УДК 004.92 doi.org/10.33298/2226-8553/2020.2.30.11

Глущенко О.А., Шикула О.М.

РОЗРОБКА МОБІЛЬНОГО ДОДАТКА – ГРИ НА ПЛАТФОРМІ ANDROID

Було розглянуто один з найбільших сегментів ринку програмних продуктів – відеоігри. Досліджено алгоритм створення мобільної гри. Висвітлено розробку під обрану платформу із використанням Android Studio, Cocos Studio, роботу об'єктного менеджера, обробку грою дій гравця. В ході розробки відеогри було показано її складові, а саме написання коду, створення контенту, розробка механік гри та тестування.

Запропоновано алгоритм генерування об'єктів та результат зіткнення гравця з ними. Розкривається процес розробки відеоігри та розділення його на етапи. Описано алгоритм оптимізації зберігання та використання зображень. Досліджена якісна робота з пам'яттю, насамперед, на мобільних пристроях. Детально описані поняття сцени та спрайту, встановлено правила роботи зі сценами та зображеннями на екрані мобільного пристрою. Розглянуто два схожих за своїм підходом і в той же час принципово різних по результату алгоритми виявлення зіткнень об'єктів. Описаний програмний продукт включає в себе власноруч написаний рушій – рендерер. За теоретичну основу реалізації рендерингу було взято спрощений варіант рейтрейсингу – рейкастинг. Метод рейкастингу вибрано як оптимальний через його високу швидкодію при достатній якості відео. Було обрано крос-платформовий фреймворк, який використовується для розробки інтерактивних додатків та ігор. Розглянуто використання вбудованих в ігровий движок візуального редактора, готових модулів рендеринга, анімації спрайтів і обробки зіткнень, що дуже спрощує процес розробки.

Описано структуру програмного продукту та ігрові класи сутностей, такі як персонаж, предмети. Наведено алгоритм реалізації методу рейкастингу і проведено відповідні математичні розрахунки для побудови променя. Змодельовано дизайн оформлення простору гри на основі карти, що задається, з додаванням текстур. Додатково розроблена можливість самостійної генерації рівнів.

Ключові слова: *розробка ігор, мобільний додаток, операційна система Android, алгоритм розробки мобільного додатку, рендерер, метод рейкастингу, крос-платформовий фреймворк*

Постановка проблеми. В наш час такий неоднозначний ринок, як комп'ютерні ігри, є одним із найприбутковіших серед усіх IT-напрямків. На сьогоднішній день комп'ютерні ігри (або відеоігри) вже давно не лише розвага для дітей чи математиків. За статистикою середній вік гравця у відеоігри становить 30 років. Вплив відеоігор на сфери життя людей постійно збільшується. Відеоігри мають не тільки розважальну складову, за останні 10 років вони перетворилися ледь не на витвір мистецтва, часто маючи сильну літературну, художню, смислову та інколи навіть оздоровляючу цінність. У відеоіграх є стільки жанрів, скільки є в кіно, або в літературі.

Тому, виникає питання дослідити процес розробки гри під одну з найпоширеніших платформ сучасності – Android, а саме, висвітлити цей процес поетапно, а також показати принципи розробки.

Аналіз останніх досягнень та публікацій. Розробка додатків на платформу Android вже була розглянута в роботах Р. Рупіуса, М. Мілчева, М. Доусона, М. Бакленда, та багатьох

інших. Втім жоден із них не продемонстрував розробку при поєднанні саме таких технологій, використання і поєднання ряду алгоритмів, а також їх сумісність.

Мета дослідження. Дана робота має на меті створення інді гри жанру 2D-runner на мові C++ з використанням бібліотеки STL. Одна із головних цілей – продемонструвати різні сторони мови C++, яка незважаючи на свою складність, є кросплатформенною і дуже зручною в використанні.

Основні результати дослідження. ***Сцени у мобільних додатках.*** В бібліотеці Cocos2D реалізований клас CCMenu, за допомогою якого зручно створювати список вибору альтернатив для клієнта. Також Cocos2D надає інтерфейс для відтворення кнопок і дозволяє відловлювати натискання на елементи меню.

