, відбувається її доопрацювання. Гра, не зібрана до кінця, але в яку можливо грати, називається альфа-версією. Вона може містити значні помилки і не доопрацювання, як відсутність певних можливостей, музики або об'єктів. Виявленням проблем займаються тестери, котрі грають в цю гру, намагаючись сповна скористатися всіма доступними можливостями в ній. Зазвичай на цьому етапі розробники записують рекламний трейлер, показуючи ігровий процес на відео, даючи потенційним гравцям уявлення про свою роботу. На пізнішому етапі виходить бета-версія, до тестування якої можуть залучатися і потенційні покупці гри. В бета-версії відбувається подальший пошук помилок, перевірка коректності взаємодії об'єктів ігрового світу, управління. Можливі внесення змін в оформлення, зміна ігрового балансу, можливостей персонажів.

**1.2 Основні поняття Entity Component System**

Entity-Component-System – це шаблон проектування, який забезпечує велику гнучкість в проектуванні загальної архітектури програмного забезпечення. Такі великі компанії, як Unity, Epic або Crytek використовують цей шаблон в своїх фреймворках, щоб надати розробникам дуже багатий можливостями інструмент, за допомогою якого вони розробляють власне ПЗ.

У 2007 році команда, що працювала над Operation Flashpoint: Dragon Rising, експериментувала з патерном ECS, а Адам Мартін пізніше написав докладний опис дизайну ECS, включаючи визначення основної термінології та концепцій. Зокрема, робота Мартіна популяризувала ідеї "систем" як елемента першого класу, "сутності як ідентифікатори", "компоненти як необроблені дані" і "код, що зберігається в системах, а не в компонентах або об'єктах".

Основна ціль ECS – розподіл різних проблем і завдань між сутностями (Entities), компонентами (Components) і системами (Systems). Це три основні поняття цього шаблону. Сутність це контейнер для компонентів. Компонент – це невеликий об’єкт, який не володіє ніякою логікою. Зазвичай компонент містить одне чи два поля. В ідеалі це об'єкти з простою структурою даних. Кожен тип компонента можна прикріпити до сутності, щоб дати їй щось на зразок властивості. Системи містять в собі логіку, наприклад, переміщення чи запуск анімації. Системи працюють с сутностями, які мають потрібний для неї набір компонентів.

Подібний підхід до структурування коду дозволяє позбутися проблеми «слабкого базового класу» та розподіляти весь код програми на незалежні функціональні елементи, не пов’язані між собою. Таким чином якщо якийсь з компонентів складної системи вийде з ладу, йому не вдасться повністю зламати програму. Навіть більше, він ніяк не вплине на роботу інших компонентів, що дозволяє продовжувати експлуатацію без використання непрацюючого модулю, аж до повного його аналізу та виправлення всіх помилок.

**1.3 Ігровий двигун для розробки ігор – Unity**

Unity – багатоплатформовий інструмент для розробки дво- та тривимірних додатків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосування працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360.

Є можливість створювати інтернет-додатки за допомогою спеціального під'єднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосування, створені за допомогою Unity, підтримують DirectX та OpenGL.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій підтримує три сценарних мови: C #, JavaScript (модифікація). Проект в Unity ділиться на сцени (рівні) – окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, об'єкти (моделі), так і порожні ігрові об'єкти – тобто ті, які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. Також у них є назва (в Unity допускається наявність двох і більше об'єктів з однаковими назвами), може бути тег (мітка) і шар, на якому він повинен відображатися. Так, у будь-якого предмета на сцені обов'язково присутній компонент Transform – він зберігає в собі координати місця розташування, повороту і розмірів по всіх трьох осях. У об'єктів з видимою геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить модель видимою.

Також Unity підтримує фізику твердих тіл і тканини, фізику типу Ragdoll (ганчіркова лялька). У редакторі є система успадкування об'єктів; дочірні об'єкти будуть повторювати всі зміни позиції, повороту і масштабу батьківського об'єкта. Скрипти в редакторі прикріплюються до об'єктів у вигляді окремих компонентів.

При імпорті текстури в рушій можна згенерувати alpha-канал, mip-рівні, normal-map, light-map, карту відображень, проте безпосередньо на модель текстуру прикріпити не можна – буде створено матеріал, з яким буде призначений шейдер, і потім матеріал прикріпиться до моделі. Редактор Unity підтримує написання і редагування шейдерів. Крім того він містить компонент для створення анімації, яку також можна створити попередньо в 3D-редакторі та імпортувати разом з моделлю, а потім розбити на файли.

Графічний рушій використовує DirectX (Windows), OpenGL (Mac, Windows, Linux), OpenGL ES (Android, iOS), та спеціальне власне API для Wii. Також підтримуються bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), динамічні тіні з використанням shadow maps, render-to-texture та повноекранні ефекти post-processing.

Unity підтримує файли 3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks та Allegorithmic Substance. В ігровий проект Unity можна імпортувати об'єкти цих програм та робити налаштовування за допомогою графічного інтерфейсу.

Для написання шейдерів використовується ShaderLab, що підтримує шейдерні програми написані на GLSL або Cg. Шейдер може включати декілька варіантів реалізації, що дозволяє Unity визначати найкращий варіант для конкретної відеокарти. Unity також має вбудовану підтримку фізичного рушія Nvidia PhysX (колишнього Ageia), підтримку симуляції одягу в системі реального часу на довільній та прив'язаній полігональній сітці (починаючи з Unity 3.0), підтримку системи ray casts та шарів зіткнення.

Скриптова система ігрового рушія зроблена на Mono – вільний відкритий проект з реалізації .NET Framework. Програмісти можуть використовувати UnityScript (власна скриптова мова, подібна до JavaScript та ECMAScript), C# або Boo (мова програмування, подібна до Python). Починаючи з версії 3.0, до Unity входить перероблена версія MonoDevelop для зневадження скриптів.

Із виходом версії 5.2 передбачається вбудована можливість редагувати скрипти у середовищі Visual Studio.

В Unity включено систему контролю версій Unity Asset Server для ігрових об'єктів та скриптів. Система використовує PostgreSQL, роботу зі звуком, побудовану на основі бібліотеки FMOD (з можливістю програвати Ogg Vorbis аудіофайли), відеопрогравач із кодеком Theora, рушій для побудови ландшафтів рослинності, вбудовану систему карт освітлення (Beast), мережу для мультиплеєру (RakNet) та вбудовані навігаційні меші для пошуку шляху.

**РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ**

**2.1 Призначення та область застосування**

Призначення – поглиблене вивчення сучасних методологій та паттернів і порівняння їх з фундаментальними та базовими методами розробки.

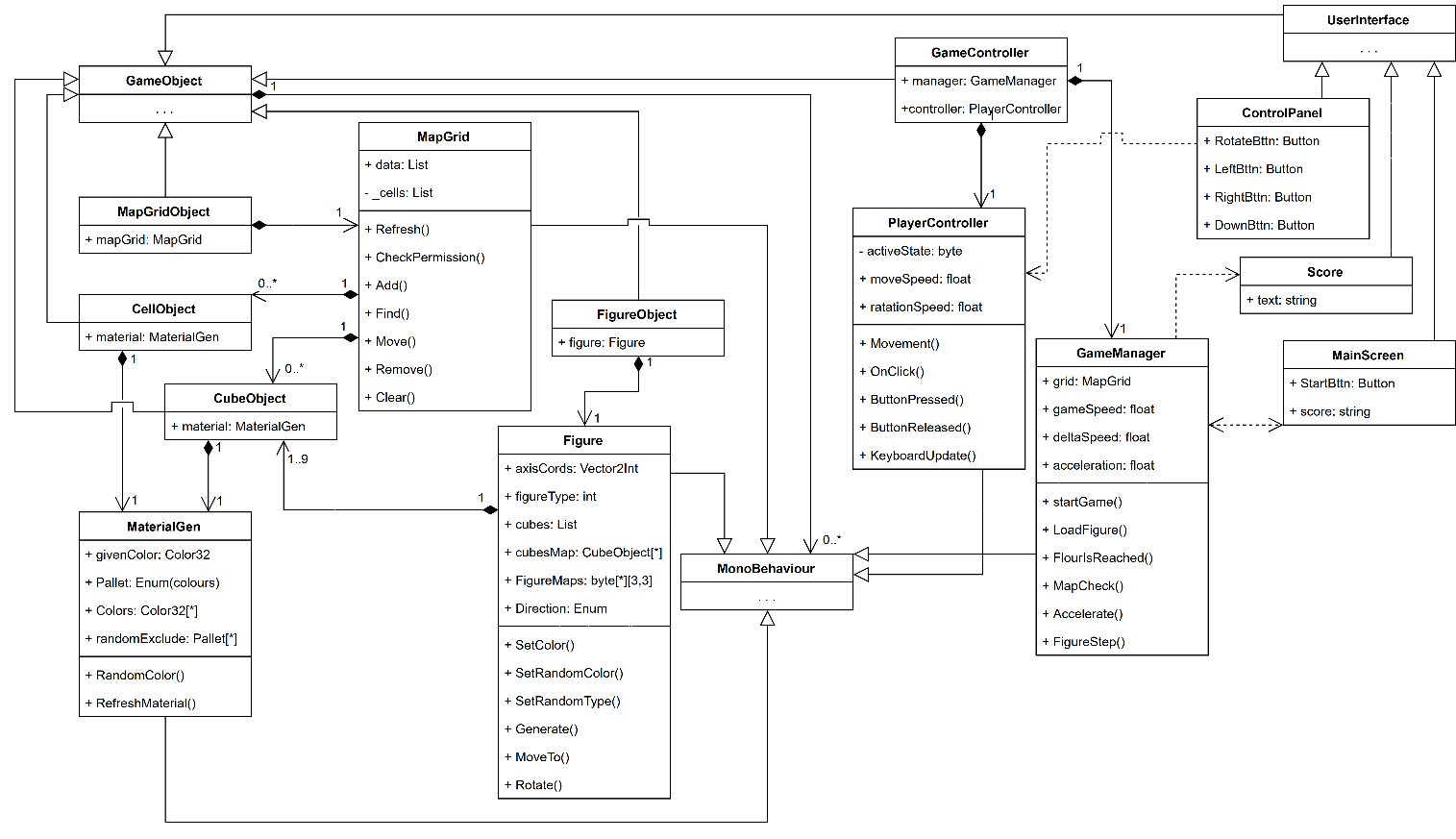
Область застосування – використовується у навчальних цілях для наочної демонстрації потужності новітніх практик розробки програмних продуктів.

