

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Грузін Роман Олександрович

**АНАЛІЗ МЕТОДІВ І ТЕХНОЛОГІЙ РОЗРОБКИ КОМП'ЮТЕРНОЇ 2D ГРИ
ДЛЯ ПЛАТФОРМИ ANDROID**

**кваліфікаційна робота
здобувача вищої освіти другого (магістерського) рівня
освітньої програми «Мультимедійні системи»
за спеціальністю 121 Інженерія програмного забезпечення**

Особистий підпис _____ Роман ГРУЗІН

Науковий керівник _____ Світлана ДОНЧЕНКО,
асистент кафедри інформаційних
технологій та систем

В.о. завідувача кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

АНОТАЦІЯ

Грузін Р. О.

Тема: Аналіз методів і технологій розробки комп'ютерної 2D гри для платформи ANDROID.

Спеціальність: 121 «Інженерія програмного забезпечення».

Установа: ЛНУ імені Тараса Шевченка, 2024р.

Магістерська робота містить: 64 с., 36 рис., 2 табл., 37 джерел.

Об'єкт дослідження: методи, підходи та інформаційні технології, використовувані для розробки ігрового програмного забезпечення для пристроїв, що працюють на платформі Android.

Предмет дослідження: процеси створення двомірного ігрового застосунку жанру "шутер" для платформи Android.

Мета роботи - аналіз методів і технологій розробки комп'ютерної 2D гри для платформи ANDROID.

Результати роботи. На основі статистичних даних по продажах був обраний жанр мобільної гри і дана оцінка популярності мобільної платформи, під яку ведеться розробка. Статистичні дані надані дослідними і консалтинговими компаніями, що спеціалізуються на ринках інформаційних технологій. До стадії проектування і програмування, програмне забезпечення було класифіковано за призначенням: робота з графікою Cocos2d 3.14, робота з кодом Microsoft VS22 і робота з фізичним світом Box2d. З огляду на обмеження, відображені в концепції гри, основними вимогами були вартість, функціональність і зручність використання. Для мінімізації часу розробки були обрані додаткові інструменти для роботи з ресурсами TexturePacker і з вихідним кодом GitHub. Особливу увагу було приділено адаптації гри під різні пристрої. Графічні ресурси були розділені на три категорії: assets / small, assets / normal і assets / large для зменшення втрати якості зображень при масштабуванні. В результаті, з мінімально можливим збільшенням необхідної пам'яті для гри, вдалося отримати якісне зображення на девайсах з різним дозволом екрану.

Висновок. Результатом виконання завдання розробки мобільного додатку ОС ANDROID комп'ютерної 2D гри стала гра «Скажені кулі» і два незалежних модуля: DebugDraw і PhysicsSystem.

Ключові слова: ANDROID, 2D ГРА, ІНТЕРФЕЙС, ГРАФІЧНИЙ ДВИЖОК, ГРАФІКА, МОБІЛЬНИЙ ПРИСТРІЙ, MICROSOFT VS 2022 , COCOS2D 3.14, BOX2D, TILED, TEXTURE PACKER, GITHUB.

ANNOTATION

Hruzin Roman

Theme: Analysis of methods and technologies for developing a 2D computer game for the ANDROID platform.

Speciality: 121 "Software Engineering".

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2024 year.

Master's work of: 64 p., 36 im, 37 sources.

A research object of: - methods, approaches and information technologies used to develop game software for devices running on the Android platform.

The article of research- processes of creating a two-dimensional game application of the "shooter" genre for the Android platform.

An aim of research is - analysis of methods and technologies for developing a 2D computer game for the ANDROID platform.

Job performanes. Based on sales statistics, a mobile game genre was selected and an assessment of the popularity of the mobile platform under development was given. Statistical data provided by research and consulting companies specializing in information technology markets. Before the design and programming stage, the software was classified by purpose: working with Cocos2d 3.14 graphics, working with Microsoft VS22 code, and working with the Box2d physical world. Given the limitations reflected in the game concept, the main requirements were cost, functionality and usability. To minimize development time, additional tools were chosen to work with TexturePacker resources and GitHub source code. Special attention was paid to the adaptation of the game for different devices. Graphic resources have been divided into three categories: assets / small, assets / normal and assets / large to reduce the loss of image quality when scaling. As a result, with the minimum possible increase in memory required for the game, it was possible to get a high-quality image on devices with different screen resolutions.

Conclusion. The result of the task of developing a mobile application for the ANDROID OS of a 2D computer game was the game "Crazy Bullets" and two independent modules: DebugDraw and PhysicsSystem.

Keywords: ANDROID, 2D GAME, INTERFACE, GRAPHICS ENGINE, GRAPHICS, MOBILE DEVICE, MICROSOFT VS 2022 , COCOS2D 3.14, BOX2D, TILED, TEXTURE PACKER, GITHUB.

