СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра програмування та математики

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
спеціалізація „Інженерія програмного забезпечення”

на тему „Розробка програмних засобів мобільного додатку на базі Андроїд”

Виконав: студент групи ІПЗ-16д _____ О.П. Арєф’єв
(підпис) (ініціали і прізвище)

Керівник _____ Д.М. Марченко
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ В.О. Лифар
(підпис) (ініціали і прізвище)

Рецензент_____

Сєвєродонецьк - 2020

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра програмування та математики

Освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення”
(шифр і назва спеціальності)
спеціалізація „Інженерія програмного забезпечення”
(назва спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Лифар В.О.
“____” _____2020 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

_____ _ Арєф'єв Олександр Павлович_____
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка програмних засобів мобільного додатку на базі___ Андроїд._____
_____
_____

Керівник роботи ___професор, д.т.н.Марченко Дмитро Миколайович___,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “___”_____20__року №___

2. Строк подання студентом роботи_____20 травня 2020 р. _____
3. Вихідні дані до роботи___Об’єктом даної розробки є процес створення___ мобільного ігрового додатку._____
_____
_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Розробка ігрових мобільних додатків, відмінності Unity 3D та Unreal Engine, модель, архітектура та основні класи, створення та реалізація проекту._____
_____

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників)
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	30.03.20	
2	Укладання і погодження з керівником плану і етапів виконання роботи	05.04.20	
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	11.04.20	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	22.04.20	
5	Укладання та тестування програмного продукту	06.05.20	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	03.05.20	
7	Здача готової пояснювальної записки на кафедру	13.05.20	
8	Укладання доповіді і презентації	30.05.20	

Студент _____ О.П. Арєф’єв
(підпис) (ініціали і прізвище)

Керівник роботи _____ Д.М. Марченко
(підпис) (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ

дипломної роботи студента гр. ІПЗ-16д Арєф'єва О.П.

Науковий керівник

Доцент, к.т.н. _____ Марченко Д.М.

Оцінка наукового керівника: _____

Рецензент _____

ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:

_____

Голова ЕК

_____ ЛифарВ.О.

підпис

РЕФЕРАТ

Робота містить: 35 сторінок основного тексту, 25 сторінок додатків, 15 рисунків, 16 використаних джерел.

Метою випускної кваліфікаційної роботи є вивчення особливостей розробки мобільного ігрового додотку для операційної системи Android. Даний мобільний додтак спрямованний на розважальний ігровий контент.

В ході розробки ігрового мобільного додатку для операційної системи Android були проаналізовані відмінності Unity 3D та Unreal Engine при розробці мобільних додатків, визначені існуючі види мобільних Android-додатків, вивчені види чистої архітектури для мобільних додатків, ознайомилися з середою розробки, був створенний мобільний додаток для операційної системи Android, було проведено тесутваня мобільного додатку. Вироблено опис процесу розробки і тестування системи. Реалізовано, а також описаний користувальницький інтерфейс, зроблені знімки екранних форм програмного засобу. Продемонстровано результат виконаної роботи.

Система задовольняє всім вимогам, пред'явленим в технічному завданні.

Зміст

ВСТУП

Актуальність. Мобільні пристрої стали незмінним атрибутом в нашому житті, без мобільного телефону для сучасної людини вже немає комфорту. Телефон перестав бути тільки засобом спілкування між людьми, а став добрим способом розважитися. Мобільні пристрої нещодавно з'явилися, але вже міцно увійшли в нашу повсякденність. Зараз мобільні пристрої настільки функціональні, що користувач може їх використовувати як фотоапарат, музикальний плеєр, сканер штрих кодів, електронний гаманець, мобільні ігрові додатки тощо.

Об’єкт дослідження: Мобільний ігровий додаток для ОС Android.

Предмет дослідження: Мобільні додатки для ОС Android. Причини поширення цієї операційної системи такі, що Android з легкістю підтримує велику кількість пристроїв різних виробників.

Мета дослідження: Метою даної дипломної роботи є створення ігрового додатка для ОС Android.

Задачі дослідження:

1. Розробка графічного візуального інтерфейсу додатка;

2. Реалізація програмного модуля для відображення об'єктів різних категорій;

3. Тестування та рендеріг об’єктів.

1. АНАЛІТИЧНИЙ ОГЛЯД

1.1 Розробка додатків для мобільних пристороїв Android

Написання програм під Android це серйозний технологічний процесс, який вимагає ретельного планування і має свої особливості:

• Різноманітність мобільних гаджетів**.** При розробці додатків необхідно ураховувати, що користувачі мають різні за технічними показниками мобільні пристрої, на яких підтримуються різні версії операційної системи.

• Вразливість мобільних Android-додатків. Ця проблема може виникати переважно в бюджетних Android-смартфонах та через велику кількість неперевірених додатків у Google Play Market. [1]

1.2 Відмінності Unity 3D та Unreal Engine при розробці мобільних додатків

Ігровий движок - це основний набір програмних компонентів і візуальних інструментів, який дозволяє створювати і запускати інтерактивні додатки з графічним забезпеченням, яке обробляються в реальному часі.

Unity - це середовище для розробки комп'ютерних ігор, в якій об'єднані різні програмні засоби, що використовуються при створенні ПЗ - текстовий редактор, компілятор, відладчик і так далі. При цьому, завдяки зручності використання, Unity робить створення ігор максимально простим і комфортним, а мультиплатформеність движка дозволяє розробникам ігор охопити якомога більшу кількість ігрових платформ і операційних систем.

Движок Unity3D дає можливість розробляти ігри, не вимагаючи для цього особливих знань. Тут використовується компонентно-орієнтований підхід, в рамках якого розробник створює об'єкти (наприклад, головного героя) і до них додає різні компоненти (наприклад, візуальне відображення персонажа і способи управління ним). Завдяки зручному Drag & Drop інтерфейсу і функціональності графічного редактора движок дозволяє

малювати карти і розставляти об'єкти в реальному часі і відразу ж тестувати результат.

Наявність величезної бібліотеки Asset і плагінів, за допомогою яких можна значно прискорити процес розробки гри. Їх можна імпортувати і експортувати, додавати в гру цілі заготовки - рівні, ворогів і так далі. Багато Assetes надаються безкоштовно, інші пропонуються за невелику суму, і при бажанні можна створювати власний контент, публікувати його в Unity Asset Store і отримувати від цього прибуток.

Підтримка величезної кількості платформ, технологій, API. Створені на движку ігри можна легко перенести між ОС Windows, Linux, OS X, Android, iOS, на консолі сімейств PlayStation, Xbox, Nintendo, на VR- і AR-пристрої. Unity підтримує DirectX і OpenGL, працює з усіма сучасними ефектами рендеринга, включаючи новітню технологію трасування променів в реальному часі.

Фізика твердих тіл, ragdoll і тканин, система Level of Detail, колізії між об'єктами, складні анімації - все це можна реалізувати силами движка.

Unity доступний безкоштовно, що відкриває перед незалежними розробниками двері в ігрову індустрію. [3]

Для створювання складного проекту потрібні глибокі знання C#.

Повільність движка Unity 3D. Створення масштабних, складних сцен з великою кількістю компонентів може негативно вплинути на продуктивність гри, в результаті чого розробникам доведеться витратити додатковий час і ресурси на оптимізацію, а можливо - і видалення деяких елементів з проекту. Крім того, додатки, створені на Unity, досить «великовагових»: навіть найпростіша піксельна гра може займати кілька сотень мегабайт на ПК. Так, для жорсткого диска комп'ютерів це невеликий обсяг, але, якщо проект розробляється і для мобільних платформ, слід задуматися про оптимізацію його розміру.[3]

Unreal Engine - це сучасний ігровий движок на базі мови C ++.

Можливості Unreal Engine дозволяють створювати і редагувати елементи 3D анімації, спеціальні ефекти в кінофільмах і іграх, а також розробляти різні навчальні програми. Код програми на цьому движку працює на більшості сучасних платформ і операційних систем (Android, iOS, Linux, Mac OS, Microsoft Windows, PlayStation 4, PSP, Xbox One, PS Vita). Таким чином, однією з особливостей движка Unreal Engine є його універсальність.

До складу технології Unreal входить: графічний движок, движок забезпечення фізики, штучний інтелект, управління мережевою і файлової системами, а також потужний і багатофункціональний вбудований редактор UnrealEd.

Користувач за допомогою редактора може здійснювати маніпуляції з властивостями різних 3D-об'єктів, змінювати їх, програмуючи скриптові сценарії.

Всі зміни, що вносяться користувачем, такі як: розташування об'єктів, розподіл ефектів, поновлення текстур - видно через камеру редактора, що дозволяє користувачеві перевіряти можливі помилки побудови.

Редактор також дозволяє маніпулювати об'єктами, створеними за допомогою графічних пакетів, таких як 3D Max тощо.

До іншої, важливою особливості движка Unreal варто віднести вбудований редактор BluePrint: це інтуїтивна система для створення логіки 3D - середовища. Користувач створює деякі типи об'єктів, які він згодом помістить до свого проекту. Приклади роботи скриптів BluePrint (креслення) в ігровому додатку: призначення гравців з відмінними рисами, установка правил і проведення гри, призначення кнопок для управління персонажем і т.д. При цьому, за допомогою «креслень» логіку рівня або гри користувач може створити не написавши жодного рядка коду, що, безумовно, полегшить процес написання логіки тим людям, хто не сильний в програмуванні.

Обидва механізми надзвичайно конкурентоспроможні, коли справа доходить до розробки ігрових та неігрових додатків, 3D-візуалізації,

розробки додатків AR/VR реалістичного рендеринга і анімації. [3]

1.3 Управління пам'яттю в Android-додатках

Оперативна пам'ять (RAM) є дуже важливим ресурсом в середовищі розробки програмного забезпечення, і ще цінніша на операційній системі мобільних пристроїв, де пам'ять часто може бути обмежена.