**2.2 Покрокова розробка**

У процесі розробки проекту використовуємо ігровий двигун Unity 2018 та відповідний фреймворк для С# на програмному середовищі Visual Studio 2017.

Все починається з документування необхідної логіки та, в результаті, побудови діаграми класів. Фреймворк уже містить власну реалізацію методологій Entity Component System. Середовищем надається контейнер сутностей та клас графічних об’єктів на сцені – GameObject, а також інтерфейс логіки та реалізації окремих компонентів – MonoBehavior. В сукупності з потужною та налагодженою системою залежностей, обробників подій, «колекціонерів сміття» та інших проявів ECS це утворює неймовірно гнучке середовище для побудови незалежних та стресо-стійких компонентно-орієнтованих архітектур.

Відповідно до вимог програми створюємо необхідний перелік графічних класів, які будуть наслідувати GameObject та перелік компонентів, які наслідуватимуть MonoBehavior та побудуємо діаграму класів з описом їх взаємодії (рис. 2.1).

 Рис.2.1 – Побудова діаграми класів проекту.

Наступним кроком створюємо, та налаштовуємо новий 3Д проект на Unity. Через графічний інтерфейс двигуна створюємо дочірні класи від GameObject. Додаємо на сцену модель кубу та модель клітинки для подальшої роботи з картою та розміщуємо у просторі кімнати програми (рис. 2.2).

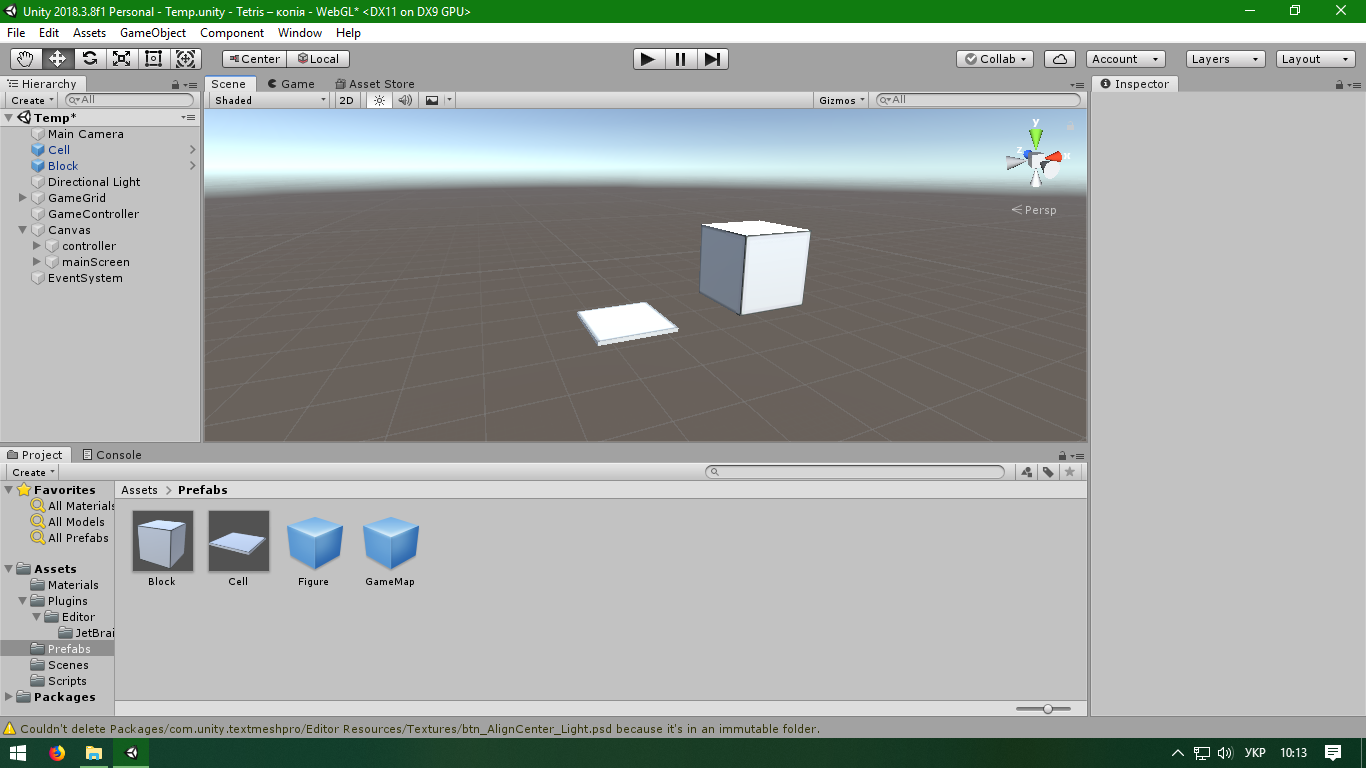


Рис.2.2 – Розміщення графічних об’єктів.

Далі для надання програмі стилю знадобляться різноманітні текстури унікальних кольорів. Щоб не завантажувати набір текстур кольорової палітри вручну створимо генератор матеріалів, який перефарбовує будь-який елемент на сцені у вибраний колір. Для цього створимо MaterialGen компонент:

*public class MaterialGen : MonoBehaviour {*

*public Color32 color ;*

*public Color32 givenColor {*

*get => color;*

*set {*

*color = value;*

*RefreshMaterial();*

*}*

*}*

*void RefreshMaterial() {*

*Renderer rend = GetComponent<Renderer>();*

*rend.material.color = givenColor;*

*rend.sharedMaterial.SetFloat("\_Metallic", 0.35F);*

*rend.sharedMaterial.SetFloat("\_Glossiness", 0.35F);*

*}*

*}*

Після цього варто додати базову палітру приємних на вигляд кольорів, щоб витримувати гру в одному стилі. Для цього додаємо ряд константних значень:

public enum Pallet { Red, Pink, Purple, DeepPurple, Indigo, Blue, LightBlue, Cyan, Teal, Green, LightGreen, Lime, Yellow, Amber, Orange, DeepOrange, Brown, White, BlueGrey, Grey, Black};

*public static readonly Color32[] Colors = {*

*new Color32(198, 40, 40, 255),*

*. . .*

*new Color32(27, 27, 27, 255),*

*};*

Тепер за допомогою енумерації Pallet можна отримати відповідний колір з масиву Colors. Постійно задавати ці значення буде не зручно враховуючи постійну зміну кольорів фігур тетрісу. Отже необхідно створити метод випадкового вибору кольору і його оновлення:

*public Color32 RandomMaterialColor(){*

*return givenColor = RandomColor(randomExclude);*

*}*

*private static Random \_rand = new Random();*

*public static Color32 RandomColor(Pallet[] exclude = null) {*

*int value;*

*do{*

*value = \_rand.Next(0,Colors.Length);*

*} while (exclude != null && Array.IndexOf(exclude, (Pallet) value) != -1);*

*return Colors[value];*

*}*

Також для зручності роботи з редактором реалізуємо набір тегів параметру Color, що дозволить змінювати матеріал не запускаючи проект та не виправляючи код. В результаті отримуємо гнучкий генератор матеріалів з вбудованою палітрою кольорів у сучасному та гармонійному стилі (рис.2.3).