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1 ОГЛЯД СТАНУ ПИТАННЯ ТА ДЕТАЛІЗАЦІЯ ЗАДАЧ РОЗРОБКИ	8
1.1. Аналіз мобільної платформи Google Android System	8
1.2. Вибір жанру мобільної гри	13
1.3. Створення концепції мобільної гри	21
1.4. Огляд аналога проєктованої мобільної гри.....	23
1.5. Висновки до розділу	25
РОЗДІЛ 2 ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МОБІЛЬНОГО ДОДАТКУ ОС ANDROID КОМП'ЮТЕРНОЇ 2D ГРИ	26
2.1. Вибір движка для роботи з графікою	27
2.2. Вибір середовища розробки	30
2.3. Вибір фізичного движка.....	33
2.4. Вибір інструментів розробки для роботи з ресурсами	33
2.5. Висновки до розділу	35
РОЗДІЛ 3 ПРОЄКТУВАННЯ ТА КОНСТРУЮВАННЯ 2D ГРИ	36
3.1. Опис інтерфейсу гри.....	37
3.2. Реалізація основного ігрового циклу	41
3.2.1. Змінний тимчасовий крок.....	41
3.2.3. Фільтрація тимчасового кроку.....	45
3.3. Адаптація під різні роздільні здатності мобільних пристроїв	46
3.4. Реалізація ігрового штучного інтелекту	48
3.5. Налаштування.....	54
3.6. Тестування	56
3.7. Підпис гри.....	57
3.8. Висновки до розділу	58
ВИСНОВКИ	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	62
ДОДАТОК А.....	65

ДОДАТОК Б	67
ДОДАТОК В	70
ДОДАТОК Г	74

ВСТУП

Сьогодні смартфони стали невід'ємною частиною життя сучасної людини і задовольняють її найперші потреби. Ці мобільні пристрої надають можливість on-line спілкування та доступу до Інтернету, але їх використання не обмежується лише цими аспектами. Сьогодні смартфони дозволяють навчатися, розважатися, стежити за здоров'ям і почуватися комфортно в світі високих технологій, що покращують якість життя. Крім того, вони допомагають заощадити час, що є надзвичайно важливим. Усе це стає можливим завдяки мобільним застосункам - програмному забезпеченню, розробленому для роботи на мобільних пристроях, таких як смартфони та планшети. Перші мобільні застосунки використовувалися переважно для перевірки електронної пошти, але їх популярність та використання розширилися в різних сферах, включаючи ігри, розваги та бізнес. В залежності від мети проекту розрізняють кілька типів мобільних додатків, включаючи корпоративні, контентні, утиліти та ігри. І саме розширений ринок мобільних додатків, зокрема ігрових, став ключовим аргументом для вибору теми дослідження.

Розробка мобільних додатків для операційної системи Android є одним з популярних напрямків сучасних ІТ-технологій. Запит на портативні пристрої на ринку техніки зростає, тому розробники активно створюють широкий спектр професійних додатків для цих платформ. ОС Android встановлена на мільйонах пристроїв, таких як смартфони, комунікатори, планшети, нетбуки і багато інших. Одним з основних переваг Android є його відкритість та безкоштовність, що дозволяє кожному користувачеві встановлювати різноманітні додатки, включаючи ігри, бізнес-застосунки, програми для спілкування, відеозв'язку, медіа-ресурсів, ЗМІ та онлайн-ресурсів. Вибір додатків майже необмежений. Крім того, розробники надають зручний набір інструментів для створення додатків і добре документоване SDK (Software Development Kit).

Об'єкт дослідження: методи, підходи та інформаційні технології, використовувані для розробки ігрового програмного забезпечення для пристроїв, що працюють на платформі Android.

Предмет дослідження: процеси створення двомірного ігрового застосунку жанру "шутер" для платформи Android.

Мета роботи - аналіз методів і технологій розробки комп'ютерної 2D гри для платформи ANDROID.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. сформулювати концепцію гри;
2. провести аналіз та обрати програмне забезпечення для реалізації комп'ютерної 2D гри;
3. розробити мобільний додаток ОС Android комп'ютерної 2D гри засобами Microsoft Visual Studio.

Методи дослідження - техніко-економічний з використанням комп'ютерних технологій, технічний аналіз, методи моделювання інформаційних процесів.

Перший розділ визначає поняття концепції. На основі статистичних даних по продажах був обраний жанр мобільної гри і дана оцінка популярності мобільної платформи, під яку ведеться розробка. Статистичні дані надані дослідними і консалтинговими компаніями, що спеціалізуються на ринках інформаційних технологій:

- Gartner;
- SurveyMonkey Intelligence;
- Shared2you.

Другий розділ присвячений вибору необхідного програмного забезпечення для розробки гри. При виборі враховувалися ряд вимог до ПЗ і обмеження, які накладає сформульована в першому розділі концепція гри.