Занадто мало RAM завжди буде проблемою. Якщо в системі недостатньо оперативної пам'яті для роботи, тоді все починає ставати проблемою. Програми, що працюють у фоновому режимі, закриваються передчасно (або коли Ви цього не хочете).

Від даного виду пам'яті залежить складність і кількість додатків, що одночасно запускаються на девайсе. Оперативна пам’ять впливає на швидкість запуску програм та ігор і багатозадачність процесів. У тому випадку, якщо обсягу оперативної пам'яті досить, тоді з легкістю можна відкривати 10-15 вкладок браузера, заходить в додатки, розмовляти і т.д.

З огляду на той факт, що обсяг оперативної пам'яті сильно впливає на продуктивність мобільних пристроїв, в разі її нестачі важкі додатки можуть просто не завантажиться, реакція на команди буде сповільненою. На ринку представлений величезний вибір апаратів з різним об'ємом оперативної пам'яті, природно, чим її більше, тим вище цінник.

При розробці мобільних додатків важно враховувати той факт, що додаток треба оптимізувати під більшість мобільних пристроїв, де може бути недостатньо оперативної пам’яті.

1.4 Використання інтерфейсу на мобільних пристроях

Треба розуміти, що мобільні пристрої відрізняються від персональних комп’ютерів. Тому, важливо розуміти як користувач буде користуватися мобільним додатком – чи буде він простий та зручний.

При розробці мобільного додатку розробник повинен враховувати наступні речі:

- Унікальність. Додаток має відрізнятися від інших додатків та мати свої переваги.
- Привабливість. Додаток має бути цікавим, з привабливим інтерфейсом, так, щоб користувачу було приємно їм користуватися.
- Надійність. Додаток повинен бути розроблений з урахуванням мобільного пристрою користувача, щоб це не заважало роботі смартфона.

Різні додатки вимагають різних підходів, дизайнерських рішень і технік. У використанні мобільних пристроїв з сенсорним екраном є декілька рекомендацій:

- Відклик. Відклик та швидкість роботи – не одне й те саме. Але користувач повинен розуміти, що його запит оброблюється та треба зачекати деякий час. Виконання деяких операцій потребує часу.
- Великі пальці. З появою тач-скрин інтерфейсів не варто забувати, що додатки потрібно розробляти не під вказівний, а під великий палець. Користувачеві більш зручно використовувати для роботи в додатку два великих пальця.
- Цільові об’єкти. Розробник повинен продумати як розмістити об’єкти на екрані, щоб користувачеві було зручно використовувати та переміщуватися у додатку.
- Елементи управління. Елементи управління краще помістити у нижній частині екрану, щоб вони візуально не заважали при використанні додатку.
- Прокрутка. Зручний метод для переміщення по мобільним додаткам.[5]

1.5 C#

C # - мова програмування, що поєднує об'єктно-орієнтовані і контекстно-орієнтовані концепції. Розроблена в 1998 – 2001 роках групою

інженерів під керівництвом Андерсом Гейлсбергом,в компанії Microsoft. Мова розробки додатків для платформи Microsoft .NET Framework.

Мова C # відноситься до сім'ї мов з C-подібним синтаксисом, з них його синтаксис найбільш близький до C ++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, коментарі у форматі XML.

1.6 Технічне завдання на розробку додатка

В результаті виконання даного дипломного проекту повинен бути створений ігровий додаток для операційної системи Android. Додаток повинен бути зручним для користувача, простим в управлінні, інтуїтивно зрозумілим. Додаток призначається для функціонування в операційній системі Android, пізніше і на інші операційні системи.

При розробці ігрового додатка для операційної системи Android потрібно вирішити наступні завдання:

- Проаналізувати відмінності Unity 3D та Unreal Engine при розробці мобільних додатків
- Визначити існуючі види мобільних Android-додатків
- Вивчити види чистої архітектури для мобільних додатків
- Ознайомитися з середою розробки
- Реалізація проекту для операційної системи Android
- Тесутваня мобільного додатку

2. ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ГАЛУЗІ. ВИДИ ТА АРХІТЕКРУТА. ЖИТТЄВИЙ ЦИКЛ СЕРВІСІВ. ПРОБЛЕМИ ТА РИЗИКИ

2.1 Модель

Модель – це відтворення чи відображення об'єкту, задуму (конструкцій), опису чи розрахунків, що відображає, імітує, відтворює принципи організації або функціонування, певні властивості, ознаки та характеристики об'єкта дослідження чи відтворення (оригіналу). Таке визначення дає популярна інтернет-енциклопедія Вікіпедія. Модель – зразок, образ об’єкту чи системи, замінник оригіналу, що представляє суттєві властивості оригіналу. Виходячи з наведених визначень, модель розробки програмного забезпечення – це представлення програмного продукту у формі, відмінної від реальної, з детальним описом етапів виконання процесів для досягнення мети (отримання програмного продукту).

Початок будь-якого проекту з розробки програмного забезпечення починається з двох питань:

1) Для кого?

Тобто, для кого створюється це програмне забезпечення, яка цільова група користувачів та які їх основні вимоги.

2) Яким чином?

Тобто, як будуватиметься процес розробки цього програмного забезпечення.

2.2 Універсальна модель (модель, яка застосовується)

Універсальний процес (*Unified Process, UP*) – модель аналізу, проектування й розробки програмного забезпечення. Основні характеристики цієї моделі наступні:

– сценарії використання як основа проекту;

– пріоритет архітектури;

– ітераційний і інкрементний (ітерація зі збільшенням) процес розробки;

– прогнозування ризиків;

– об’єктно- і сервісно-орієнтоване проектування;

– активне нагромадження й повторне використання об’єктно-орієнтованих шаблонів і об'єктів.

Ітеративна та інкрементна розробка – це основа циклічного процесу розробки програмного забезпечення, для зменшення слабких сторін, що з’являються при реалізації водоспадної моделі.

Універсальний процес розробки має п’ять основних етапів, що постійно виконуються на чотирьох фазах моделі, поки не буде отриманий програмний продукт з визначеними якостями. Виконання етапів – ітерація. Кожна ітерація – проміжний випуск програмного продукту.

Етапи:

– визначення всіх можливих вимог до проекту;

– прикладне моделювання на основі зібраних вимог;

– створення архітектури на основі об’єктно-орієнтованого підходу;

– реалізація прототипів (на фазах ітерацій) та програмного продукту (наприкінці реалізації моделі);

– перевірка зробленої роботи та тестування програмного продукту.

Слід зазначити, що на кожному етапі виконується чітка послідовність дій. наприклад, на етапі реалізації виконуються наступні кроки**:**

– реалізації прототипу архітектури;

– реалізації компонентів (класів і об'єктів);

– тестування компонентів;

– інтеграції компонентів;

– складання додатка:

– створення тестів на основі схем використання;

– перевірки архітектури;

– планування наступного складання;

– переходу до наступної ітерації.

Фази схожі до фаз спіральної моделі та мають визначені цілі:

– «Дослідження» націлене на створення моделі предметної області;

– ітерації фази «Пророоблення» мають на меті створення базової архітектури;

– ітерації фази «Створення» націлені на створення продукту шляхом послідовного випуску версій;

– фаза «Готовність» призначена для перевірки готовності продукту до експлуатації.

На останній фазі передбачено не лише розгортання програмного продукту, усунення проблем, інформування замовника про необхідні зміни та доопрацювання продукту, а й підготовку всієї проектної документації та настанов для користувача.

За цією моделлю також передбачено використання компромісного трикутника за графіками виконання, ресурсами та характеристиками продукту. Компромісний трикутник реалізується на етапі прикладного моделювання. [6]

2.3 Існуючі види мобільних додатків

Мобільні додатки - це програми, розроблені спеціально для смартфонів, планшетних комп'ютерів і інших мобільних пристроїв. Вони дозволяють вирішувати різні завдання: від мобільної картографії до розваг та інші функції.

Існує три типи мобільних додатків:

- Нативні. Нативні («рідні») додатки - це додатки, розроблені на мовах програмування певної платформи (Android, Apple). Це дозволяє використовувати всі можливості мобільного пристрою - адресну книгу користувача, геолокацію, камеру, або датчики прискорення. Робота цих

програм не вимагає підключення до Інтернету. Багато з них не можуть працювати функціонально, перебуваючи довгий час в режимі «оффлайн».

- Веб-додатки. Веб-додатки - це додатки, що використовують веб-технології для роботи на мобільному пристрої. Один додаток (з однаковим вихідним кодом) може працювати на різних пристроях і платформах. При запуску програми в браузері, вони запускаються незалежно від моделі телефону. На відміну від нативних додатків веб-додатки не треба скачувати на мобільний пристрій. Веб-додатку можуть бути знайдені та використані просто через пошук в браузері. Веб-додатки можуть оновлюватися у реальному часі.
- Гібридні додатки. Гібридні додатки - це поєднання між нативними додатками та веб-додатками. Ці додатки дозволяють розробляти кросплатформені програми для використання веб-технологій (такі як HTML, JavaScript і CSS), і в той же час мають доступ до функцій мобільного пристрою. Гібридні додатоки - це нативні додатки з вбудованим HTML. Веб-компоненти можуть бути завантажені з Інтернету або вже упаковані в додатки. Дані програми дозволяють поєднувати переваги нативних додатків з «довговічністю» або технологічної актуальністю, забезпечується останніми веб-технологіями.[7]

В наш час розробники віддають перевагу саме гібридним мобільним додаткам.

2.4 Архітектура додатка та його основні класи

Розробка ОС Android була розпочата в 2003 молодою компанією Android Inc. У 2005 році ця компанія була куплена Google. Головні особливості архітектури Android були визначені цей період. Архітектурні концепції та фінансові ресурси Google зробили вирішальний вплив на архітектуру Android.