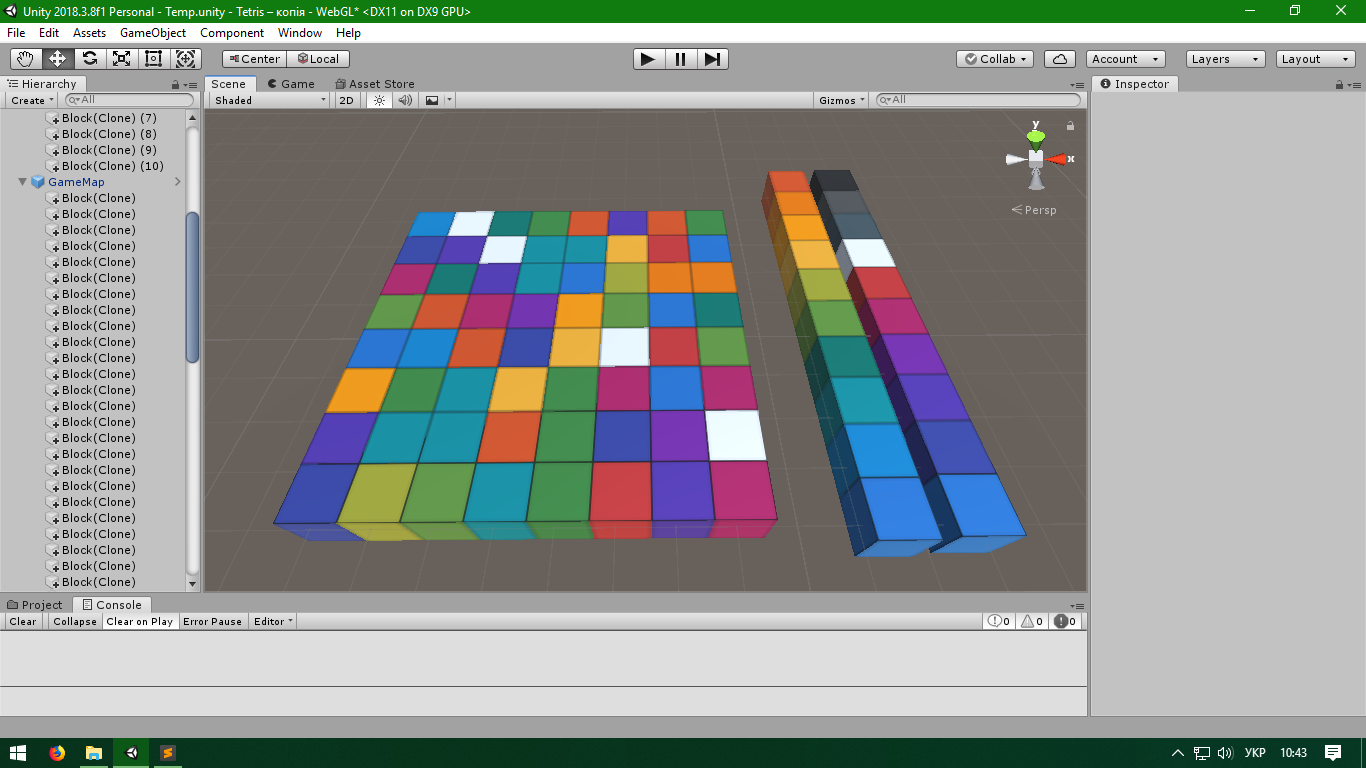


Рис.2.3 – Тестування генератору матеріалів у двигуні.

Далі для гри в Тетріс обов’язково знадобиться карта з квадратною сіткою. Для гнучкості і подальшого карти в майбутніх проектах реалізуємо його максимально незалежним та надамо зручний інтерфейс користування:

*public class MapGrid : MonoBehaviour {*

*public Vector2Int gridSize;*

*public Vector2 deltaSpace;*

*public GameObject cell;*

*public List<GameObject> \_cells;*

*void Refresh() {*

*\_cells = \_cells ?? new List<GameObject>();*

*for (int i = \_cells.Count-1; i >= 0; i--) {*

*DestroyImmediate(\_cells[i]);*

*\_cells.RemoveAt(i);*

*}*

*for (int ySize = 0; ySize < gridSize.y ; ySize++) {*

*for (int xSize = 0; xSize < gridSize.x ; xSize++) {*

*GameObject newCell = Instantiate(cell, transform);*

*\_cells.Add(newCell);*

*newCell.transform.localPosition = new Vector3(*

*transform.position.x + deltaSpace.x \* xSize, 0,*

*transform.position.z + deltaSpace.y \* ySize);*

*} } }*

Таким чином реалізовано гнучкий алгоритм генерації карти, що створює багаторазові копії базового елементу (одного квадрату) з урахуванням відстані між ними(рис. 2.4).

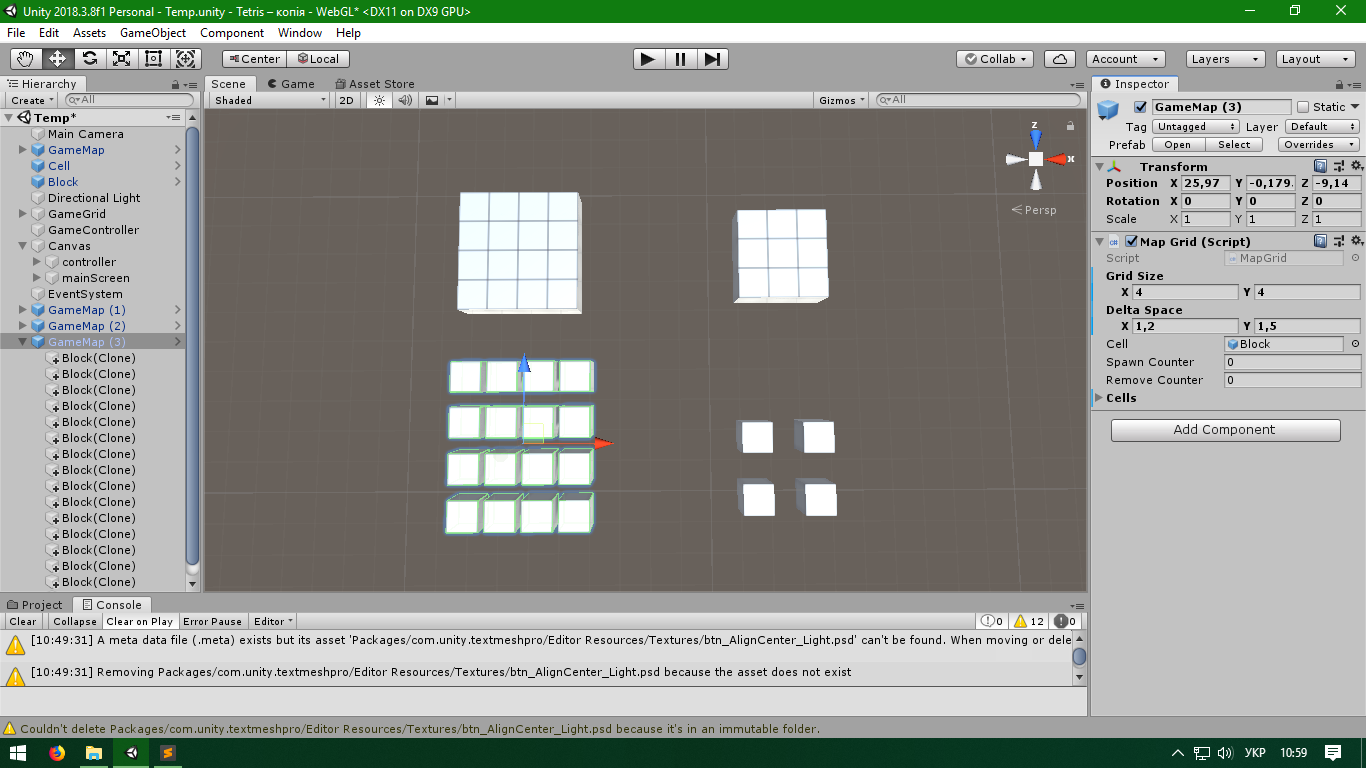


Рис.2.4 – Тестування гнучкості генератору сітки карти.

Настав час реалізувати додавання сторонніх об’єктів на базову карту. Для цього необхідно розширити функціональність поточного компоненту. Реалізувати: метод додавання нових елементів;

public bool CheckPermission(Vector2Int position) => 0 <= position.x && position.x < gridSize.x && 0 <= position.y && position.y < gridSize.y;

*public bool Add(GameObject obj, Vector2Int position, bool ignore = false){*

*if (obj == null || !CheckPermission(position) || Data[(int)position.y][(int) position.x] != null) return false;*

*RemoveByPos(Find(obj));*

*Data[(int) position.y][(int) position.x] = obj;*

*Vector3 selfPos = \_cells[(int)(gridSize.x \* position.y + position.x)].transform.position;*

*obj.transform.position = new Vector3(selfPos.x, selfPos.y, selfPos.z);*

*return true;*

*}*

Метод пошуку елементів;

*public Vector2Int Find(GameObject obj)*

*{*

*for (int y = gridSize.y - 1; y >= 0; y--)*

*{*

*for (int x = gridSize.x - 1; x >= 0; x--)*

*{*

*if (Data[y][x] == obj)*

*{*

*return new Vector2Int(x,y);*

*}*

*}*

*}*

*return Vector2Int.one \* -1;*

*}*

Метод видалення непотрібних елементів.