У третьому розділі, присвяченому проєктуванню та конструюванню гри, були виділені додаткові підзадачі. При вирішенні таких підзадач, як «реалізація ігрового штучного інтелекту» і «розробка основного ігрового циклу» були порушені аспекти з інших областей знань: робота з векторами і згладжування.

РОЗДІЛ 1

ОГЛЯД СТАНУ ПИТАННЯ ТА ДЕТАЛІЗАЦІЯ ЗАДАЧ РОЗРОБКИ

При створенні ігор на мобільні платформи на початкових етапах проектування необхідно чітко описати концепцію гри. Концепція проекту [1] - його опис, що дає інформацію про те, як в заданих умовах проєкт задовольнить пред'явленим до нього вимогам. Концепція, в свою чергу, розкриває такі поняття, як ігрова механіка (геймплей) - ігровий процес з точки зору гравця і сеттинг (обстановка). Також для розуміння теми необхідно визначити класифікації мобільних ігор за жанрами і по платформі.

Таким чином, перед створенням програми для чіткого формулювання завдання, необхідно дати об'єктивну оцінку ринку двомірних ігор, а саме:

- оцінка популярності жанрів;
- оцінка популярності мобільної платформи Android;
- оцінка позитивних і негативних сторін обраних жанрів;
- оцінка актуальності ігор.

Тобто, потрібно провести аналіз, який дасть можливість вирішити, гру якого жанру найкраще створювати, а також сформулювати чітку концепцію.

1.1. Аналіз мобільної платформи Google Android System

Android - операційна система для смартфонів, планшетів і нетбуків, яку компанія Google придбала у розробника програмного забезпечення Android Inc. у 2005 році. Ця операційна система базується на модифікованому ядрі Linux. Пізніше Google та інші учасники Open Handset Alliance співпрацювали над спільною розробкою цієї нової операційної системи. Для підтримки та подальшого розвитку платформи було створено проєкт Android Open Source Project (AOSP). У Android існує велика спільнота розробників, які розширюють функціональність пристроїв.

Операційна система Android має свій офіційний магазин додатків - Android Market. В ньому представлені як платні, так і безкоштовні програми. На даний момент українським користувачам доступні лише безкоштовні програми та ігри.

Завдяки відкритості Android, користувачі можуть завантажувати додатки із інших джерел.

Розробники під Android переважно використовують мову програмування Java та користуються бібліотеками, розробленими Google, для керування пристроєм.

Офіційно про операційну систему Android стало відомо 5 листопада 2007 року, коли було оголошено створення Open Handset Alliance - консорціуму, що об'єднав 80 компаній. Велика частина коду Android була випущена під ліцензією Apache.

У серпні 2005 року Google придбала компанію Android Inc., після чого Android Inc. стала підрозділом Google. Після покупки Енді Рубін, Річ Майнер і Кріс Уайт залишилися в Android Inc. З'явилися чутки в Інтернеті про те, що Google має намір увійти на ринок мобільних телефонів після придбання Android Inc.

Після отримання підтримки від Google, команда під керівництвом Енді Рубіна почала розробку операційної системи на базі ядра Linux. У грудні 2006 року з'явилися чутки про те, що Google планує випустити смартфон під своїм брендом, відомий як "Гуглофон".

Кожна версія операційної системи Android має назву десерту. Початкові букви назв версій відповідають літерам латинського алфавіту по порядку.

Android Market - це інтернет-магазин додатків для смартфонів, що працюють на операційній системі Android, і є ініціативою альянсу Open Handset Alliance (ОНА), з Google на чолі. Він включає в себе широкий спектр додатків, таких як ігри, клієнти соціальних мереж, офісні програми, новинні додатки, фінансові інструменти та багато інших.

Google оголосила про відкриття цього онлайн-магазину додатків для Android 22 жовтня 2008 року. Спочатку в Україні відсутній був офіційний магазин додатків на смартфонах, але 12 січня 2010 року Samsung Україна повідомила про його запуск, і пізніше Android Market з'явився і у вендорів, таких як Motorola,

HTC, LG і Sony Ericsson, що значно збільшило інтерес до смартфонів на Android в Україні.

Програми для Android розробляються у специфічному байт-кодi для віртуальної машини Dalvik, що є нестандартним для традиційного Java.

Google надає безкоштовно доступний набір інструментів для розробки Android (Android SDK), який можна використовувати на x86-платформах з операційними системами Windows XP, Windows Vista, Mac OS X (версія 10.4.8 або вища) і Linux. Для розробки потрібна версія JDK 5 або JDK 6. Для створення додатків для Android можна використовувати мову Java (не нижче версії Java 1.5). Для розробки на платформі Eclipse доступний плагін "Android Development Tools" (ADT), який сумісний з версіями Eclipse 3.3-3.5. Також існує плагін для IntelliJ IDEA, який спрощує розробку Android-додатків. Наразі розроблено експериментальний плагін для середовища розробки NetBeans IDE.