Операційна система Android має три дуже різних і сильно відокремлених один від одного рівня:

- В основі лежить модифікована і урізана версія Linux.
- Над рівнем Linux знаходиться рівень інфраструктури додатки.
- Рівень написаних в Google Android-додатків. Взагалі кажучи, вони є розширенням рівня інфраструктури, оскільки розробник може використовувати ці додатки або їх частини як будівельні блоки для власних розробок.

Для того, щоб один і той же додаток міг працювати на різному апаратному забезпеченні, компанія Google використовувала контейнер-орієнтовану архітектуру (container-based architecture). У такій архітектурі двійковий код виконується програмним контейнером і ізолюється від деталей конкретного апаратного забезпечення. Приклади всім знайомі - Java і C #. В обох мовах двійкового коду не залежить від специфіки апаратного забезпечення і виконується віртуальною машиною.

Архітектура Android є фреймворк-орієнтованої (framework-based), на противагу вільної (free-style) архітектурі. В противагу цьому фреймворк-орієнтовані додатки ґрунтуються на існуючому фреймворку. Їх розробка зводиться до розширення деяких класів або реалізації інтерфейсів, наданих фрейморком. Такий додаток не може бути запущено поза фреймворка або без нього. Прикладом можуть бути веб-додатки на Java, в яких розробники реалізують інтерфейс Servlet або розширюють одну з його реалізацій, або додатки Eclipse RCP, в якому розробник розширює один з класів Editor або View.

У плані обробки взаємодії між призначеним для користувача інтерфейсом і його логікою Android слід архітектурному шаблоном «Model-View-ViewModel» (MVVM). Архітектура MVVM була створена з метою поділу праці дизайнера і програміста.

Різні частини Android-додатків можуть викликати один одного і

взаємодіяти між собою тільки формально. Фреймворк Android використовує кілька шаблонів взаємодії:

- Обмін повідомленнями за допомогою класу Intent спостерігач з використанням класів Intent і BroadcastReceiver.
- Пізнє зв'язування з подальшим викликом методу використовується для доступу до контент-провайдерам (ContentProviders) і локальним (внуріпроцессним) сервісів (Services).
- Пізнє зв'язування і міжпроцесова взаємодія (Inter-process Procedure Communication, IPC) для виклику сервісів (AIDL).

Android-додаток складається з:

- Java – класів, що є підкласами основних класів з Android SDK (View, Activity, ContentProvider, Service, BroadcastReciever, Intent) і Java-класів, у яких немає батьків в Android SDK.[7]
- Маніфесту додатки.
- Ресурсів на зразок рядків, зображень тощо.
- Файлів.

2.5 Java класи

На рисунку 2.1 зображено ієрархію основних класів Android SDK.

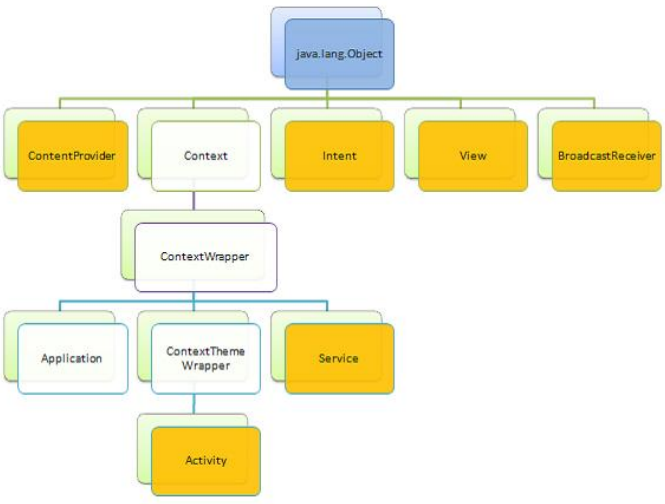

Рисунок 2.1. Ієрархія класів Android SDK

Жовтим кольором виділені класи, якими розробники користуються частіше. Інші класи також важливі, але цими класами користуються рідше.

Клас View – базовий клас для всіх віджетів призначених для користувача інтерфейсу (GUI widgets). Інтерфейс Android додатку являє собою дерево примірників спадкоємців цього класу.

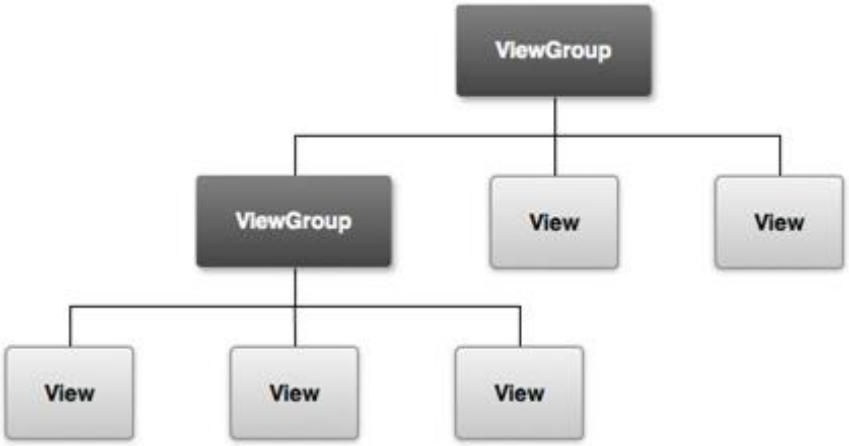

Рисунок 2.2. Ієрархія компонентів, що визначає компоновку інтерфейсу користувача

Класс Activity та його підкласи містять логіку, що лежить за призначеним для користувача інтерфейсу. Цей клас відповідає ViewModel в архітектурному шалоні «Model-View-ViewModel» (MVVM). Activity має життєвий цикл, а саме:

- Активно і виконується – призначений для користувача інтерфейс знаходиться на передньому плані (вершина стека активності)
- Припинено – якщо інтерфейс користувача втратив фокус, але його можна побачити.
- Завершено – якщо інтерфейс користувача невидимий.

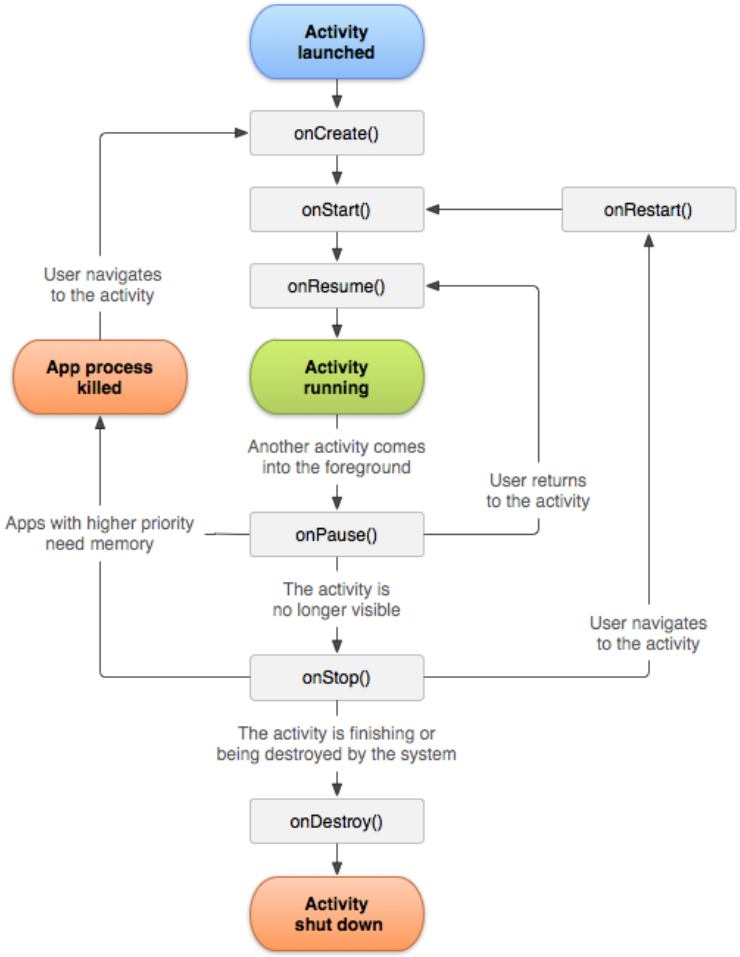

Рисунок 2.3. Життєвий цикл активності

Клас ContentProvider та його підкласи являє собою model в архітектурі MVVM.

Клас Service та його підкласи. Вони були створені для рішення логічної проблеми. Даний клас може виконувати завдання, коли процес, в якому вони працюють знаходяться не на передньому плані. Service також має життєвий цикл, це означає, що він може інстанціювати і бути запущеним Android – додатком за деякими умовами.

Клас BroadcastReceiver та його підкласи являють собою «передплатника» в механізмі взаємодій видавець/передплатник, реалізованому в архітектурі Android.[8]

2.6 Маніфест Android-додатків

Маніфест Android – це дуже важлива частина Android-додатка. Він являє собою XML файл і виконує декілька функцій:

- Визначає ім’я Java пакета програми
- Описує компоненти додатку – Activity, Service, BroadcastReceiver, ContentProvider
- Зумовлює процеси, які будуть містити компоненти програми.
- Перераховує бібліотеки, з якими додатки повинні бути зв’язані.

їм вистачить.

2.7 Ресурси

Додатки містять у собі різні ресурси: зображення, ряди, матеріали, аудіо, відео, анимації, компонування елементів призначені для користувача інтерфейсу (layout – слой). Відділення коду від ресурсів передбачає використовування ресурсів для різних конфігурацій пристроїв: дозвіл екрану, мова тощо.

Кожен з цих видів ресурсів знаходиться в спеціальній папці resources/.