Таким чином, використання CCMenu порушує звичну для iPhone-розробників реалізацію за допомогою паттерна ModelViewController, але значно економить час, оскільки надає безліч готових рішень. Клас CCScene (бібліотеки Cocos2D) дозволяє створювати сцени – 2D графічні об'єкти для відображення користувачу ігрової ситуації. Розглянемо взаємодію сцен між собою.

На початку роботи програми AppDelegate (перший клас, який отримує управління) створює MainMenuScene і звідти, в залежності від вибору гравця буде йти пересування по сценам. При натисканні на елемент меню створюється сцена і їй передається керування. Після того, як гравець виконав всі бажані дії (вибрати героя, подивитися результати, подивитися правила гри) і перейшов до гри, управління передається GameController`у.

Механізм збереження інформації та робота з пам'яттю. В казуальні ігри грають у вільний час. Найчастіше це відбувається в маршрутках, в чергах, в «вільні п'ять хвилин». В зв'язку з цим часто користувач зупиняє гру, не догравши до кінця. Для відновлення гри з місця зупинки реалізований механізм збереження і відновлення гри. Збереження запускає клас LevelController, він посилає повідомлення класу LevelState «записати себе в файл».

LevelState записує свої поля в xml фай, причому, якщо поля є об'єктами класу «ігрові об'єкти», то їм надсилається повідомлення «записати свої поля в словник» і віддати його об'єкту, який викликав дану функцію. При необхідності продовжити гру отриманий файл десеріалізується і всі об'єкти відновлюються [1].

Мобільні пристрої мають істотні обмеження за обсягом пам'яті. Однак для створення повноцінної, привабливою для користувача гри необхідна робота з великою кількістю графічних ресурсів і музикою. Тому встає питання про збільшення продуктивності програми та знаходження вузьких місць в роботі з пам'яттю.

Для економії пам'яті були зроблені наступні заходи:

1. Оптимізація зберігання і використання картинок.

2. Збереження анімації.

Робота всіх версій Cocos2D заснована на використанні спрайтів. Спрайти можна розглядати як прості 2D зображення, але також вони можуть бути контейнером для інших спрайтів. В Cocos2D, розміщенні разом спрайти створюють сцену, наприклад, рівень гри або головне меню. Спрайтами можна керувати на основі подій у вихідному коді або як частиною анімації. Над спрайтами можна проводити певні дії: переміщувати, повертати, масштабувати, змінювати зображення і т.п. [1].

В можливості даної бібліотеки входять:

1. Управління сценами. Всю гру можна розбити на сцени. Кожна сцена це окремий підпроект. Між сценами можна переключатися з використанням різних ефектів.

2. Спрайт (растрове зображення, вільно переміщається по екрану) і менеджери спрайтів. Бібліотека надає великі можливості по управлінню спрайтами. Менеджери спрайтів дозволяють оперувати атласами спрайтів, завдяки чому можна ефективно використовувати пам'ять. Так само є класи для роботи з спрайтовими шрифтами, використання яких істотно прискорює відображення тексту.

3. Анімація (actions). Анімація в Cocos2D здійснюється за допомогою спеціальних класів, які називаються Actions. Їх можна застосовувати майже до будь-якого об'єкта в грі. В бібліотеці є великий набір типів анімацій, а також можна створювати свої.

4. Базова реалізація меню і кнопок.

5. Система частинок. Движок підтримує системи частинок. Завдяки цьому можна створювати різні ефекти, наприклад дощ, сніг, феєрверк. Частинки можуть відрізнятися розмірами, обертатися, до них може застосовуватися гравітація, можна налаштовувати їх час життя і так далі.

6. Вбудовані фізичні движки Box2d iChipmunk [2].

7. CocosLive - сервіс для онлайн-рекордів. Cocos2D використовує сервери, які надаються GoogleAppEngine для зберігання онлайн-рекордів, і надає API для роботи з цими рекордами, але в даному проекті використовується сервіс OpenFeint, так як він набагато популярніший у користувачів і зручніший в використанні.

Бібліотека Cocos2D дозволяє працювати з OpenGL безпосередньо. Текстура в Open GL ES повинна мати ширину і висоту, кратні ступеню двійки, наприклад, 64x128, 256x1024, 512x512 і т.п. На рис. 1 текстура розміром 144x93 стала 256x128 [2].