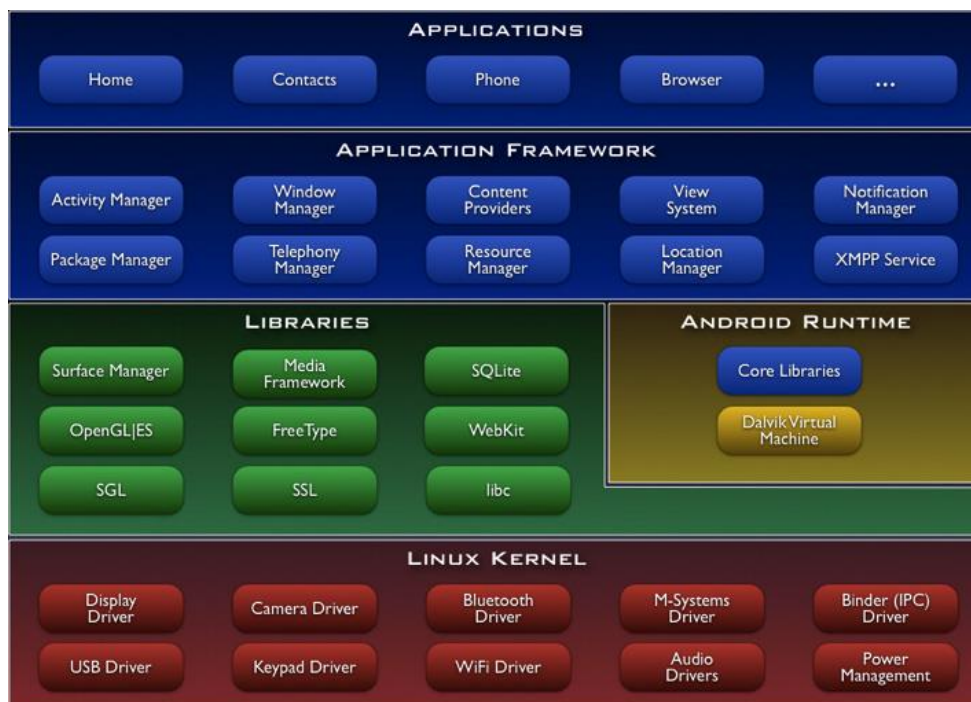


Рис. 1.1. Архітектура OS Android

- Рівень додатків (Applications):

У склад Android включено набір основних додатків, які включають клієнти електронної пошти та SMS, календар, різноманітні карти, браузер, програму для управління контактами та інші функціональність. Всі додатки, які працюють на платформі Android, розроблені на мові програмування Java.

- Рівень каркаса додатків (Application Framework):

У Android існує можливість використовувати повний функціонал API, який використовується в основних додатках. Архітектура платформи побудована таким чином, що будь-який додаток може використовувати вже наявні можливості інших програм, якщо останні надають доступ до своєї функціональності. Таким чином, архітектура платформи сприяє багаторазовому використанню компонентів операційної системи та додатків.

Основою всіх додатків є набір систем і служб.

1. Система уявлень (View System) - це розширений набір ідей, який містить багатий набір функціональності для створення зовнішнього вигляду додатків. Він включає різноманітні компоненти, такі як списки, таблиці, поля введення, кнопки та інші елементи.
2. Контент - провайдери (Content Providers) - це сервіси, які дозволяють додаткам отримувати доступ до даних, що належать іншим додаткам, а також надавати доступ до своїх власних даних.
3. Менеджер ресурсів (Resource Manager) призначений для отримання доступу до різноманітних ресурсів, включаючи текстові, графічні та інші типи.
4. Менеджер сповіщень (Notification Manager) забезпечує можливість будь-якому додатку відображати повідомлення для користувача у рядку статусу.
5. Менеджер дій (Activity Manager) керує життєвим циклом додатків і забезпечує систему навігації, що дозволяє відстежувати історію взаємодії з додатками та їх діями.

- Рівень бібліотек (Libraries):

У платформі Android включено набір бібліотек C/C++, які використовуються різними компонентами операційної системи. Розробникам надається доступ до функцій цих бібліотек через використання Application Framework. Ось кілька прикладів цих бібліотек:

1. System C library - BSD - стандартна системна бібліотека C (libc) для вбудованих пристроїв, які працюють на базі операційної системи Linux, реалізована в платформі Android.
 2. Media Libraries - бібліотеки, які базуються на PacketVideo's OpenCORE, використовуються для забезпечення підтримки програвання та запису популярних аудіо- та відеоформатів, таких як MPEG4, H.264, MP3, AAC, AMR, JPG, PNG та інші.
 3. Surface Manager - менеджер поверхонь управляє доступом до підсистеми відображення 2D - і 3D- графічних шарів.
 4. LibWebCore - сучасний движок web -браузера, який надає всю міць вбудованого Android- браузера.
 5. SGL - движок для роботи з 2D -графікою.
 6. 3D libraries - движок для роботи з 3D - графікою, заснований на OpenGL ES 1.0 API.
 7. FreeType - бібліотека, призначена для роботи зі шрифтами.
 8. SQLite - потужний легковаговий движок для роботи з реляційними БД.
- Рівень середовища виконання (Android Runtime):

Android має в своєму складі набір бібліотек ядра, які надають значну частину функціональності, аналогічну бібліотекам ядра мови Java. У платформі використовується віртуальна машина Dalvik, яка оптимізована для роботи з реєстром, на відміну від стандартної стек-орієнтованої віртуальної машини Java. Кожна програма запускається в окремому процесі з власним екземпляром віртуальної машини. Dalvik використовує формат Dalvik Executable (*.dex), який оптимізований для ефективного використання пам'яті додатками. Це досягається завдяки базовим функціям ядра Linux, таким як організація потокової обробки та низькорівневе керування пам'яттю. Байт-код Java, на якому написані ваші програми, компілюється в формат dex за допомогою утиліти dx, яка входить до складу набору інструментів розробки (SDK).

- Рівень ядра Linux (Linux Kernel):

Android базується на версії 2.6 операційної системи Linux, що дає платформі доступ до різних системних служб ядра, таких як управління пам'яттю і процесами, забезпечення безпеки, робота з мережею і драйверами. Крім того, ядро виступає як шар абстракції між апаратним і програмним забезпеченням.

1.2. Вибір жанру мобільної гри

Зараз існує велика кількість різноманітних комп'ютерних ігор, і їх творці постійно випускають нові. Оскільки створення комп'ютерних ігор пов'язано з розважальною сферою, їх класифікація не така проста, як може здатися на перший погляд. Жанри ігор формувалися без певної структури та на основі інтуїтивного підходу протягом тривалого часу. Розробники ігор проводили сміливі експерименти, досліджуючи нові ігрові механіки. Невдалим експериментам не приділялося багато уваги, а успішні ігри ставали прикладом для інших розробників. Розробники копіювали популярну ігрову механіку, додавали свої ідеї, і таким чином навколо найпопулярніших ігор утворювалися цілі класи схожих ігор, які отримали назву жанрів.

Поділ на жанри є дуже корисним на практиці. Ігри певного жанру вже мають свою зацікавлену аудиторію. Коли розробник оголошує, що випускає гру в певному жанрі, гравці вже мають уявлення про те, що відбуватиметься в грі, навіть без додаткових пояснень від розробників. Часто розробник компанія не може однозначно визначити жанр нової гри, оскільки бажає продати цей продукт якомога більшій кількості користувачів, і тому не хоче обмежувати потенційну аудиторію тематичними рамками.

Зазвичай ігри класифікуються за жанрами і кількістю гравців. Вони поділяються на кілька типів, такі як квести, екшн, рольові ігри (РПГ), файтинги, стратегії, симулятори, логічні та азартні ігри, а також інші. Ця класифікація допомагає гравцям швидше зорієнтуватися в світі ігор. Наприклад, якщо гра відноситься до жанру квесту, гравці можуть очікувати розв'язання головоломок і пошук предметів для продовження сюжету. З іншого боку, якщо гра відноситься до жанру екшну, гравцям може бути цікавим швидкий темп гри та активні битви.

Класифікація також допомагає розробникам ігор визначити свою цільову аудиторію та залучити більше гравців, пропонуючи їм ігровий досвід, який вони вже знають і люблять у певному жанрі.

Отже, поділ ігор на жанри є важливим і корисним інструментом для розуміння та класифікації комп'ютерних ігор. Цей підхід допомагає гравцям знайти ігри, які відповідають їхнім вподобанням, а розробникам — залучити більше гравців шляхом пропозиції ігор у вже визначених жанрах.

Більше того, класифікація ігор за жанрами дозволяє створити зручну систему для сприйняття та обговорення ігрового контенту. Гравці можуть легко спілкуватися та обмінюватися враженнями про ігри, використовуючи загальноприйняті терміни і жанрові визначення. Наприклад, якщо хтось каже про "рогалик", інші гравці відразу розуміють, що йдеться про рольову гру зі случайно генерованими рівнями та постійним розвитком персонажа. Такі загальні терміни та жанрові уявлення сприяють обміну інформацією, порівнянню ігор та вибору нових ігор для гри.

Звісно, варто зазначити, що ігрові жанри не є жорсткими і непорушними категоріями. Багато сучасних ігор поєднують елементи з різних жанрів і використовують гібридні механіки. Це дозволяє розробникам експериментувати та пропонувати нові інновації в ігровій індустрії. Такі гібридні ігри можуть викликати нові жанрові підходи та розширити можливості для створення унікального ігрового досвіду.