Resources:

- animator - файли, які містять у собі готову записану анімацію;
- animation - файли, які визначають анімацію;
- color - файли, які визначають кольор матеріалу;
- layout - файли, які визначають елементи призначені для користувача інтерфейсу;
- menu - файли, які містять меню програми;
- values - файли, які містять значення рядків, матералів, чисел;

Ресурси, які є в розглянутих піддиректоріях, є ресурсами за замовчуванням, щоб визначити залежні від конфігурації альтернативи для безлічі ресурсів:

1. Створена директорія resources/, присвоює діній директорії назву в такій формі: ім'я_ресурсу-специфікатор_конфігурації, де

- ім'я_ресурсу - назва директорії, відповідного ресурсу за умовчанням;
- конфігурації - назва, що визначає конфігурацію, для якої використовуються дані ресурси.

Ресурси зберігаються каталог, файл ресурсів називатися так само, як відповідний файл ресурсів за замовчуванням.

2.8 Файли

Android-додатки використовують декілька різних типів файлів:

- Файли загального призначення
- Файли Бази Даних
- Файли Opaque Binary Blob (зашифровані файли)
- Кеш файли

2.9 Проблеми, ризики, які можуть виникнути

Важко і навіть неможливо. Багато хто після декількох (або навіть одного) успішного проекту наповнюються упевненістю, що написати ігровий проект їм під силу. Насправді, ігри - це одне з найскладніших напрямків розробки. І чим «серйозніше» гра тим проект складніше. В процесі створення гри розробник може зіткнутися з нерозв'язних проблемами, що вбивають на кореню ентузіазм навіть у самих упертих.

Відраза до ігор. Згодом у кожного досвідченого розробника ігор розвивається відраза до ігор. Спочатку просто вони стає менш цікавими. Потім починаєш помічати, що вони зовсім не цікаві, а цікаво тільки, як вони

працюють. Чим далі, тим гірше.

Конкуренція і якість продукту. Іграми займаються багато студій і незалежні розробники. Існує своєрідний «рівень входження» в цей бізнес. Зараз не можна зробити успішну гру, намальовану аквареллю (так, такі ігри зустрічалися на початку 2000-х). Вона просто не витримає конкуренції. Відповідно, аби що не зробиш. Тут ховається важливий психологічний момент - початківець розробник змушений робити хороший продукт, інакше він буде відчувати постійне відчуття страху неуспішності свого продукту.

Час і ресурси. І найпоширеніша помилка - це те, що ресурсів (час, гроші, знання) [10]

3. СТВОРЕННЯ ТА ПРОЕКТУВАННЯ ІГРОВОГО ДОДАТКУ ДЛЯ операційної системи ANDROID

3.1 Середовище розробки

Під час створення гри відбулося ознайомлення з платформою Unity 3D та її інструментами. Це можна побачити нижче на зображені 3.1.

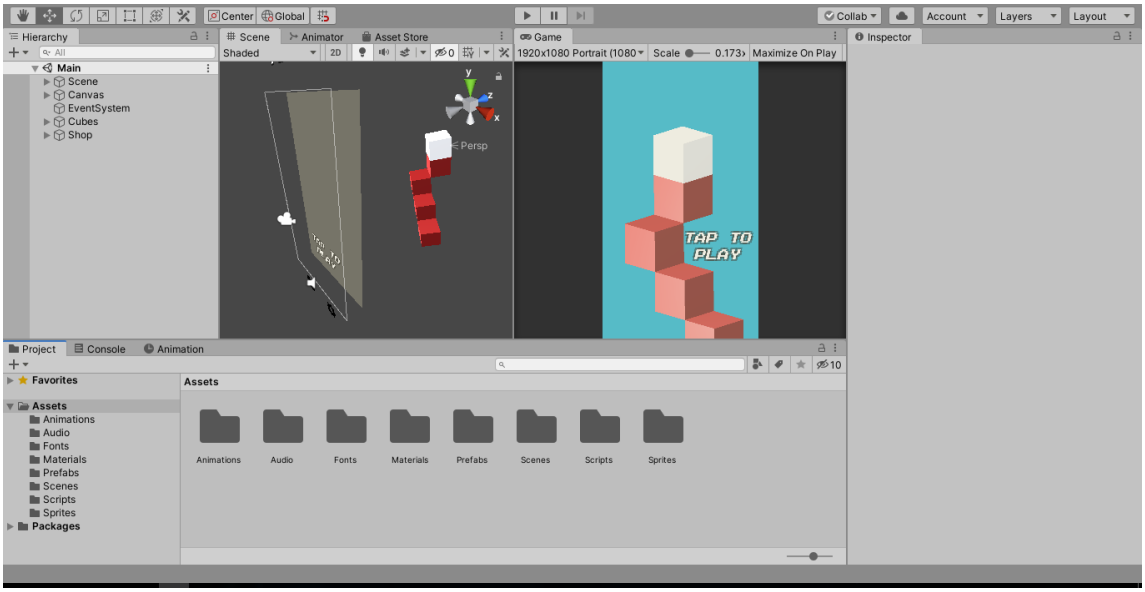

Рисунок 3.1. Середовище розробки

На цій самій платформі локацією є послідовно повторювана фонова заставка в програмі. З кожним кроком гравець набирає все більший рекорд. Коли гравець програє лічильник очок обнуляється, і нарахування очок після респауна, починається знову, але попередній рекорд зберігається, що дозволяє гравцю змагатися з самим собою та іншими гравцями.

На початку гри відбувається ініціалізація інтерфейсу і всіх змінних.

3.2 Реалізація та створення проекту

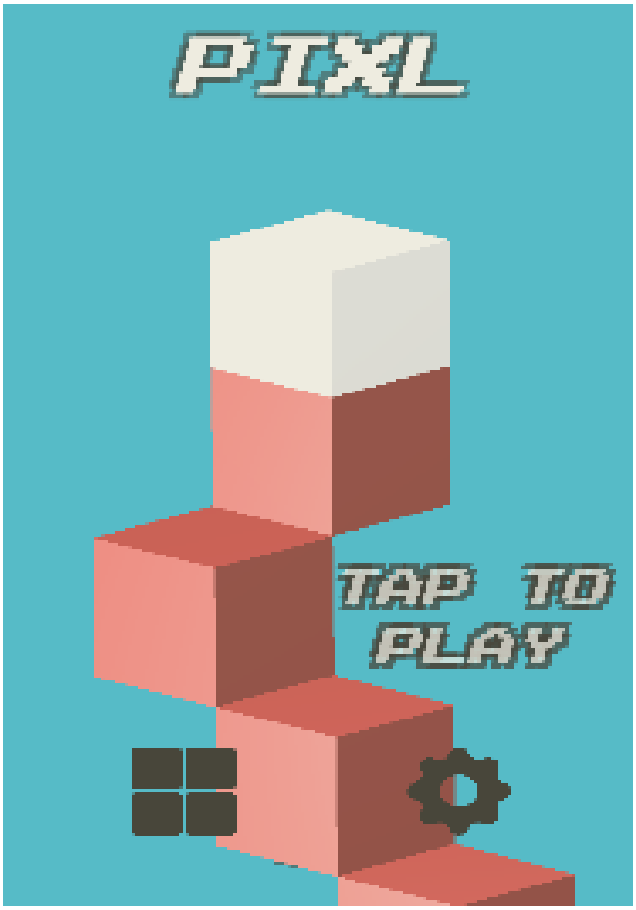

Рисунок 3.2. Головне меню гри

На зображені 3.2 зображено головне меню програми. Тобто графічний інтерфейс користувача, де гравцю дозволяється взаємодіяти з грою та її налаштуваннями.

Щоб завантажити гру був використаний метод, зміст методу Application.LoadScene () - очистити поточну сцену і інстанціювати наступну сцену програми. Іноді потрібно, щоб головний ігровий об'єкт з началальної сцени був перенесений на іншу сцену (наприклад, музика або головний об’єкт програми, як у даній програмі).

Щоб завантажити наступну сцену програми, тобто просто почати гру, потрібно натиснути на будь-яке місто головного меню, окрім кнопок «магазин» та «налаштування». Після цього, завдяки використаному методу та спеціальних анімацій для даної програми запускається ігровий рівень.

Це зображено нижче на рисунку 3.3.

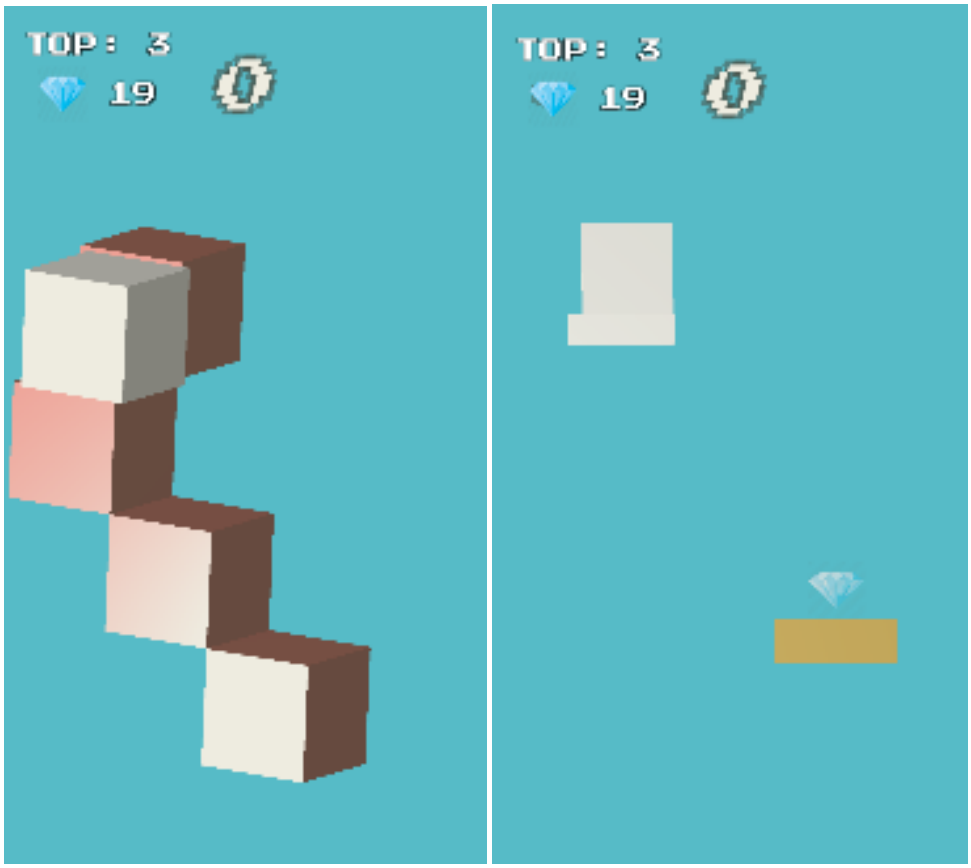

Рисунок 3.3. Запуск та початок гри

На головній сторінці можна побачити фон, на якому розташовані об’єкти гри. В данному Android-додатку фон являє собою нерухомий об’єкт.