*public GameObject RemoveByPos(Vector2Int position, bool ignore = false)*

*{*

*if (!CheckPermission(position))*

*return null;*

*GameObject oldObject = Data[(int) position.y][(int) position.x];*

*if (!oldObject)*

*return null;*

*Data[(int) position.y][(int) position.x] = null;*

*if (!ignore)*

*removeCounter++;*

*return oldObject;*

*}*

Також варто реалізувати переміщення об’єктів, що являє собою видаленням та додаванням їх же в межах сітки.

Після реалізації карти варто перейти до реалізації того, що буде на ній – фігур. Сукупності квадратів будуть поєднуватися в фігури Тетрісу для подальшої зручної роботи з ними. Створимо компонент фігури та додамо метод генерації об’єктів у певній формі:

*public bool Generate(Vector2Int cords, MapGrid mapGrid = null, int type = -1){*

*bool isSuccessful = true;*

*grid = (mapGrid == null) ? grid : mapGrid;*

*figureType = (type == -1) ? figureType : type;*

*var chosenMap = FigureMaps[figureType];*

*for (int y = chosenMap.GetLength(0)-1; y >= 0; y--) {*

*for (int x = chosenMap.GetLength(1)-1; x >= 0; x--) {*

*if (chosenMap[y, x] == 1) {*

*GameObject current = Instantiate(cube, transform);*

*if (!grid.Add(current, new Vector2Int(cords.x + x, cords.y + y)))*

*isSuccessful = false;*

*cubes.Add(current);*

*cubesMap[y,x] = current;*

*}}}*

*axisCords = cords;*

*return isSuccessful;*

*}*

Порядок квадратів фігури буде передаватися двовимірним масивом. Разом з ним буде йти інформація про колір та позицію самої фігури. Для зручності контролю квадратами реалізовуємо паттерн «фасад» та метод переміщення і повороту.

Тепер необхідно створити перехоплювач подій натискання на клавіатуру, кнопок інтерфейсу та, власне, сам інтерфейс гри(рис. 2.5). Також варто винести кілька властивостей компоненту для коригування швидкості реакції на подію та ліміту частоти повторювання викликів. Для зручності реалізуємо змінні швидкості переміщення та швидкості обертання.

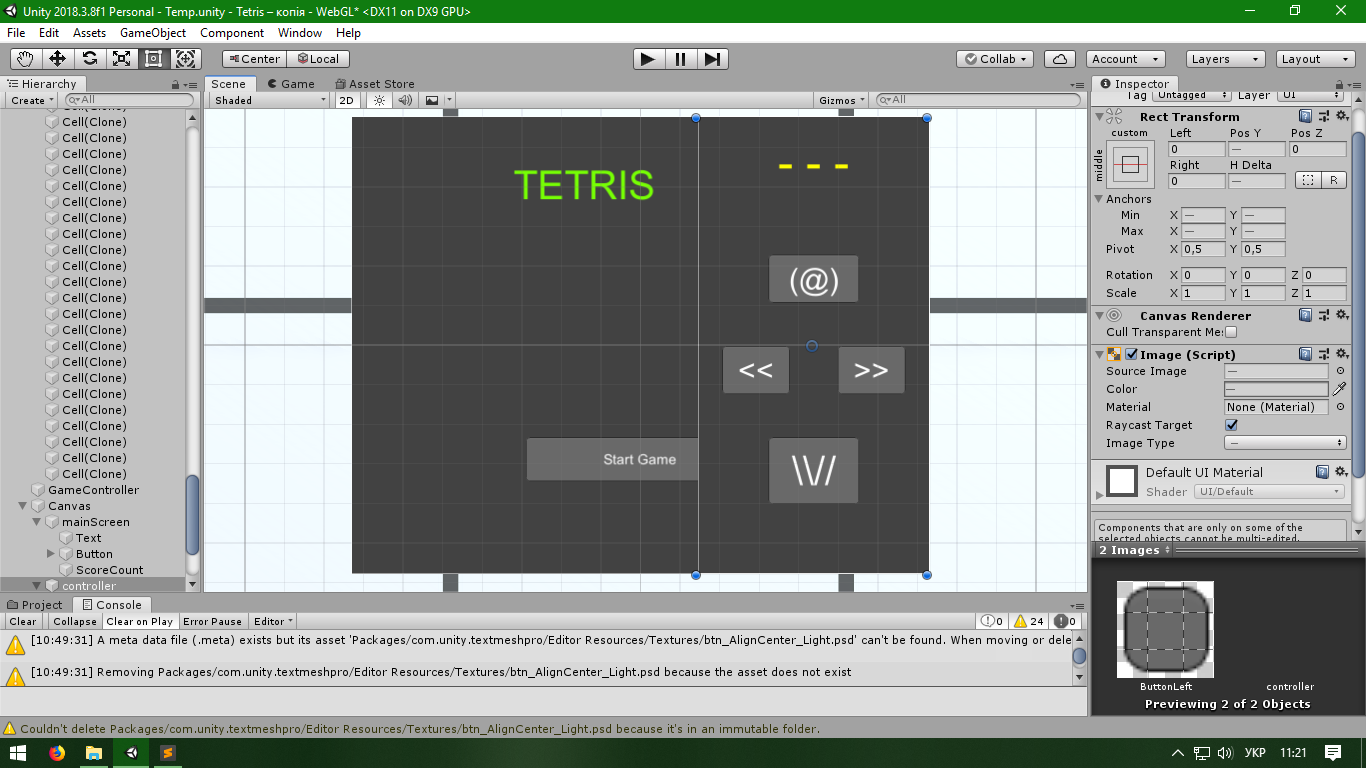


Рис.2.5 – Створення головного екрану та інтерфейсу гри.

Тепер, коли готові всі функціональні компоненти, настав час створити «мозок» гри – GameManager. Він являється ключовим компонентом системи, поєднує між собою логіку окремих компонентів в одне ціле та керує загальним ходом програми. Все, що необхідно зробити – організувати у ньому відповідні зв’язки для злагодженого виконання завдань, а також перехоплювач ключових подій гри, щоб реагувати на додавання нових елементів, виявлення та видалення повних рядків, запуск гри, ведення рахунку тощо:

*public class GameManager : MonoBehaviour { . . .*

*public void StartGame() {*

*mainScreen.SetActive(false);*

*grid.Start(); LoadFigure();*

*StartCoroutine(nameof(FigureStep)); StartCoroutine(nameof(Accelerate));*

*}*

*public void FlourIsReached() {*

*MapCheck(); LoadFigure();*

*}*

*. . .*

*}*

Налагодивши усі змінні, що впливають на ігровий процес та скомпонувавши всі частини програми воєдино можемо спестерігати гнучку архітектуру розробки в дії, що дозволяє без зайвих зусиль змінити структуру карти або особливості інших компонентів гри (рис. 2.6).