У нас виявилося більше незайнятого простору, ніж зайнятого під текстуру. Для однієї текстури це не так критично, але ж тільки для анімації вихлопних газів машини використовується 125 текстур. Для вирішення даної проблеми застовується атлас текстур.

Атлас - велика текстура, яка складається з великої кількості невеликих текстур. Таким чином, можна розташувати текстури в атласі так, щоб залишалося якомога менше невикористаної пам'яті [3].

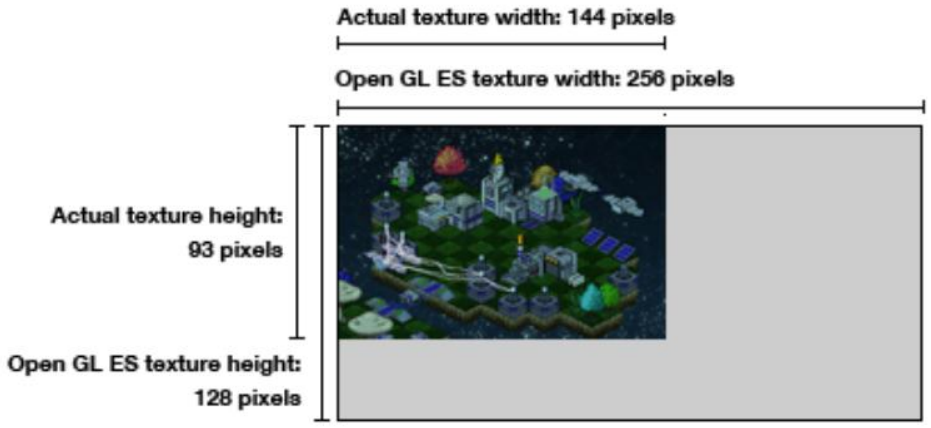

Рисунок 1 – Зберігання текстури

Так як існують спрайти, які виглядають так само, коли їх змінюють в протилежному напрямку, то для зменшення використовуваної пам'яті можна брати ті ж текстури, а відбиття провести в коді. Те ж саме можна сказати і про зміну розміру картинок, про зміну кута повороту [4].

Тому в Атласі можна успішно зберігати картинки в перевернутому вигляді, якщо це необхідно для оптимізації зберігання.

На створення анімації пересування потрібна велика кількість виділенної пам'яті, тому часто повторювану анімацію логічно зберігати, і тоді замість виділення пам'яті під нову дію необхідно буде тільки проініціалізувати цю анімацію.

Інструмент розробки Android Studio. Android Studio прийшло на зміну плагіну ADT для платформи Eclipse. Середовище побудоване на базі вихідного коду продукту IntelliJ IDEA

Community Edition, що створено компанією JetBrains. Android Studio розвивається в рамках відкритої моделі розробки та поширюється під ліцензією Apache 2.0[5].

Бінарні складання підготовлені для Linux (для тестування використаний Ubuntu), Mac OS X і Windows. Середовище надає засоби для розробки застосунків не тільки для смартфонів і планшетів, але і для портативних пристроїв на базі Android Wear, телевізорів (Android TV), окулярів Google Glass і автомобільних інформаційно-розважальних систем (Android Auto). Для застосунків, спочатку розроблених з використанням Eclipse і ADT Plugin, підготовлений інструмент для автоматичного імпорту існуючого проекту в Android Studio [5].

Середовище розробки адаптоване для виконання типових завдань, що вирішуються в процесі розробки застосунків для платформи Android. Також в середовище включені засоби для спрощення тестування програм на сумісність з різними версіями платформи та інструменти для проектування застосунків, що працюють на пристроях з екранами різної роздільності (планшети, смартфони, ноутбуки, годинники, окуляри тощо). Крім можливостей, присутніх в IntelliJ IDEA, в Android Studio реалізовано кілька додаткових функцій, таких як нова уніфікована підсистема складання, тестування і розгортання застосунків, заснована на складальному інструментарії Gradle і підтримуюча використання засобів безперервної інтеграції [6].

Для прискорення розробки застосунків представлена колекція типових елементів інтерфейсу і візуальний редактор для їхнього компонування, що надає зручний попередній перегляд різних станів інтерфейсу застосунку (наприклад, можна подивитися як інтерфейс буде виглядати для різних версій Android і для різних розмірів екрану). Для створення нестандартних інтерфейсів присутній майстер створення власних елементів оформлення, що підтримує використання шаблонів. В середовище вбудовані функції завантаження типових прикладів коду з GitHub [6].