Гра являється 3D, тому всі об’єкти повинні знаходиться на точках (0,0,0). В даному мобільному додатку присутній Паралакс ефект. Цей ефект можна зробити двома способами:

• Користувач гри та камера рухаються, всі інші об’єкти нерухомі;

• Користувач гри та камера нерухомі, всі інші об’єкти рухаються.

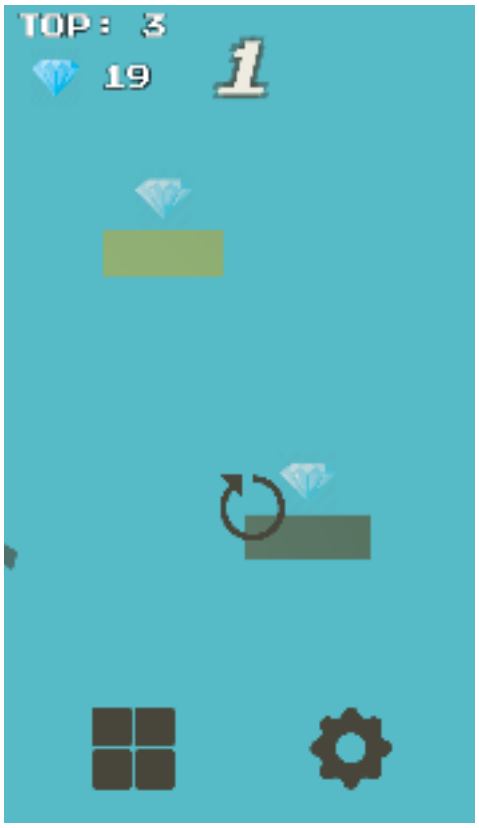

Рисунок 3.4. Меню паузи або меню програшу

Якщо у користувача відбувається програш рівня, тоді гра зупиняється та вспливає кнопка, яка дозволяє почату гру з початку або вийти назад до головного меню. Можна побачити вище на рисунку 3.4.

Ця функція реалізована теж завдяки методу Application.LoadScene (), дозволяє переключатися між сценами. З даного меню також можна скористатися кнопками «магазин» та «налаштування».

Коли гравець доходить до певної точки гри, гра починає становитися складніше. Наприклад, блоки починають рухатися, с кожним разом все швидше, і ракурс камери починає змінюватися у різні боки, щоб гравцю було складніше йти далі.

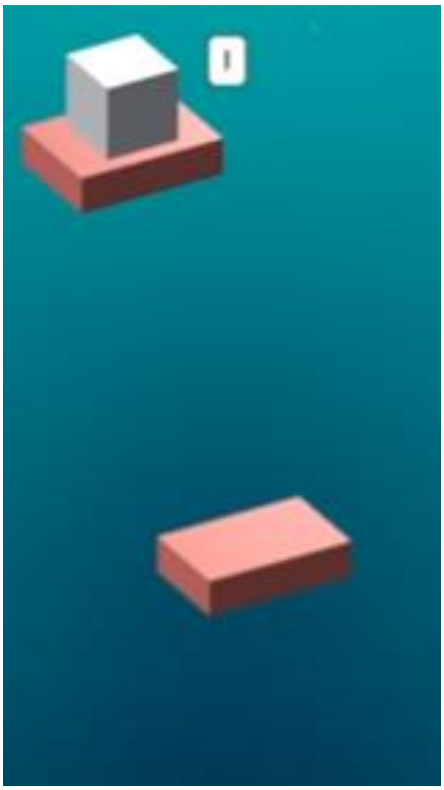

Рисунок 3.5. Ускладнення рівня

```
if (CubeJump.count_blocks > 0)
    {
    if (CubeJump.count_blocks % 1 == 0 && !hard)
    {
        print ("Hard");
        Camera.main.GetComponent <Animation> ().Play ("Harder");
        detectClick.transform.position = new Vector3 (35.87f, -11.44f, -1.33f);
        detectClick.transform.eulerAngles = new Vector3 (19.459f, -63.667f, 0f);
        hard = true;
    }
    else if (CubeJump.count_blocks % 2 == 0 && hard)
    {
```

```
hard = false;
print("Easy");
detectClick.transform.position = new Vector3 (26.89502f, -15.76862f, 0.16f);
detectClick.transform.eulerAngles = new Vector3 (0f, 0f, 0f);
Camera.main.GetComponent <Animation> ().Play ("Easy");
}
```

Найголовніша задача користувача в даній грі, це набрати як можна більший рекорд, завдяки проходженню далі і далі. Тому в програмі реалізовано запис набраного рекорду та запис зібраних алмазів, які потрібно збирати під час ігрового процесу. Цей запис відбувається два рази. Перший запис, це головний рекорд, тобто найбільша кількість очок за весь час у грі. Другий запис, це запис, який відбувається на даний момент гри. Це видно нижче на рисунку 3.6.

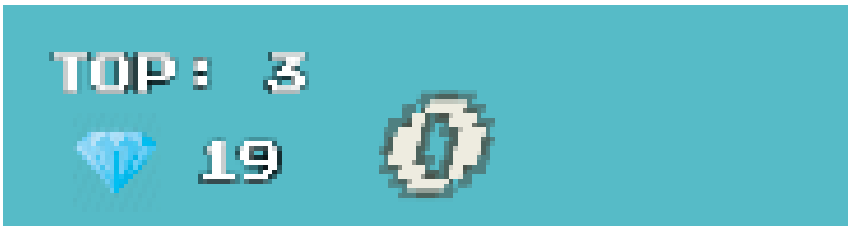

Рисунок 3.6. Запис ігрових даних

Реалізація даної функції:

```
void Start()
   {
      record.text = "Top: " + PlayerPrefs.GetInt ("Record").ToString ();
      txt = GetComponent <Text> ();
       CubeJump.count_blocks = 0;
   }
```

```
void Update()
{
  if (txt.text == "0")
  gameStart = true;
   if (gameStart)
   {
        txt.text = CubeJump.count_blocks.ToString ();
        if (PlayerPrefs.GetInt ("Record") < CubeJump.count_blocks)
        PlayerPrefs.SetInt ("Record", CubeJump.count_blocks);
        record.text = "Top: " + PlayerPrefs.GetInt ("Record").ToString
();
        }
}
```

Магазин. Для чого потрібно збирати алмази?

Алмази – це ігрова валюта у даній грі, завдяки їй гравець може здійснювати покупки для свого ігрового об’єкту, тобто скін. Зображено нижче на рисунку 3.7.

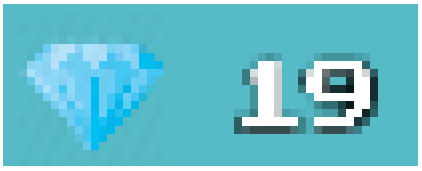

Рисунок 3.7. Запис зібраних діамантів

Запис зібраних алмазів також записується завдяки аналогічному запису рекорду гравця.

```
public Text diamonds;
private Text txt;

  void Start()
```

```
{
    txt = GetComponent <Text> ();
        txt.text = PlayerPrefs.GetInt ("Diamonds").ToString ();
}

void OnTriggerEnter (Collider other)
    {
        if (other.tag == "Diamond")
        {
        Destroy (other.gameObject);
        PlayerPrefs.SetInt ("Diamonds", PlayerPrefs.GetInt
("Diamonds") + 1);
            diamonds.text = PlayerPrefs.GetInt
("Diamonds").ToString ();
        }
    }
```

Нижче на рисунку 3.8 зображений ігровий магазин, де знаходяться різноманітні скіни для ігрового об’єкту. Тобто завдяки цим скінам можна змінювати зовнішній вигляд ігрового об’єкту. Кожний скін коштує певну кількість алмазів, від дешевих до дорожчих.

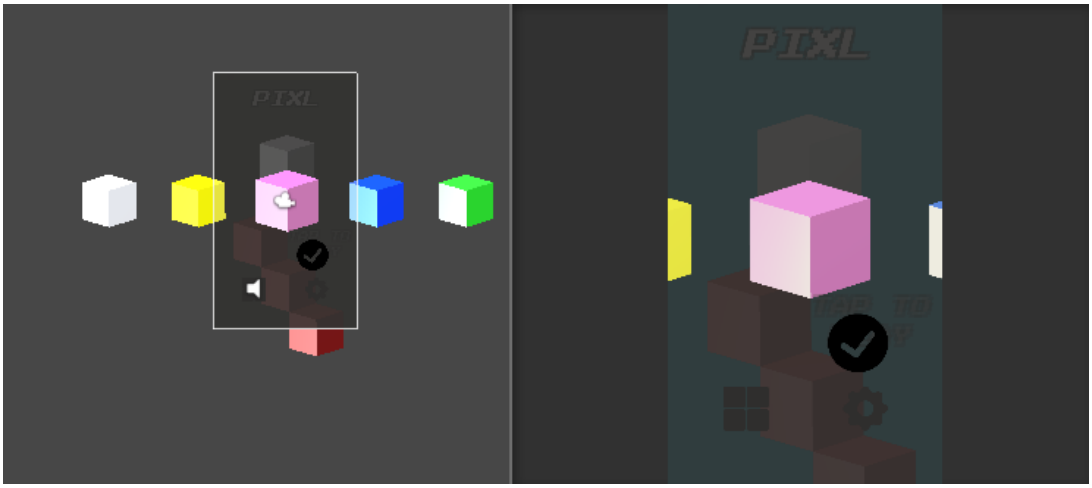

Рисунок 3.8. Магазин

Перейти до магазину можна завдяки кнопкам, які знаходять на екрані головного меню. Або у меню паузи. Зображено на рисунку 3.9.