Загалом, класифікація ігор за жанрами є важливою складовою розвитку ігрової індустрії. Вона допомагає гравцям зорієнтуватися в широкому виборі ігор та знаходити ті, які відповідають їхнім вподобанням. Розробники ж можуть використовувати жанрову класифікацію для просування своїх ігор та привертання цільової аудиторії.

У більшості випадків ігри класифікуються за жанрами та кількістю гравців. Жанрова класифікація включає квести, екшн, рольові ігри (РПГ), файтинги, стратегії, симулятори, логічні, азартні та інші (зображені на рисунку 1.2).

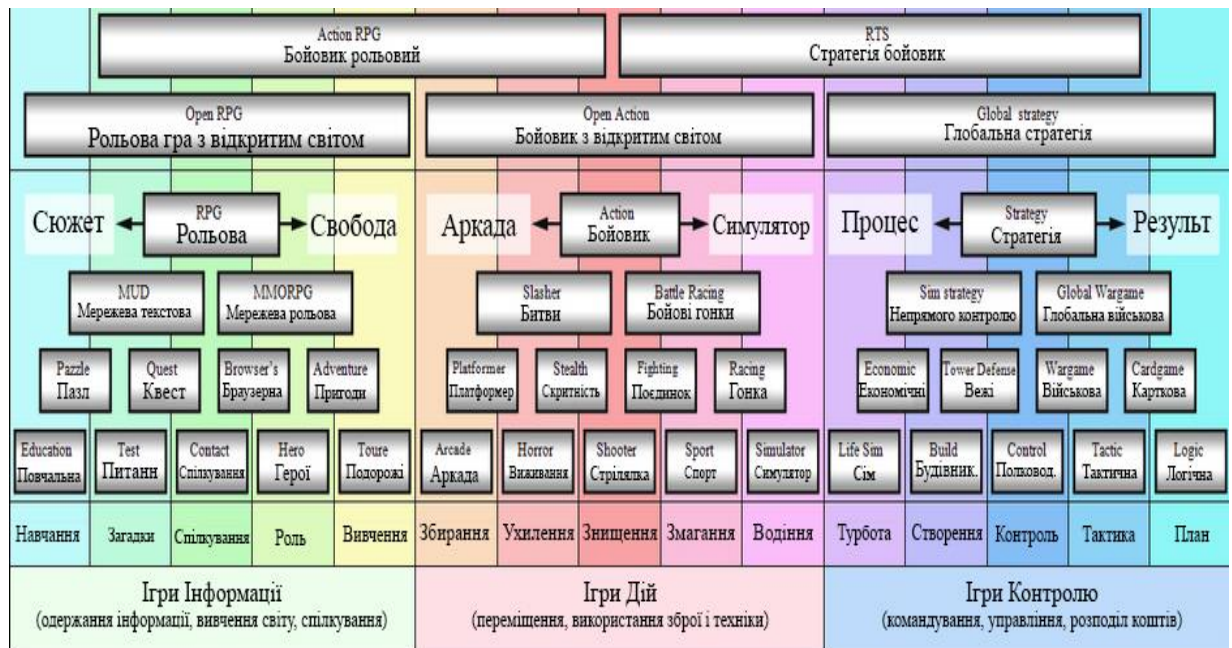


Рис. 1.2. Типи ігор

Розглянемо найпопулярніші жанри ігор, які на сьогоднішній день є широко поширеними:

1. Квести - це ігри, в яких один або кілька персонажів подорожують з метою досягнення поставленої мети, подолання різних труднощів і розв'язання головоломок.



Рис. 1.3. Піджанри квестів

2. Екшн (action) - це популярний жанр, що включає бродилки-стрілялки від першої особи, в яких головний акцент зроблений на швидкість, реакцію та бойові вміння.

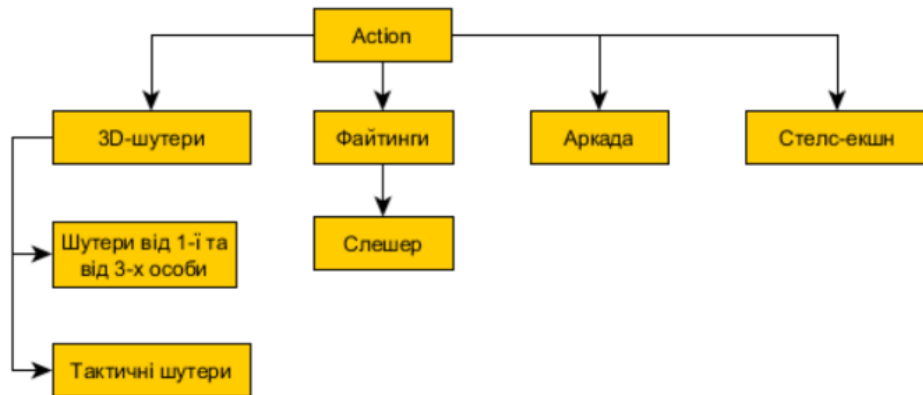


Рис 1.4. Піджанри Action

3. Рольові ігри (РПГ) - в цих іграх гравець приймає роль певного персонажа і виконує поставлені перед ним завдання, розвиваючи його навички, збираючи екіпірування та взаємодіючи з іншими персонажами.