До складу також включені пристосовані під особливості платформи Android розширені інструменти рефакторингу, перевірки сумісності з минулими випусками, виявлення проблем з продуктивністю, моніторингу споживання пам'яті та оцінки зручності використання. В редактор доданий режим швидкого внесення правок. Система підсвічування, статичного аналізу та виявлення помилок розширена підтримкою Android API. Інтегрована підтримка оптимізатора коду ProGuard [7]. Вбудовані засоби генерації цифрових підписів. Надано інтерфейс для управління перекладами на інші мови. На рис. 2 можно побачити інтерфейс Android Studio.

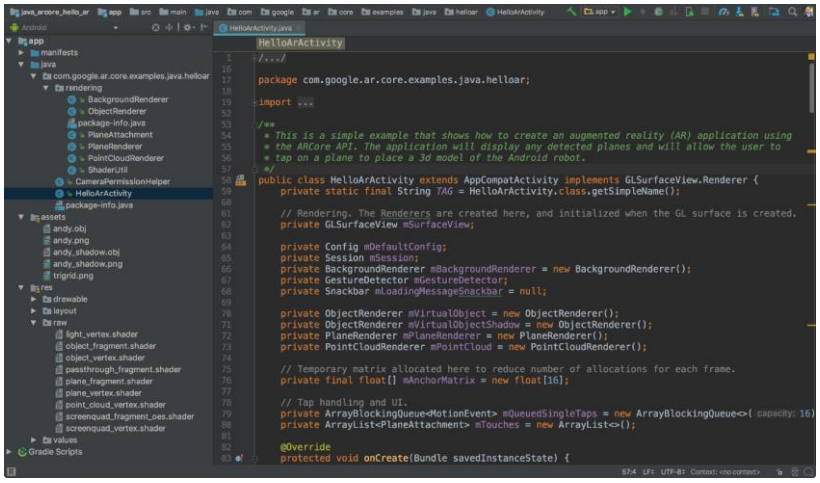

Рисунок 2 – Android Studio

Алгоритми зіткнень об'єктів в мобільному додатку. Для 2D, як правило, є три різні варіанти виявлення зіткнень:

- На основі зображення.
- Прості геометричні форми (орієнтовані обмежувальні прямокутники, кола).
- Складні геометричні фігури (опуклі багатокутники, увігнуті багатокутники, еліпси, тощо).

Виявлення зіткнення на основі зображень є точним та простим у використанні та розумінні. Що стосується ігор, які використовують зображення для малювання, то виявлення зіткнень на основі зображень означає, що всякий раз, коли спрайти на екрані перекриваються, вони також перекриваються в системі виявлення зіткнень[7]. Вони також корисні для ігор, де потрібні деформовані об'єкти зіткнення, такі як для зруйнованого рельєфу. Їх головним недоліком є те, що вони дуже неефективні в порівнянні з іншими методами, особливо при обертанні і масштабуванні об'єктів зіткнення.

Прості геометричні форми є як легкими в роботі, так і дуже ефективними. Вони використовуються, якщо висока точність не потрібна, або об'єкти зіткнення добре вписуються в прості геометричні форми, які ідеально підходять, іноді навіть краще, ніж зображення (наприклад, якщо ваші об'єкти зіткнення - це кульки). Головним недоліком простих форм є важкість забезпечення їх точності при моделюванні складних фігур. Для високої точності, де основні фігури не підходять, потрібно або об'єднати прості форми в складніші форми, або використовувати більш загальні і складні форми [8].

Складні геометричні форми можуть бути точними і відносно ефективними або неефективними в залежності від складності використовуваної форми (форм) для представлення об'єкта зіткнення [8]. Важливим недоліком є складність використання. Коли об'єкти зіткнення не співпадають з наявними геометричними фігурами, то для їх представлення необхідно використовувати декілька форм, можливо різних, що потребує часу. Крім того, деякі з форм є складними і їх важко створити, якщо неможливо їх автоматично генерувати з зображення. Важливою перевагою є те, що обертання і масштабування є ефективними і легкими, особливо в порівнянні з виявленням зіткнень на основі зображень.