Рисунок 3.9. Кнопки «Магазин» і «Налаштування»

Завдяки кнопці «Налаштування» можна налаштовувати програму як потрібно користувачу (наприклад, вимкнути або увімкнути звук у програмі).

Створення спрайтів для гри відбувалося в програмі Adobe Illustrator CC.

Створення спрайтів може бути різним, все залежить від тематики та ідеї розробника програми.

Приклад можна побачити на зображені нижче на рисунку. 3.10.

Рисунок 3.10. Приклад створення спрайтів

Створення анімації для гри відбувалося в програмі Dragon Bones.

Завдяки анимації можна зробити максимально красиву та плавну гру, це робіть гру більш цікавою та приятну для користування. Анімація створюється завядки готовому спрайту, спрайт розбивається на частини, і кожна частина налаштувається під ту анімацію, яка нам потрібна(наприклад, анімація стрибку).

Приклад можна побачити на зображені нижче на рисунку 3.11.

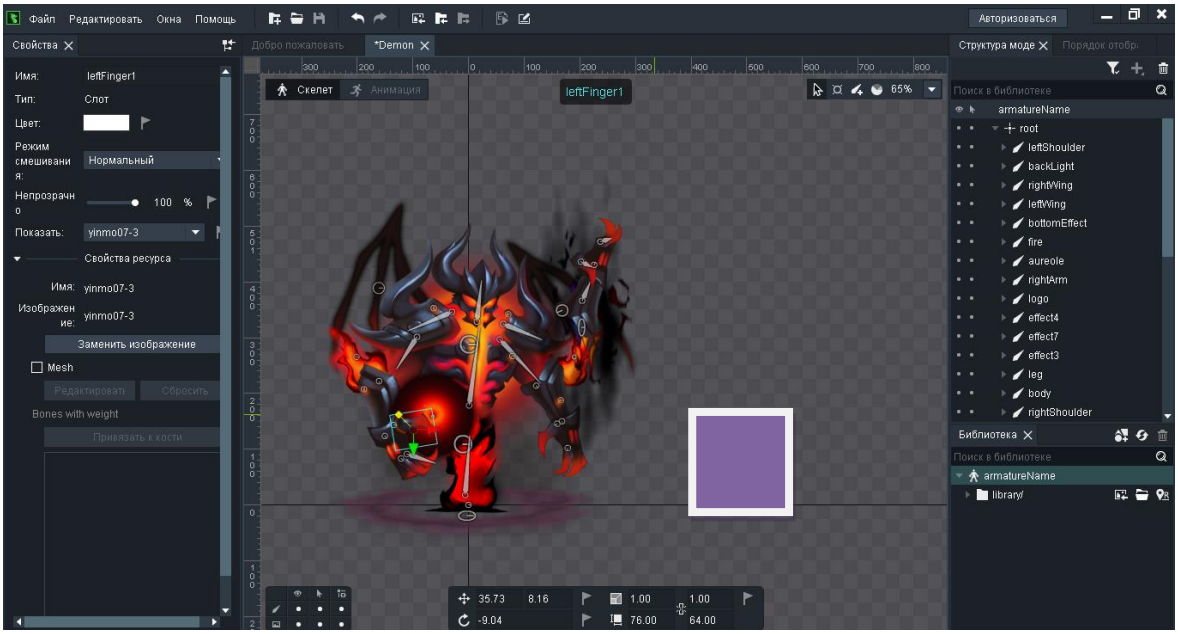

Рисунок 3.11. Приклад створення анімації

3.3 Тестування

При розробці мобільного додатку для Android важливо проводити тестування на реальному приладі. Розробник може використовувати будь-який Android-прилад як середовище для запуску, налагодження та тестування додатків.

Для ефективного тестування мобільних додатків можливо використовувати різноманітні інструменти, які удосконалюють та пришвидшують процес тестування.

Існують такі інструменти:

- Емулятори. Це програма, які копіюють поведінку інших приладів. Емулятори допомагають протестуватися складні сценарії, які не рекомендують проводити на справжніх мобільних приладах.

- Сервіси для бєта-тестування. Бєта-тестування – це активна робота з майже готовою версією мобільного додатку для знаходження можливих помилок та їх подальшого усунення.

- Сбір статистики. Статистична інформація про те, як користувачі працюють з додатком буде не зайвою.

Тестування Android-додатків включає в себе такі етапи:

- Реєстрація та вхід до системи. Зручність та легкість при вході при реєстрації та входу до системи мобільного додатку.

- Пункти меню. Особливо це стосується мобільних приладів з невеликим екраном. Пункти меню повинні бути візуально доступні і зручні при натисканні на кнопки меню пальцем.

- Ключові функції. Функції, які пов’язані з набором тексту, скролінгом, кнопкою «Назад» повинне обов’язково перевірятися в ході тестування.

- Переривання. Необхідно протестувати як веде себе додаток, коли

батарея на повному, середньому, низькому заряді. Наскільки робота інших мобільних додатків (найчастіше стандартних) може вплинути на роботу Android-додатку.

- Автоповорот. Деякі мобільні додатки можливо використовувати не тільки в портретній орієнтації, але й в альбомній. Тому треба перевірити, щоб функціональність мобільного додатку не змінювалася при переході з портретного режиму в альбомний.
- Вплив мобільного додатку на роботу приладу. Треба перевірити як працює мобільний додаток в звичайному режимі. Для тестування необхідно запустити мобільний додаток і використовувати його від 6 - 12 годин. Потрібно перевіряти кожен час рівень заряду батареї на мобільному приладі, якщо заряд батареї швидко падає, це указує на недоліки в мобільному додатку.

ВИСНОВКИ

В результаті виконання даного дипломного проекту на основі унікального сценарію та текстур створений ігровий додаток для операційної системи Android. Додаток зручний в використанні, простий в управлінні, інтуїтивно зрозумілий. Додаток призначений для функціонування під управлінням операційної системи Android.

В ході розробки ігрового мобільного додатку для операційної системи Android були вирішені такі задачі:

- Було проаналізовані відмінності Unity 3D та Unreal Engine при розробці мобільних додатків
- Були визначені існуючі види мобільних Android-додатків
- Були вивчені види чистої архітектури для мобільних додатків
- Ознайомилися з середою розробки
- Було реалізовано проект для операційної системи Android
- Було проведено тесутваня мобільного додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Джозеф Хокинг - Unity в действии ;
2. Зиборов, В.В. Visual C# 2012 на примерах / В.В. Зиборов. - М.: БХВ-Петербург, 2013. - 480 с.
3. Алон Торн - Искусство создания сценариев в Unity;
4. Стиллмен Э., Грин Дж. - Изучаем C#. Включая C# .NET 4.0 и Visual Studio 2010. 2-е издание (Бестселлеры O'Reilly) – 2012;
5. Касаткин, А. И. Профессиональное программирование на языке си. Управление ресурсами / А.И. Касаткин. - М.: Высшая школа, 2012. - 432 с.
6. Фримен Эр., Фримен Эл., Сьерра К., Бейтс Б. - Паттерны проектирования – 2011;
7. Агуров, Павел C#. Сборник рецептов / Павел Агуров. - М.: "БХВ-Петербург", 2012. - 432 с.
8. Албахари, Джозеф C# 3.0. Справочник / Джозеф Албахари , Бен Албахари. - М.: БХВ-Петербург, 2013. - 944 с.
9. Джонатан Линовес – Віртуальна реальність Unity 3D;
10. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. - 266 с.
11. Бишоп, Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2013. - 472 с.
12. Зиборов, Виктор Visual C# 2010 на примерах / Виктор Зиборов. - М.: "БХВ-Петербург", 2011. - 432 с.
13. Ішкова, Э. А. Самоучитель C#. Начала программирования / Э.А. Ишкова. - М.: Наука и техника, 2013. - 496 с.
14. Марченко, А. Л. Основы программирования на C# 2.0 / А.Л. Марченко. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2011. - 552 с.
15. Лотка, Рокфорд C# и CSLA .NET Framework. Разработка бизнес-объектов / Рокфорд Лотка. - М.: Вильямс, 2010. - 816 с.

16. Мак-Дональд, Мэтью Silverlight 5 с примерами на C# для профессионалов / Мэтью Мак-Дональд. - М.: Вильямс, 2013. - 848 с.