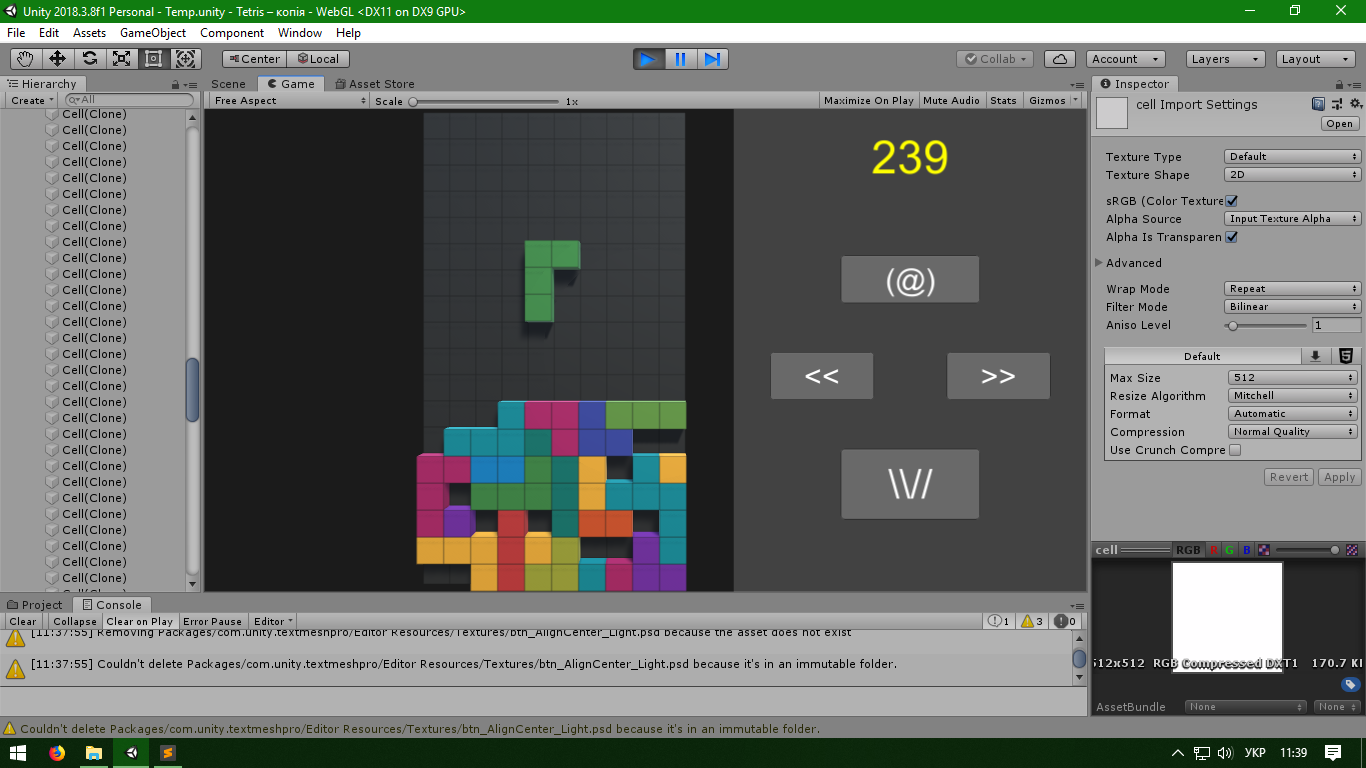


Рис.2.6 – Стандарний режим роботи гри

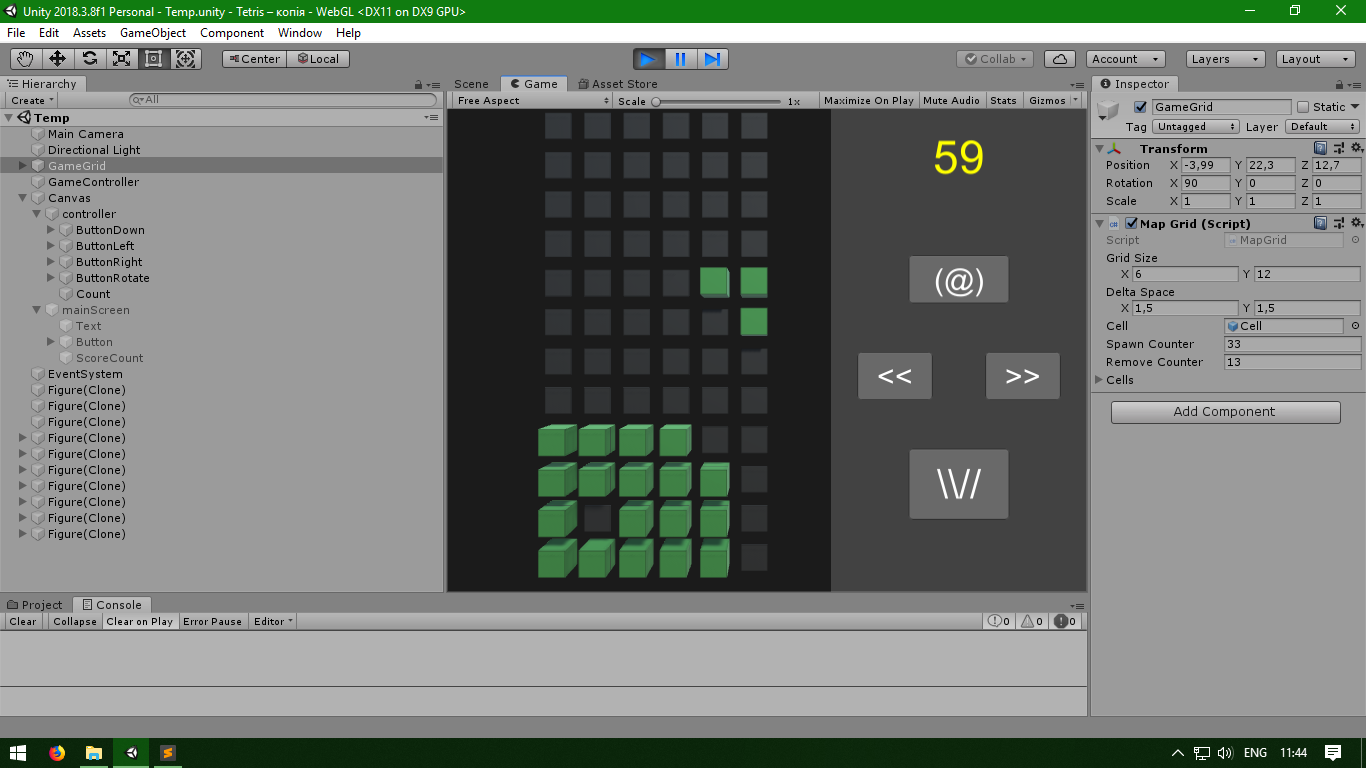


Рис.2.7 – Режим зі змінами характеристик карти та фігур

По завершенню скомпонувати проект для роботи у інтернет браузері з використанням технології OpenGL.

**2.3. Опис алгоритму програми**

Основний алгоритм роботи програми можна представити таким чином:

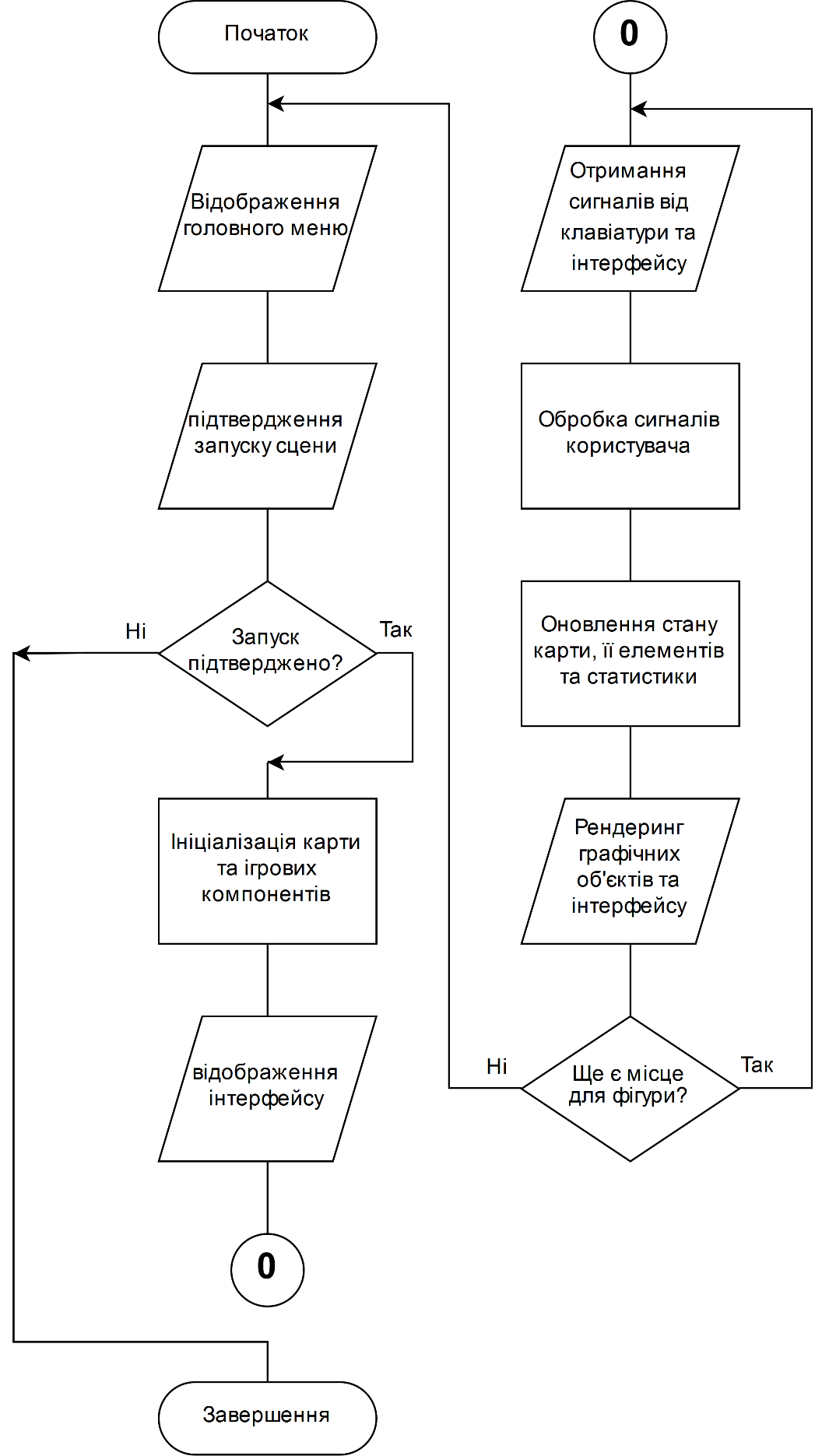


Рис.2.8 – Схема алгоритму роботи проекту.