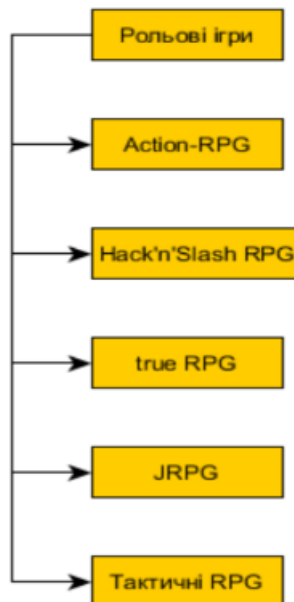


Рис. 1.5. Піджанри рольових ігор

4. Стратегії та логічні ігри - ці ігри наслідують діяльність управлінця, де гравець приймає рішення, розв'язує головоломки та планує свої дії з метою досягнення успіху.



Рис. 1.6. Піджанри стратегії

5. Симулятори - це ігри, що імітують керування автомобілем, космічним кораблем, літаком та іншими об'єктами. Вони спрямовані на реалістичне відтворення певного виду діяльності.



Рис. 1.7. Піджанри Симулятор/Менеджери

6. Азартні і логічні ігри - ці ігри сприяють розвитку розумових навичок і вимагають від гравця логічного мислення та стратегічного планування.

Існує багато платних і безкоштовних ігор, де зустрічаються різні жанри, включаючи екшн, РПГ, деякі види стратегій, квести та інші. У таких іграх присутні різноманітні персонажі та геймплейні елементи, які залежать від конкретної гри. Це дозволяє гравцям знайти ігру, яка найкраще відповідає їхнім інтересам і вподобанням.

Таким чином, незалежно від того, чи це платна чи безкоштовна гра, ігрові жанри пропонують різноманітність варіантів для гравців. Вони можуть обрати екшн-пригоди, в яких досліджують світ та борються з ворогами, або погрузитися у глибокий сюжет РПГ і відчувати себе героєм фантастичного світу. Вибір жанру допомагає гравцям зорієнтуватися в світі ігор та знаходити ігри, які найбільше їх захоплюють.

Ігри без персонажа, такі як логічні і симулятори, відрізняються відсутністю сюжету та не викликають сильної емоційної залежності або впливу на психіку гравця. Ці типи ігор часто пропонують інтелектуальні завдання, що сприяють розвитку мислення.

У стратегічних іграх також відсутні персонажі, а гравець управляє грою зі свого обличчя. Однак, особливістю стратегій є система внутрішніх факторів, що регулюють геймплей. Гравець приймає рішення та спостерігає за їх наслідками, побудовуючи логіку подальших дій.

Таким чином, ігри без персонажа, такі як логічні і симулятори, пропонують інтелектуальні завдання, сприяючи розвитку мислення, тоді як стратегічні ігри з відсутністю персонажа мають особливу систему внутрішньоігрових факторів, де гравець приймає рішення та будує логіку своїх дій.

Рольові ігри (РПГ), а також окремі екшн і квести, є найвідомішими іграми, де гравцеві надається свобода вибору. Гравець самостійно обирає персонажа, одягає його та наділяє різними здібностями. Головна особливість цих ігор полягає в тому, що гравець має велику свободу дій, які впливають на хід та результат гри. Грати в такий тип ігор може викликати стійке захоплення, оскільки гравець співпереживає своєму персонажу, розвиває його відповідно до своїх уподобань і стає активним учасником ігрового світу. Ігровий процес сприймається як

реальність, а багатьом РПГ-іграм подібність з фантастичними книгами з можливістю керування головним героєм становить особливий інтерес для любителів читання. Захоплення такими іграми можна порівняти з пристрастю до книг, воно не має шкідливих наслідків, а навпаки, сприяє розвитку фантазії, захоплює пізнанням історій та відображенням реальності.

Ігри з лінійним сюжетом, такі як стрілялки, не передбачають вибору гравця. Гравець ідентифікується з готовим персонажем, який вже має визначені характеристики. Розвиток сюжету в таких іграх залежить не від вибору гравця, а від успіху або випадковості. Деякі стрілялки дозволяють грати командою. Такі ігри надають емоційне розслаблення, а успішне проходження часто залежить від швидкої реакції гравця.