ДОДАТКИ

ПРОГРАМНИЙ КОД

// Анимация кнопок/переходы на другие страницы/функции кнопок

```
public class Buttons : MonoBehaviour
{
    public GameObject shopBG;
    public Sprite mus_on, mus_off;
    public float bigger = 0.75f, lower = 0.5f;
    void Start ()
    {
        if (gameObject.name == "Music")
        {
            if (PlayerPrefs.GetString ("Music") == "off")
            {
                GetComponent <Image> ().sprite = mus_off;
                Camera.main.GetComponent <AudioListener> ().enabled = false; // отключение музыки
            }
        }
    }
    void OnMouseDown()
    {
        transform.localScale = new Vector3 (bigger, bigger, bigger);

    }
    void OnMouseUp()
    {
        transform.localScale = new Vector3 (lower, lower, lower);
```

```
        }
        void OnMouseUpAsButton ()
        {
            switch (gameObject.name)
            {
                case "Restart":
                    SceneManager.LoadScene ("Main");
                    break;
                case "Music":

                    if (PlayerPrefs.GetString ("Music") == "off")
                    {
                        GetComponent <Image> ().sprite = mus_on;
// музыка включена в настройках
                        PlayerPrefs.SetString ("Music", "on");
                        Camera.main.GetComponent
<AudioListener> ().enabled = true; // вкучаем музыку
                    }
        else
        {
        GetComponent <Image> ().sprite = mus_off; // мзыка выключена в
настройках
            PlayerPrefs.SetString ("Music", "off");
            Camera.main.GetComponent <AudioListener> ().enabled = false; //
выключаем музыку
        }
        break;
            case "Setting":
                for (int i = 0; i < transform.childCount; i++)
```

```
                transform.GetChild (i).gameObject.SetActive
(!transform.GetChild (i).gameObject.activeSelf);
                        break;
                case "Shop":
                        shopBG.SetActive (!shopBG.activeSelf);
                        break;
                }
        }
}
// Анимация объектов  на главном экране!
public class ScrollObjects : MonoBehaviour
{
        public float speed = 5f, checkPos = 0f;
        private RectTransform rec;

        void Start()
        {
                rec = GetComponent<RectTransform>();
        }
        void Update()
        {
                if (rec.offsetMin.y != checkPos)
                {
                        rec.offsetMin += new Vector2 (rec.offsetMin.x, speed);
                        rec.offsetMax += new Vector2 (rec.offsetMax.x, speed);
                }
        }
}

// Мерцание текста
```

```
public class TextFade : MonoBehaviour
{
    private Text txt;
    private Outline oLine;

  void Start()
  {
    txt = GetComponent<Text>();
        oLine = GetComponent<Outline>();
  }
  void Update()
  {
    txt.color = new Color (txt.color.r, txt.color.g, txt.color.b, Mathf.PingPong(Time.time, 1.0f));
        oLine.effectColor = new Color(oLine.effectColor.r, oLine.effectColor.g, oLine.effectColor.b, txt.color.a);
  }
}

// Всплывание ромбика
public class Item : MonoBehaviour
{
    public GameObject item;
    void Start()
    {
        StartCoroutine(spawn());
    }
    IEnumerator spawn()
    {
        while(true)
```

```
                {
                        Vector3 pos = Camera.main.ScreenToWorldPoint (new Vector3 (Random.Range (0, Screen.width), Random.Range(0, Screen.height), Camera.main.farClipPlane / 2));
                        Instantiate (item, pos, Quaternion.Euler (0, 0, Random.Range (0f, 360f)));
                        yield return new WaitForSeconds(5f);
                }
        }
    }
    // Анимация ромбика!
    public class ItemAnimation : MonoBehaviour
    {
        private SpriteRenderer item;
        private float _movementSpeed = 0.1f;

        void Start ()
        {
            item = GetComponent<SpriteRenderer>();
            Destroy (gameObject, 5f);
        }
        void Update()
        {
            item.color = new Color (item.color.r, item.color.g, item.color.b, Mathf.PingPong (Time.time / 2.5f, 1.02f));
    // Движение ромбика
            transform.position += transform.up * Time.deltaTime * _movementSpeed;

        }
```

```
}

// Проверка клика на экране! Старт игры!
// Анимации
public class DetectClick : MonoBehaviour
{
    public GameObject[] cubes;
    public GameObject buttons, m_cube, spawn_blocks, diamonds;
    public Animation cubes_anim, block;
    public Light dirLight;
    public Text playTxt, gamename, record;
    private bool clicked;
    void Update()
    {
        if (clicked && dirLight.intensity != 0)
            dirLight.intensity -= 0.05f;
    }
    void OnMouseDown()
    {
        if (!clicked)
            {
                StartCoroutine(delCubes());
                clicked = true; // Клик срабатывает только раз.
                playTxt.gameObject.SetActive(false);
                record.gameObject.SetActive (true);
                diamonds.SetActive (true);
                gamename.text = "0";
                buttons.GetComponent<ScrollObjects>().speed = -
5f;
```

```
                    buttons.GetComponent<ScrollObjects>().checkPos
= 130f;

        m_cube.GetComponent<Animation>().Play("StartGameCube");
                    StartCoroutine(CubeToBlock());
                    cubes_anim.GetComponent<Animation>().Play();
                }
            }
            IEnumerator delCubes()  // Уничтожение начальных кубиков
            {
                for (int i = 0; i < 3; i++)
                {
                    yield return new WaitForSeconds(0.7f);
                    cubes[i].GetComponent<FallCube>().enabled = true;
                }
                spawn_blocks.GetComponent<SpawnBlock>().enabled = true;
            }
            IEnumerator CubeToBlock() // Анимация куб в блок
            {
                yield return new
WaitForSeconds(m_cube.GetComponent<Animation>().clip.length + 0.5f);
                block.Play();
                // Добавляем RigidBody ФИЗИКА
                m_cube.AddComponent<Rigidbody>();
            }
        }
        // Уничтожение начальных кубов!
        public class FallCube : MonoBehaviour
        {
           void Start()
```

```
    {
        Destroy (gameObject, 1f);
    }
    void Update()
    {
        transform.position += new Vector3(0, 0.1f, 0);
    }
}


// Спаун новых блоков (рандомно)
public class SpawnBlock : MonoBehaviour
{
    public GameObject block, allCubes, diamond;
    private GameObject blockInst;
    private Vector3 blockPos;
    private float speed = 7f;
    private bool onPlace;
    void Start()
    {
            spawn();
    }
    void Update()
    {
            if (blockInst.transform.position != blockPos && !onPlace)
                blockInst.transform.position = Vector3.MoveTowards
(blockInst.transform.position, blockPos, Time.deltaTime * speed);
            else if (blockInst.transform.position == blockPos) // Блок не
стоит привязан к месту
                onPlace = true;
```

```
            if (CubeJump.jump && CubeJump.nextBlock)
            {
                spawn ();
                onPlace = false;
            }       }
        float RandScale ()
        {
            float rand;
            if (Random.Range (0, 100) > 80)
                rand = Random.Range (1.2f, 2f);
            else
                rand = Random.Range (1.2f, 1.5f);
            return rand;
        }
        void spawn ()
        {
            blockPos = new Vector3 (Random.Range(0.74f, 1.42f),
Random.Range(-0.5f, -3.2f), -3.75f);
            blockInst = Instantiate (block, new Vector3 (5f, -6f, 0f),
Quaternion.identity) as GameObject;
            blockInst.transform.localScale = new Vector3 (RandScale (),
blockInst.transform.localScale.y, blockInst.transform.localScale.z);
            blockInst.transform.parent = allCubes.transform;
            if (CubeJump.count_blocks % 1 == 0)
    // Создание Diamond
            {
                GameObject diamondInst = Instantiate (diamond, new
Vector3 (blockInst.transform.position.x, blockInst.transform.position.y + 0.6f,
blockInst.transform.position.z), Quaternion.Euler
(Camera.main.transform.eulerAngles)) as GameObject;
```

```
                diamondInst.transform.parent = blockInst.transform;
            }
        }
    }

    // Прыжки куба (главное объекта)
    public class CubeJump : MonoBehaviour
    {
        public static int count_blocks;
        public static bool jump, nextBlock;
        public GameObject mainCube, buttons, lose_buttons;
        private bool animate, lose, addLose;
        private float scratch_speed = 0.5f, startTime, yPosCube; // Скорость
сжатия кубика


        void Awake () // что бы при рестарте не уезжала сцена
        {
            jump = false;
            nextBlock = false;
        }
        void Start ()
        {
            StartCoroutine (CanJump( )); // Начало игры, можно прыгать
        }
        void FixedUpdate ()
        {
            if (animate && mainCube.transform.localScale.y > 0.4f)
                PressCube (-scratch_speed);
            else if (!animate && mainCube != null)
```

```
		{
			if (mainCube.transform.localScale.y < 1.5f)
				PressCube (scratch_speed * 2f);
			else if (mainCube.transform.localScale.y != 1.5f)
			mainCube.transform.localScale = new Vector3 (1.5f, 1.5f, 1.5f);
		}
		if (mainCube != null) {
			if (mainCube.transform.localPosition.y < -15f)
			{
				Destroy (mainCube, 1f);
				print ("Player Lose");
				lose = true;
			}
		}
		if (lose && !addLose)
			PlayerLose ();
	}
	void PlayerLose ()
	{
		addLose = true;
		buttons.GetComponent <ScrollObjects> (). speed = 5f;
		buttons.GetComponent <ScrollObjects> (). checkPos = 0;
		if (!lose_buttons.activeSelf)
			lose_buttons.SetActive (true);
		lose_buttons.GetComponent <ScrollObjects> (). checkPos = 130;
	}
	void OnMouseDown ()
	{
```

```
        if (nextBlock && mainCube.GetComponent<Rigidbody>())
        {
            animate = true;
            startTime = Time.time;

            yPosCube = mainCube.transform.localPosition.y;
        }
    }
    void OnMouseUp ()
    {
        if (nextBlock && mainCube.GetComponent<Rigidbody>())
        {
            animate = false;
            // jupm
            jump = true;
            float force, diff;
            diff = Time.time - startTime;
            if (diff < 3)
                force = 190 * diff;
            else
                force = 300f;
            if (force < 60f)
                force = 60f;
            mainCube.GetComponent                    <Rigidbody>
().AddRelativeForce (mainCube.transform.up * force);
            mainCube.GetComponent                    <Rigidbody>