**РОЗДІЛ 3. КЕРІВНИТВО КОРИСТУВАЧА**

**3.1 Програмне і апаратне забезпечення програми**

Програма представлена у вигляді пакету файлів з форматом .rar та має розмір 5 Mb. Створена для середовищ інтернет браузерів Firefox, Edge, Safari, Google Chrome та інших на базі ядра Chromium на пристроях з підтримкою SFML/OpenGL. Має «обгортку» у вигляді index.html файлу для вбудовування в сайт. Програмні й апаратні вимоги:

* Інтернет браузер Firefox, Edge, Safari, Google Chrome або інший на базі ядра Chromium;
* графічний драйвер SFML/OpenGL;
* 0.5 Gb оперативної пам’яті пристрою.

**3.2 Керівництво користувача**

Для використання програми необхідно розпакувати застосунок з архіву «TetrisModule.rar» до зручної вам папки. При умові належного програмного та апаратного забезпечення запускаємо базову сторінку index.html за допомогою браузеру та очікуємо завантаження гри (рис.3.1).

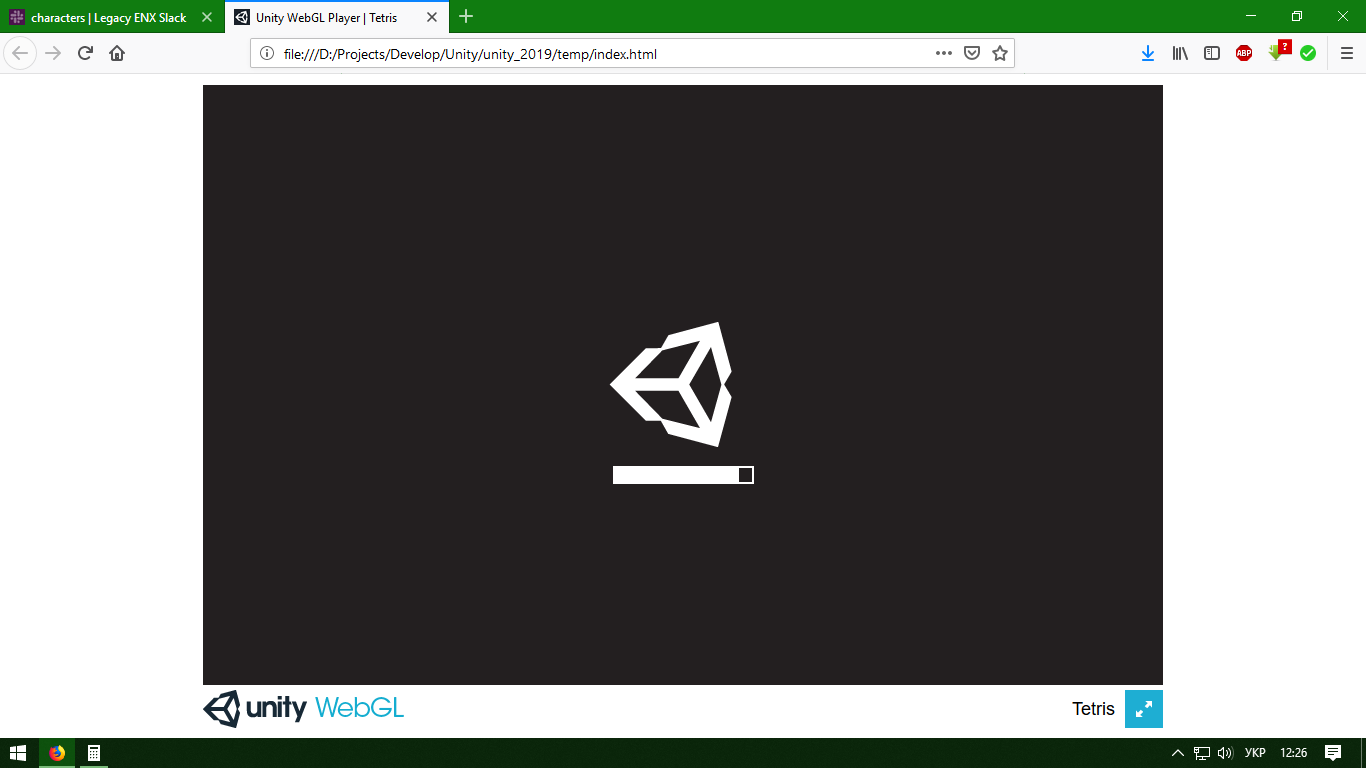


Рис.3.1 – Меню підготовки програми.

По завершенню процесу завантаження відображається головне меню.

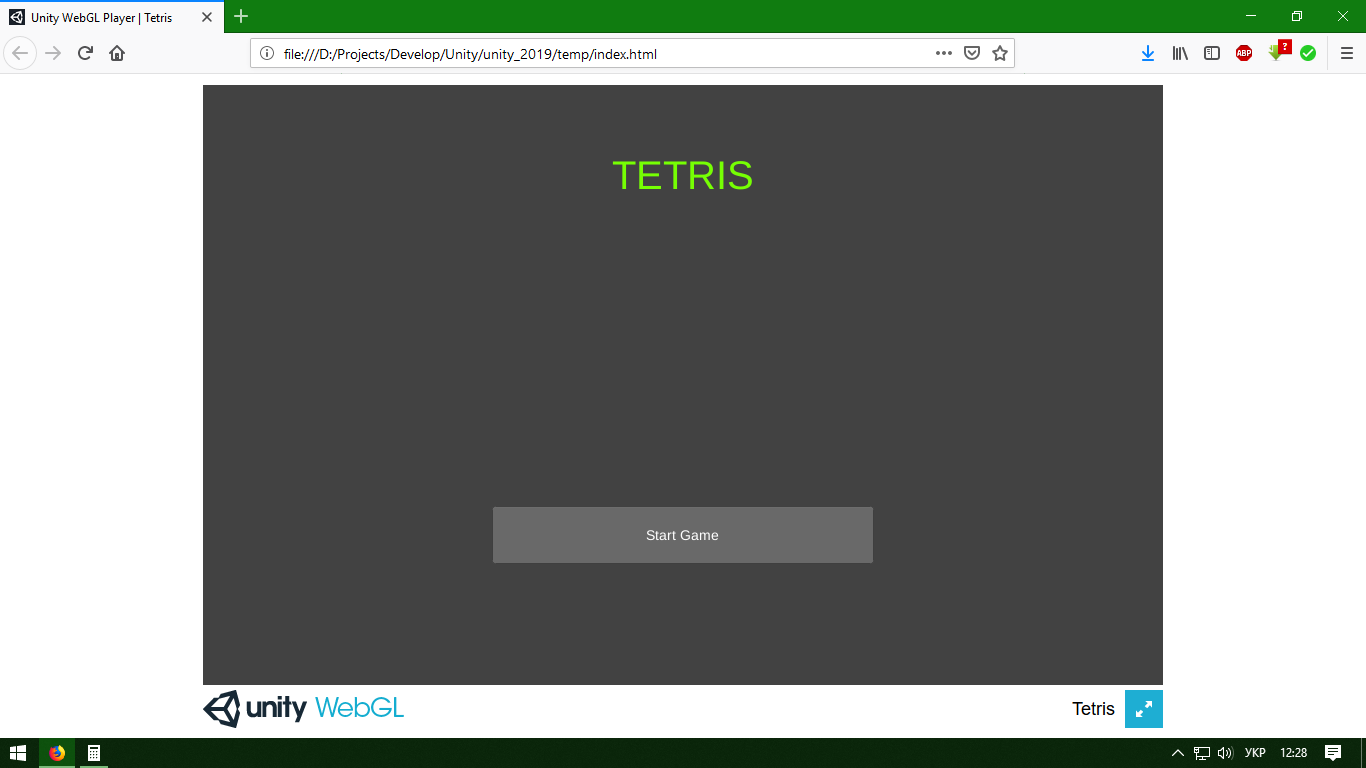


Рис.3.2 – Головне меню застосунку.

Меню містить у собі назву самої гри та кнопку старту. У випадку повторного відкриття меню без перезавантаження застосунку, сторінки або браузеру також по середині буде відображено жовтим текстом сумарна кількість набраних очок за попередню гру (рис.3.3).