Виявлення зіткнень на основі зображень зазвичай розглядається як погане рішення, оскільки воно часто є неефективним, особливо при використанні обертання і масштабування. До цієї категорії також належить виявлення зіткнень по піксельно, також відоме як pixel-perfect collision detection, яке виявляє зіткнення між об'єктами, які представлені у вигляді зображень [9].

В цій грі використовується два типи виявлення зіткнень: так званий pixel-perfect collision для виявлення зіткнень між кулями та ворогами, і виявлення зіткнень на основі простих геометричних форм, а саме обмежувальних для виявлення зіткнень між стінами та об'єктами (персонажем гравця, ворогами та кулями)[9].

Виявлення зіткнень стрільби проходить наступним чином. Спочатку зіткнення кулі та ворога перевіряється методом простих геометричних форм, і при наявності зіткнення, далі перевіряється ще раз, але вже методом pixel-perfect collision [10]. Для виявлення зіткнень методом pixel-perfect collision, на основі текстур двох об’єктів, зіткнення яких перевіряється, створюються бітові маски і по піксельно перевіряється, чи накладається одна текстура на іншу.

Оскільки в грі одночасно може існувати всього кілька куль, а вороги мають незвичайну форму, яку дуже складно точно обвести геометричними формами, доцільно використовувати метод pixel-perfect collision, що дозволяє максимально точно перевірити наявність зіткнення двох об’єктів.

Також виявлення зіткнень у два етапи сильно оптимізує процес обчислення. [10]

Архітектура всього проекту. *С*початку для поділу логіки та інтерфейсу було вирішено використовувати типове рішення патерн ModelViewController. Але, по-перше, для роботи з графікою найвигідніше використовувати бібліотеку Cocos2D, механізми якої передбачають

об'єднання в собі відразу і уявлення, і контролера, і моделі. Деякі об'єкти архітектури виявилося не виправдано складно моделювати в рамках даного патерну.

По-друге, класична парадигма GUI додатків не підходить, так як у нас є не тільки користувач, що впливає на зміну інтерфейсу, але і різні об'єкти, які взаємодіють з 2D сценою і іншими об'єктами. Тому додаток побудовано як взаємодія ігрових об'єктів, які інкапсулюють кожен свою логіку [11].

Всі класи розділені на 3 блоки. В першому блоці представлені класи Сцен, які відображаються користувачу і клас GameCharacter, від якого успадковуються всі анімовані об'єкти гри. Клас GameCharacter має методи для відтворення себе на сцені.

В другому блоці знаходяться класи Контролерів, які в залежності від параметрів гри (GameParameters), рівня (LevelParameters) і стану (GameState, LevelState), а також дій гравця здійснюють взаємозв'язок між об'єктами гри, змінюючи їх стан, а також посилають повідомлення сцені відображати нову ігрову ситуацію[12].

Третій блок відповідає за логіку гри. Класи даного блоку містять інформацію про поточний Стані гри і рівня і про Параметри гри і рівня, які задаються адміністратором гри.

На рис. 3 можна побачити структуру даного програмного продутку.