().AddRelativeForce (mainCube.transform.right * -force);
            StartCoroutine (checkCubePos ());
            nextBlock = false;
        }
```

```
        }


        void PressCube (float forse)
        {
            //mainCube.transform.localPosition += new Vector3 (0f, forse *
Time.deltaTime, 0f);
            mainCube.transform.localScale += new Vector3 (0f, forse  *
Time.deltaTime, 0f);
        }
        IEnumerator checkCubePos ()
        {
            yield return new WaitForSeconds (1.5f);
            if (yPosCube == mainCube.transform.localPosition.y)
            {
                print ("Player Lose");
                lose = true;
            }
            else
            {
                while (mainCube.GetComponent <Rigidbody>
().IsSleeping ())
                {
                    yield return new WaitForSeconds (0.05f);
                    if (mainCube == null)
                    {
                        break;
                    }

                }
```

```
                    if (!lose)
                    {
                        nextBlock = true;
                        count_blocks++;
                        print ("Next one");

                        mainCube.transform.localPosition = new Vector3 (mainCube.transform.localPosition.x, mainCube.transform.localPosition.y, mainCube.transform.localPosition.z);
                        mainCube.transform.eulerAngles = new Vector3 (0f, mainCube.transform.eulerAngles.y, 0f);
                    }
                }
            }
            IEnumerator CanJump () // После секунды можно прыгать!
            {
                while (!mainCube.GetComponent <Rigidbody> ())
                    yield return new WaitForSeconds (0.05f);
                yield return new WaitForSeconds (1f);
                nextBlock = true;
        // движение кубов
        public class MoveCubes : MonoBehaviour
        {
            private bool moved = true;
            private Vector3 target;
           void Start()
           {
            target = new Vector3 (-4.69f, 3.24f, -1.5f);
           }
           void Update()
```

```
    {
        if (CubeJump.nextBlock)
            {
                if (transform.position != target)
                    transform.position = Vector3.MoveTowards
(transform.position, target, Time.deltaTime * 5f); // Движение блоков в позицию
                else if (transform.position == target && !moved)
                {
                    target = new Vector3(transform.position.x - 2f,
transform.position.y + 2.89f, transform.position.z);
                    CubeJump.jump = false;
                    moved = true;
                }


                if (CubeJump.jump)
                    moved = false;
            }
    }
}

// удаление блоков
public class DeleteBlocks : MonoBehaviour
{
    void OnTriggerEnter (Collider other)
    {
        if (other.tag == "Cube")
            Destroy (other.gameObject);
    }
}
```

```
// сложность игры/смена камеры
public class Harder : MonoBehaviour
{
    public GameObject detectClick;
    private bool hard;
    void Update ()
    {
        if (CubeJump.count_blocks > 0)
        {
            if (CubeJump.count_blocks % 1 == 0 && !hard)
            {
                print ("Hard");
                Camera.main.GetComponent <Animation> ().Play
("Harder");
                detectClick.transform.position = new Vector3
(35.87f, -11.44f, -1.33f);
                detectClick.transform.eulerAngles = new Vector3
(19.459f, -63.667f, 0f);
                hard = true;
            }
        else if (CubeJump.count_blocks % 2 == 0 && hard)
            {
                hard = false;
                print("Easy");
                detectClick.transform.position = new Vector3
(26.89502f, -15.76862f, 0.16f);
                detectClick.transform.eulerAngles = new Vector3
(0f, 0f, 0f);
```

```
                    Camera.main.GetComponent <Animation> ().Play
("Easy");
                        }
                    }
                }
            }

            // счет рекорда
            public class Score : MonoBehaviour
            {
                public Text record;
                private Text txt;
                private bool gameStart;
              void Start()
              {
                    record.text = "Top: " + PlayerPrefs.GetInt ("Record").ToString
();
                    txt = GetComponent <Text> ();
                CubeJump.count_blocks = 0;
              }
              void Update()
              {
                if (txt.text == "0")
                        gameStart = true;
                    if (gameStart)
                    {
                        txt.text = CubeJump.count_blocks.ToString ();
                        if (PlayerPrefs.GetInt ("Record") <
CubeJump.count_blocks)
```

```
                    PlayerPrefs.SetInt ("Record",
CubeJump.count_blocks);
                    record.text = "Top: " + PlayerPrefs.GetInt
("Record").ToString ();
                }
            }
        }

        // счет айтемов
        public class CountDiamonds : MonoBehaviour
        {
            private Text txt;
            void Start()
            {
                txt = GetComponent <Text> ();
                    txt.text = PlayerPrefs.GetInt ("Diamonds").ToString ();
            }
        }

        // сбор айтемов
        public class CollectDiamonds : MonoBehaviour
        {
            public Text diamonds;
            void OnTriggerEnter (Collider other)
            {
                if (other.tag == "Diamond")
                {
                    Destroy (other.gameObject);
                    PlayerPrefs.SetInt ("Diamonds", PlayerPrefs.GetInt
("Diamonds") + 1);
```

```
                diamonds.text = PlayerPrefs.GetInt ("Diamonds").ToString ();
            }
        }
    }

    // цвет платформ
    public class RandColor : MonoBehaviour
    {
        public Color[] colors;
       void Start()
       {
          GetComponent <MeshRenderer> ().material.color = colors [Random.Range (0, colors.Length)];
       }
    }

    // цвет платформ меняется под цвет кубика
    public class BlockSameColor : MonoBehaviour
    {
        private bool firstOne;
        void OnCollisionEnter (Collision other)
        {
            if (other.gameObject.tag == "Cube" && firstOne)
                other.gameObject.GetComponent <MeshRenderer> ().material.color = GetComponent <MeshRenderer> ().material.color;
            if (!firstOne)
                firstOne = true;
        }
    }
```

```
// Магазин
public class ShopBG : MonoBehaviour
{
	private bool activeLose;
	public GameObject detectClick, allCubes, loseBtns; //playTxt
   void OnEnable()
   {
	//playTxt.SetActive (false);
		detectClick.GetComponent <BoxCollider> ().enabled = false;
		allCubes.SetActive (true);
		if (loseBtns.activeSelf)
		{
			activeLose = true;
		loseBtns.SetActive (false);
		}
   }
	void OnDisable()
	{
		//playTxt.SetActive (true);
		if (activeLose)
			loseBtns.SetActive (true);
		detectClick.GetComponent <BoxCollider> ().enabled = true;
		allCubes.SetActive (false);
	}
}

// Скрол кубиков в магазине
public class ScrollCubes : MonoBehaviour
{
```

```
public GameObject cubes;
private Vector3 screenPoint, offset;
private float _lockedYPos;

void Update()
{
    if (cubes.transform.position.x > 0)
        cubes.transform.position = Vector3.MoveTowards (cubes.transform.position, new Vector3 (0f, cubes.transform.position.y, cubes.transform.position.z), Time.deltaTime * 10f);
    else if (cubes.transform.position.x < -14f)
        cubes.transform.position = Vector3.MoveTowards (cubes.transform.position, new Vector3 (-14f, cubes.transform.position.y, cubes.transform.position.z), Time.deltaTime * 10f);
}
void OnMouseDown ()
{
    _lockedYPos = screenPoint.x;
    offset = cubes.transform.position - Camera.main.ScreenToWorldPoint (new Vector3 (Input.mousePosition.x, Input.mousePosition.y, screenPoint.z));
    Cursor.visible = false;
}
void OnMouseDrag ()
{
    Vector3 curScreenPoint = new Vector3 (Input.mousePosition.y, screenPoint.z);
    Vector3 curPosition = Camera.main.ScreenToWorldPoint (curScreenPoint) + offset;
    curPosition.y = _lockedYPos;
```

```
            cubes.transform.position = curPosition;
        }
        void OnMouseUp()
        {
            Cursor.visible = true;
        }
}

// Выбор кубика в магазине
public class SelectCube : MonoBehaviour
{
        [HideInInspector]
        public string nowCube;
        public GameObject selectCube, buyCube;
        void Start ()
        {
            if (PlayerPrefs.GetString ("Cube") != "Open")
                PlayerPrefs.SetString ("Cube", "Open");
        }
        void OnTriggerEnter (Collider other)
        {
            nowCube = other.gameObject.name;
            other.transform.localScale += new Vector3 (0.25f, 0.25f, 0.25f);
            if (PlayerPrefs.GetString(other.gameObject.name) == "Open")
            {
                selectCube.SetActive(true);
                buyCube.SetActive(false);
            }
            else
            {
```

```
                selectCube.SetActive(false);
                buyCube.SetActive(true);
            }
        }
        void OnTriggerExit (Collider other)
        {
            other.transform.localScale -= new Vector3 (0.25f, 0.25f, 0.25f);
        }
    }

    // Покупка кубика в магазине
    public class BuyCube : MonoBehaviour
    {
        public GameObject selectBtn;
        public GameObject whichCube;
        public GameObject mainCube;
        void OnMouseDown ()
        {
            if (PlayerPrefs.GetInt ("Diamonds") >= 50)
            {
                PlayerPrefs.SetString (whichCube.GetComponent
<SelectCube>().nowCube, "Open");
                PlayerPrefs.SetString ("Now Cube",
whichCube.GetComponent <SelectCube>().nowCube);
                PlayerPrefs.SetInt ("Diamonds", PlayerPrefs.GetInt
("Diamonds") - 50);
                mainCube.GetComponent <MeshRenderer> ().material =
GameObject.Find (whichCube.GetComponent
<SelectCube>().nowCube).GetComponent <MeshRenderer> ().material;
                selectBtn.SetActive (true);
```

```
                gameObject.SetActive (false);
            }

    // Какой куб выбран сейчас
    public class SelectCubeNow : MonoBehaviour
    {
        public GameObject whichCube;
        public GameObject mainCube;
        void OnMouseDown ()
        {
            mainCube.GetComponent <MeshRenderer> ().material = GameObject.Find (whichCube.GetComponent <SelectCube>().nowCube).GetComponent <MeshRenderer> ().material;
            PlayerPrefs.SetString ("Now Cube", whichCube.GetComponent <SelectCube>().nowCube);
        }
    }
```