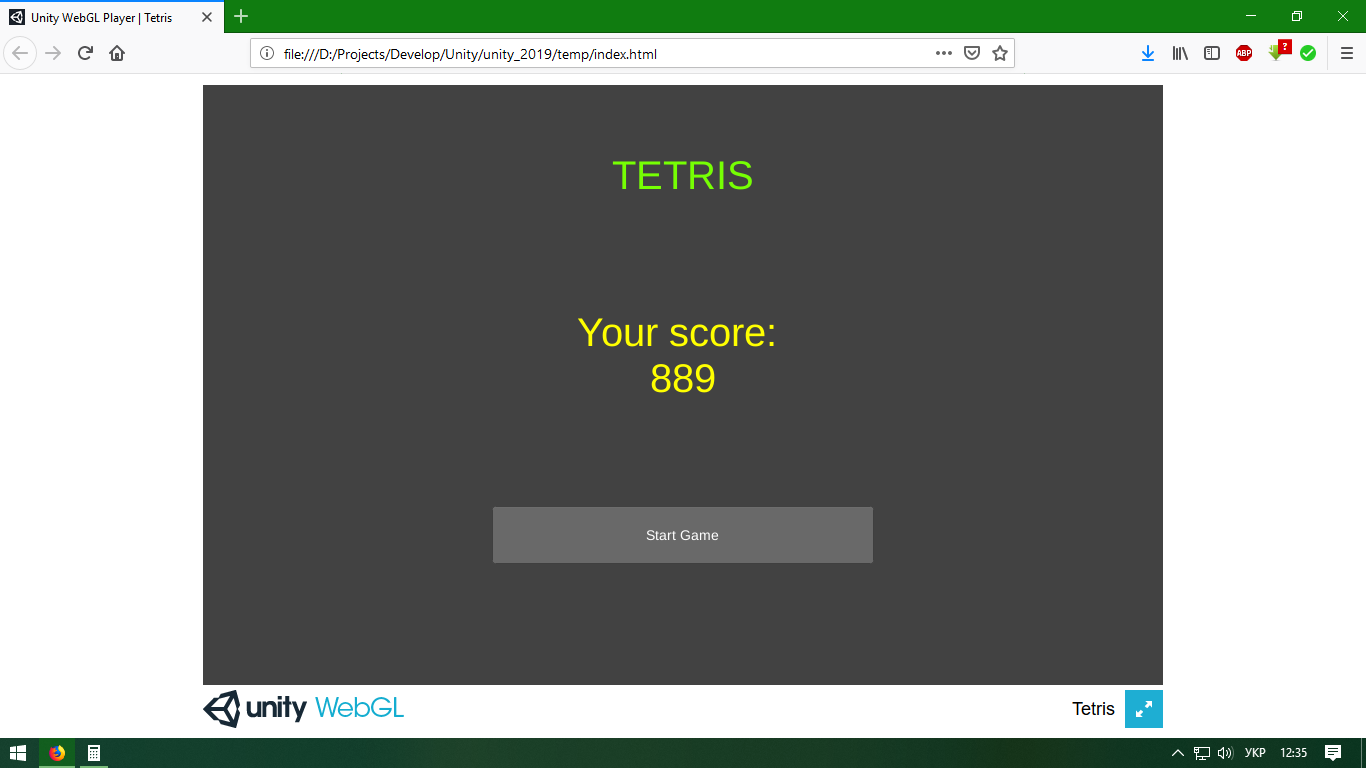


Рис.3.3 – Головне меню застосунку.

Після натискання на кнопку початку гри «Start Game» завантажується ігрова сцена та додатковий користувацький інтерфейс для керування фігурою (рис.3.4).

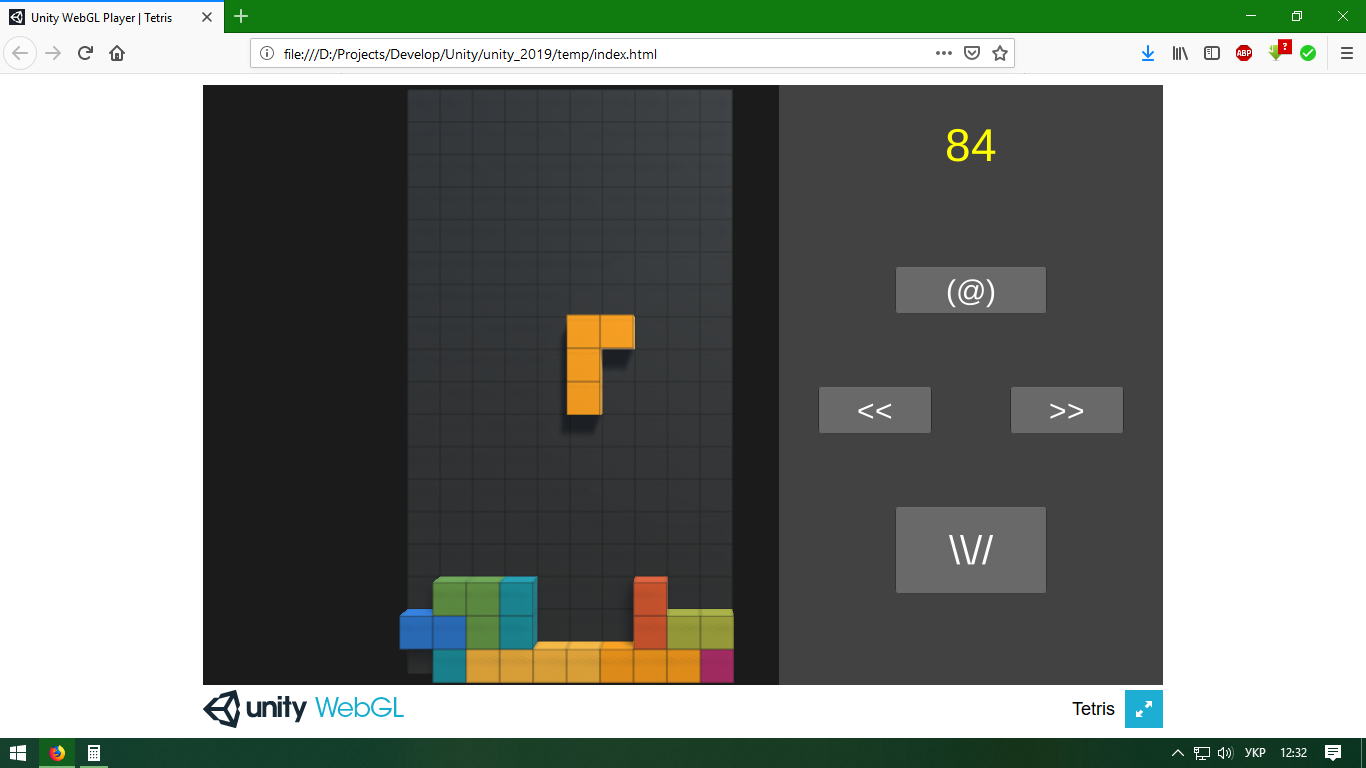


Рис.3.4 – Вигляд карти та інтерфейс гри.

На запуску карта пуста, але з плином часу з’являється різноманітні фігури, які потрібно припасовувати одна до одної. У випадку абсолютного заповнення одного з горизонтальних рядів, його кубики вилітають, а всі елементи вище опускаються на одну позицію вниз (рис.3.5).

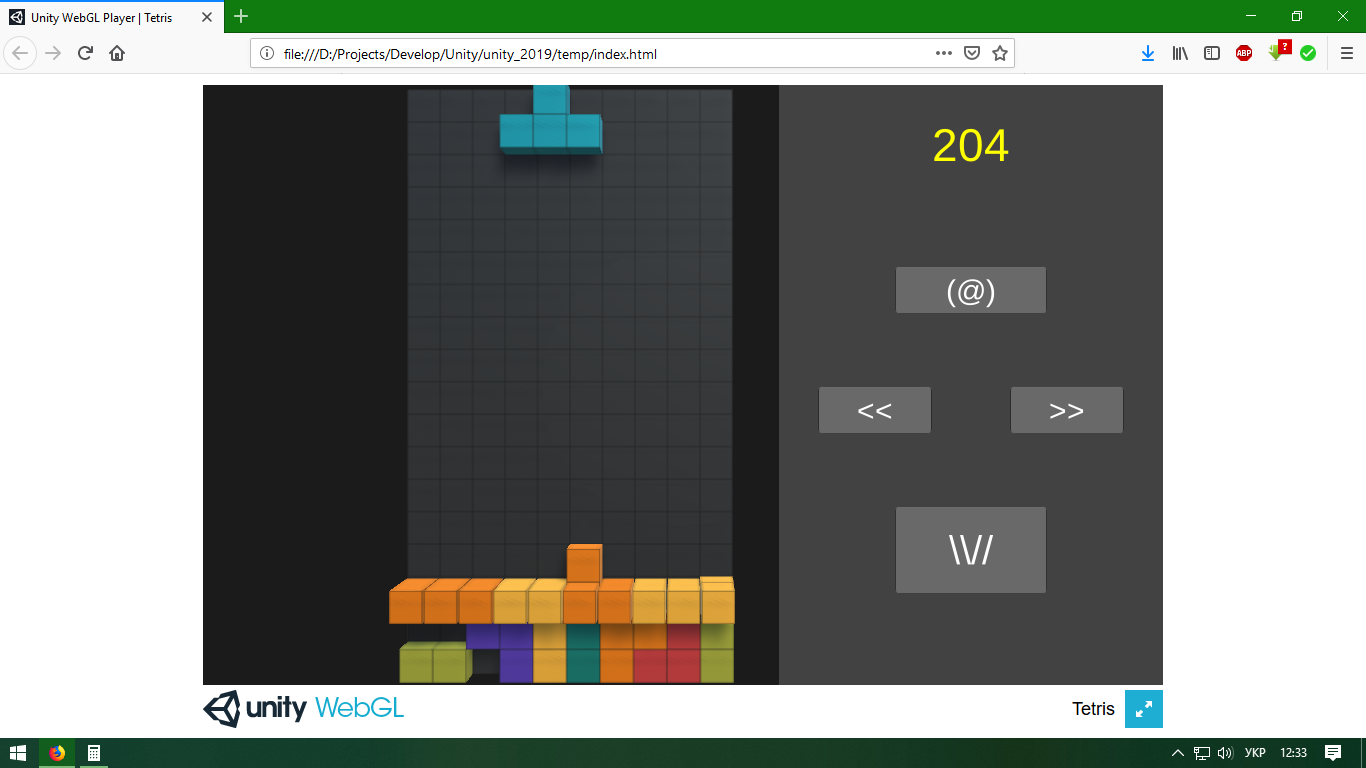


Рис.3.5 – Абсолютне заповнення горизонтального ряду.