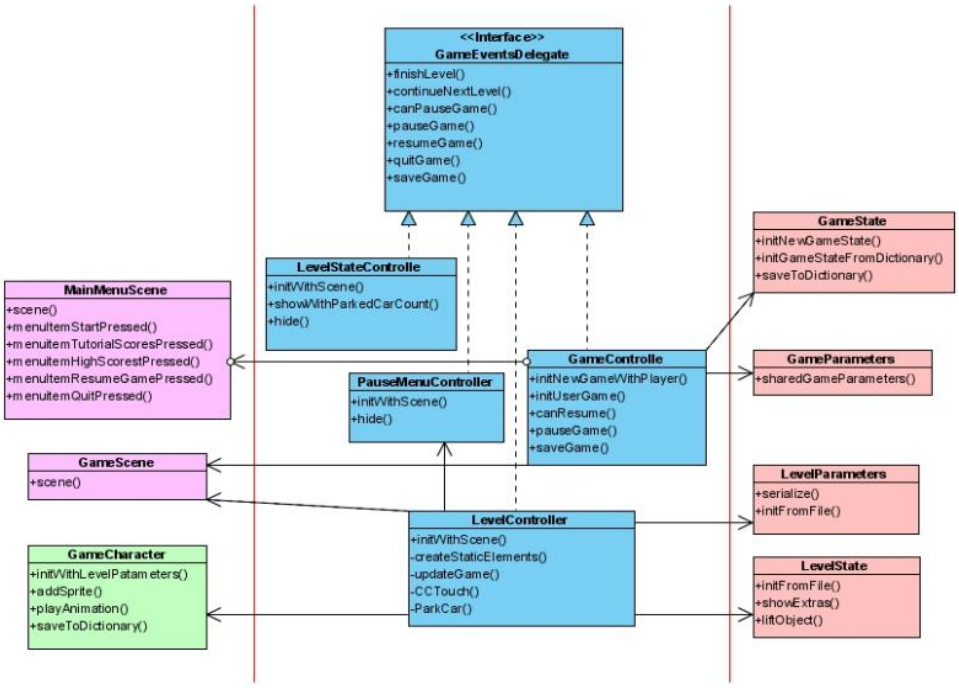

Рисунок 3 – Структура програмного продукту

Висновки. Була розкрита тема розробки відеоігор як таких і розробка відеоігор жанру 2d-runner. В ході розгляду теми про розробку відеоігор було розкрито її принципи, а саме написання коду, створення контенту, механік гри та її тестування.

В роботі були приведені два досить схожих і в той же час досить відмінних алгоритми розробки. Іншою ключовою темою стали алгоритми виявлення зіткнень об’єктів. В ході розгляду алгоритмів розглянуто всі три види виявлення зіткнень в двовимірних іграх, і розібрані ті види, які були застосовані під час розробки нашої гри.

ЛИТЕРАТУРА

1. Pupius R. Mastering SFML Game Development. Берлін: Packt Publishing, 2017, 815с.
2. Milchev M. SFML Essentials. Берлін: Packt Publishing, 2015, 263с.
3. Pupius R. SFML Game Development By Example. Berlin: Packt Publishing, 2015, 156с.

4. Zombie Top-Down Shooter Game Kit. URL: https://free-game-assets.itch.io/zombie-top-down-shooter-game-kit.
5. A* Search Algorithm. URL: https://www.geeksforgeeks.org/a-search-algorithm/.
6. Top Down Shoot Em Up Mechanics. Part 2. URL: https://pushbuttonreceivecode.com/blog/top-down-shoot-em-up-mechanics-part-2.
7. Tiled Documentation. URL: https://doc.mapeditor.org/en/stable/.
8. Доусон М. Beginning C++ Through Game Programming. Лондон: Cengage Learning, 2004, 284с.
9. Бакленд М. Programming game AI by example. Нью-Йорк: Wordware Publishing, Inc., 2004, 495с.
10. Розробка відеогри. URL: https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B2%D1%96%D0%B4%D0%B5%D0%BE%D0%B3%D1%80%D0%B8.
11. Tutorials for SFML 2.5. URL: https://www.sfml-dev.org/tutorials/2.5/.
12. Collision detection. URL: https://en.wikipedia.org/wiki/Collision_detection.

REFERENCES

1. Pupius R. (2017). Mastering SFML Game Development. Берлін: Packt Publishing, 2017, 815с.
2. Milchev M. (2015). SFML Essentials. Берлін: Packt Publishing. 263с.
3. Pupius R. (2015). SFML Game Development By Example. Berlin: Packt Publishing. 156с.
4. Zombie Top-Down Shooter Game Kit. URL: https://free-game-assets.itch.io/zombie-top-down-shooter-game-kit.
5. A* Search Algorithm. URL: https://www.geeksforgeeks.org/a-search-algorithm/.
6. Top Down Shoot Em Up Mechanics. Part 2. URL: https://pushbuttonreceivecode.com/blog/top-down-shoot-em-up-mechanics-part-2.
7. Tiled Documentation. URL: https://doc.mapeditor.org/en/stable/.
8. Доусон М. (2004). Beginning C++ Through Game Programming. Лондон: Cengage Learning. 284с. (in Russia).
9. Бакленд М. (2004). Programming game AI by example. Нью-Йорк: Wordware Publishing, Inc. 495с. (in Russia).
10. Розробка відеогри. URL: https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B2%D1%96%D0%B4%D0%B5%D0%BE%D0%B3%D1%80%D0%B8. (in Russia).
11. Tutorials for SFML 2.5. URL: https://www.sfml-dev.org/tutorials/2.5/.
12. Collision detection. URL: https://en.wikipedia.org/wiki/Collision_detection

Глущенко А.А., Шикула Е.Н.

РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ – ИГРЫ НА ПЛАТФОРМЕ ANDROID

Был рассмотрен один из наибольших сегментов рынка продуктов - видеоигры. Исследован алгоритм создания мобильной игры. Освещена разработка под выбранную платформу с использованием Android Studio, Cocos Studio, работа объектного менеджера, обработка игрой действий игрока. В ходе разработки видеоигры были показаны ее составляющие, а именно написания кода, создание контента, разработка игровых механик и тестирования.

Предложен алгоритм генерации объектов и результата столкновения игрока с ними. Раскрывается процесс разработки видеоигры и разделения его на этапы. Описан алгоритм оптимизации сохранения и использования изображений. Исследована качественная работа с памятью, в первую очередь, на мобильных устройствах. Детально описаны понятия сцены и спрайта, установлены правила работы со сценами и изображениями на экране мобильного

устройства. Были рассмотрены два похожих по своему подходу, и, в то же время, принципиально разных по результату алгоритма определения столкновений объектов. Описанный программный продукт включает в себя собственноручно написанный движок - рендерер. В качестве теоретической основы реализации рендеринга был взят упрощенный вариант рейтрейсинга - рейкастинг. Метод рейкастинга был выбран как оптимальный за его высокую скорость при достаточном качестве видео. Был выбран крос-платформенный фреймворк, который используется для разработки интерактивных приложений и игр. Рассмотрено использование встроенных в игровой движок визуального редактора, готовых модулей рендеринга, анимации спрайтов и обработки столкновений, что очень упрощает процесс разработки.

Описана структура программного продукта и игровые классы сущностей, такие как персонаж, предметы. Приведен алгоритм реализации метода рейкастинга и проведены соответствующие математические расчеты для построения прямой. Смоделирован дизайн оформления игрового пространства на основной заданной карты с добавлением текстур. Дополнительно разработана возможность самостоятельной генерации уровней.

***Ключевые слова**: разработка игр, мобильное приложение, операционная система Android, алгоритм разработки мобильного приложения, рендерер, метод рейкастинга, крос-платформенный фреймворк*

Hlushchenko A.A., Shikula E.N.

MOBILE APP DEVELOPMENT ON ANDROID PLATFORM

Here was considered one of the largest segments of the product market - video games. The algorithm for creating a mobile game is investigated. The development for the selected platform using Android Studio, Cocos Studio, the work of the object manager, the processing of the player's actions by the game are highlighted. During the development of a video game, its components were shown, namely writing code, creating content, developing game mechanics and testing.

An algorithm for generating objects and the result of a player's collision with them is proposed. The process of developing a video game and its division into stages is disclosed. An optimization algorithm for saving and using images is described. The high-quality work with memory is investigated, first of all, on mobile devices. The concepts of scene and sprite are described in detail, the rules for working with scenes and images on the screen of a mobile device are established. We considered two similar in their approach, and, at the same time, fundamentally different algorithms for determining collisions of objects. The described software product includes a hand-written engine - a renderer. As a theoretical basis for the implementation of rendering, a simplified version of raytracing was taken - rakecasting. The method of rakasting was chosen as optimal for its high speed with sufficient video quality. A cross-platform framework was chosen, which is used to develop interactive applications and games. We consider the use of the built-in visual engine in the game engine, ready-made rendering modules, animation sprites and collision processing, which greatly simplifies the development process.

The structure of the software product and game classes of entities such as character, objects are described. An algorithm for the implementation of the rakasting method is presented and the corresponding mathematical calculations are carried out to construct a straight line. The design of the design of the game space on the main map, which is set, with the addition of textures, is simulated. Additionally, the ability to independently generate levels has been developed.

***Keywords:** game development, mobile app, Android operating system, mobile app development algorithm, renderer, ray casting, cross-platform framework.*