Керувати ж можна тільки поточною рухомою фігурою. Щойно фігура опускається до підлоги, відразу з’являється нова. Фігуру можна рухати горизонтально, крутити та прискорювати падіння. У випадку з підключеною клавіатурою є такі варіанти контролю: «» - вліво, «» - вправо, «» - вниз, «» -поворот по годинниковій стрілці, «» - поворот проти годинникової стрілки. Якщо ж гра відбувається на мобільному пристрої без клавіатури (телефоні, планшеті, тощо) передбачене керування за допомогою кнопок інтерфейсу, де: «<<» - вліво, «>>» - вправо, «\\//» - вниз, «(@)» -поворот по годинниковій стрілці.

По ходу гри швидкість падаючих блоків геометрично прискорюється, що підвищує складність прямопропорційно до часу.

Гра завершується, при умові, що уже немає вільного місця для додавання нової фігури. У такому разі скидаються всі пришвидшення та сама карта і запускається початковий екран з результатами за поточну сесію.

По завершенню використання гри достатньо закрити вкладку поточного браузеру (рис.3.6).

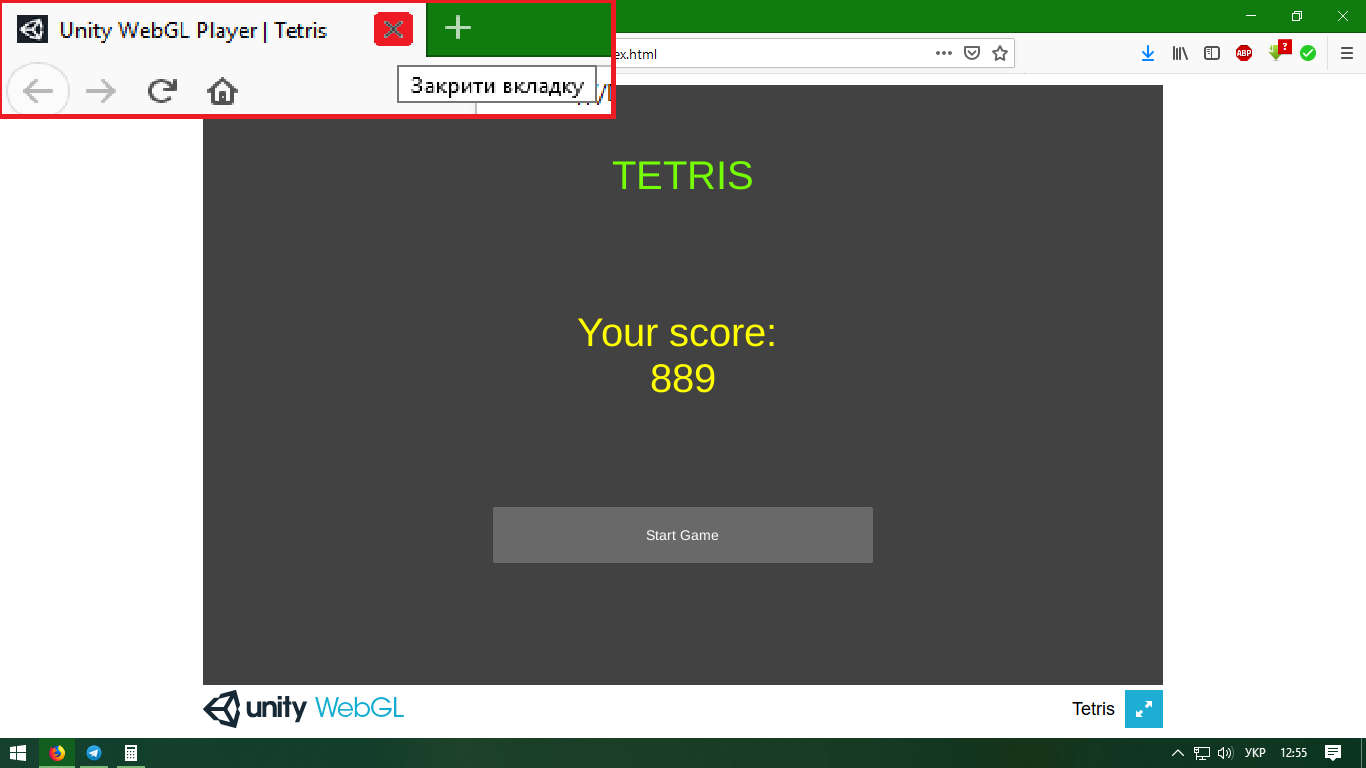


Рис.3.6 ­– Закриття вкадки з грою.

Для повторного використання достатньо запустити базову сторінку index.html та пройти всі кроки експлуатації від початку.

**ВИСНОВОК**

У процесі створення програмного модулю «Тетріс» розглянуто процес та сучасні методології розробки програмних продуктів.

При створенні розглянута мова програмування С#, а також її базові бібліотеки та фреймворки, такі як C# .NET Framework, C# CLI library. Також розглянуто фреймворк Unity3D та двигун розробки ігор Unity 2018.

Детально опрацьовано принципи та парадигми компонентно-орієнтованого програмування та популярна зараз Entity Component System. У ході розробки задіяні сучасні патерни та методології, такі як «одинак», «фасад», MVC (Model View Controller) тощо. Реалізовано наслідування від базових класів фреймворку, щоб перейняти їх логіку та мати можливість використовувати злагоджену компонентну архітектуру.

Проведено порівняння старого підходу об’єктно-орієнтованого програмування та сучасних методологій. Також переглянуто готові середовища для розробки програмних продуктів та двигуни – для ігор.

Фреймворк та сам ігровий двигун Unity 2018 – це потужне середовище розробки найрізноманітніших ігор та програм, що дозволяє створювати проект на будь-яку з бажаних платформ. Unity підтримує роботу як з DirectX так і з OpenGL, що підкреслює його кросплатформерність. Це означає що практично без втрати якості проект може бути перенесений з Windiws платформи на MacOS та Linux подібні. Щоб ще більше згладити різницю Unity має шаблонні бібліотеки, які змінюють реалізацію своїх методів відповідно до цільової платформи.

Отже, використання фреймворків, принципів, та парадигм значно пришвидшило хід розробки та дозволило реалізувати складну, про те, злагоджену систему розробки програмного продукту по компонентам. Вся логіка незалежна та підрозділяється на різні компоненти, що підвищує стійкість до збоїв та у кілька разів пришвидшує процес пошуку помилок. Також це дозволяє змінювати код незалежно від іншої частини програми, що сприяє командній розробці.

**ПЕРЕЛІК ЛІТЕРАТУРИ**

1. Этапы создания компьютерной игры (Сайт Games is Art). gamesisart.ru.

2. Шаблон проектування Entity Component System https://habr.com/ru/post/343778/

3. Unity, ECS і все-все-все https://habr.com/ru/post/358108/

4. Керівництво Unity https://docs.unity3d.com/ru/current /Manual/index.html

5. Easy-code: http://easycode.ru/lesson/basic-cpp

6. Code-live: https://code-live.ru/tag/cpp-manual/

7. Proglib.io: https://proglib.io/p/cpp-lessons/.

8. Quizful: http://www.quizful.net/post/cpp\_fundamentals

9. Replace.org.ua: http://replace.org.ua/topic/228/

10. Mvblog: http://mvblog.ru/archives/861/

11. Cyberforum: http://www.cyberforum.ru/cpp-cli-winforms/thread1514933.html

12. Bestprog: https://www.bestprog.net/uk/2016/07/31/004

13. Бублик В.В. Б90 Об’єктно-орієнтоване програмування. / В.В. Бублик. – К.: ІТкнига, 2015. – 624 с.: іл. ISBN 978-966-97182-1-1

14. Грицюк Ю.І., Рак Т.Є. Г 85 Програмування мовою С++ : навчальний посібник. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 292 с. – Статистика: іл. 10, табл. 18, бібліогр. 31. ISВN 978-966-3466-85-9

15. Справочник по С# https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/

16. Unity в действии. Мультиплатформенная разработка на C#. Джозеф Хокинг. Питер. 2016. – 336 с.

17. C# 7.0 in a Nutshell: The Definitive Reference. Joseph & Ban Albahari. Вид-во O`REILLY.-1089с.-.ISBN-13: 978-1491987650

18. Герберт Шилдт. Полный справочник по С++ 4 видання. М.: Osborne, 2016.– 800 с.

19. ООП с примерами https://habr.com/post/87119/

20. C# Unity для начинающих https://itproger.com/course/unity-csharp

21. Unity С# tutorials https://catlikecoding.com/unity/tutorials/