На перетині рольових ігор і стратегій знаходиться поняття "абстракція". Це протилежність "бойовикам", які борються зі скукою за допомогою спектакулярності. Ігри, що належать до цієї категорії, мають схематичний характер, обмежену свободу, відсутність реалістичності та візуального шоу.

У стратегічних абстракціях головним завданням є досягнення "цілі", тоді як у рольових абстракціях важливим є "підпорядкування". В результаті отримуємо ситуацію, де гравець підкоряється чужим цілям і виконує нудну роботу. При цьому можуть відбуватися події, які раніше не траплялися, але все одно гравець змушений підкорятися волі інших (це основна відмінність від "буденності"). Центральним елементом є поняття рабства і безумовне виконання наказів. Такі аспекти не мають нічого спільного з грою.

Склад відеоігрової палітри "Буденність" передбачає поєднання елементів стратегій і бойовиків. На перетині цих жанрів знаходиться поняття "симуляція" (див. "рисунок 1.8"). Це повна протилежність "рольовим іграм", які протистоять "буденності" за допомогою цікавого "сюжету".

Ігри, що належать до цієї категорії, характеризуються імітацією реальних процесів, обмеженою свободою, відсутністю новизни і повністю відсутнім сюжетом.



Рис. 1.8. Межа факторів жанру ігор

З погляду стратегій у симуляціях найбільш важливим є сам "процес". З боку бойовиків наголошується на "реалістичності", яка відтворюється на екрані. В результаті отримуємо реалістичну імітацію процесу, яка відображає звичайне життя. В грі існує кілька повторюваних подій, але нічого нового не з'являється. Центральний елемент не використовується, оскільки повна реалістичність не досяжна у відеоіграх, а також багато хто не шукає точну копію реальності в грі, яка вже існує у реальному житті.

На перетині рольових ігор і бойовиків розташовується концепт "свобода". Це протилежність стратегіям, які, в свою чергу, наголошують на "самотності" та "керівництві" над численними підлеглими. (Варто зауважити, що стратегії не борються з самотністю в звичайному розумінні, але вони відтворюють владу як найпоширенішу форму спілкування).

Для ігор, які знаходяться на цьому перетині, характерні великі відкриті світи, відсутність повної симуляції реальності, відсутність новизни та відсутність контролю над іншими учасниками.

З боку рольових ігор в "іграх свободи" основним є відсутність готових рольових зв'язків та можливість самотійного вибору, тобто "рольова свобода".

З боку бойовиків спостерігається "спрощення" реальних подій та процесів у порівнянні з симуляторами.

Узагальнюючи, результатом є рольова свобода в спрощеній моделі реального світу. Свобода однієї особи завершується там, де починається свобода

іншої, тому максимальна точка свободи - це повна самотність. Найбільш спрощена точка виявляється у порожньому світі, де є свобода простору. Найбільш вільною дією є створення чогось абсолютно нового. Загалом, центральною точкою цього елемента є повна самотність у порожньому світі, де можна творити - редактор рівнів. Прямо під час гри можливості редагування не використовуються, але окремі елементи процесу створення присутні в іграх.

За допомогою цього шестикутника легко розуміти найважливіші аспекти певних видів ігор та компроміси, з якими розробники ігор стикаються. Він також показує, що є недопустимим у іграх і що можна комбінувати між собою, а що не піддається змішанню.

Для згладжування переходів між гранями, розробники повинні ретельно прорахувати геймплей - сам процес гри з погляду гравця, який є основою зацікавленості та привабливості гри. Геймплей включає різні аспекти, включаючи технічні аспекти, такі як внутрішньоігрова механіка, методи взаємодії між "штучним інтелектом гри" та гравцем тощо. Однак, сам термін "геймплей" є загальним та зазвичай використовується для опису відчуттів, отриманих під час гри, під впливом таких факторів, як графіка, звук та сюжет. Тому пріоритетом є прорахування моделі поведінки, що є фундаментальним для жанру гри, а потім вдосконалювати та поліпшувати взаємодію з гравцем.

1.3. Створення концепції мобільної гри

Концепція зазвичай включає опис цільової аудиторії, сеттинга і жанру гри, ключових (можливо, унікальних) особливостей геймплея, методик розробки, інструментарію і технологічних рішень, ринків збуту, підтримуваних платформ. Одним з ключових факторів успіху на даному етапі є аналіз поточної ситуації на ринку - відстеження існуючих трендів, огляд найбільш успішних проєктів, вибір нових інструментів розробки для освоєння.

Виходячи з жанру, особливості гри і можливостей розробника, було обрано назву - «Скажені кулі» (The Crazy Bullets). Дана назва відображає унікальну особливість гри. Акцент робиться на реакцію гравця: як швидко він зможе впоратися з кулями, що летять в нього. А впоратися можна двома способами -

ухилитися або відбити своїм пострілом, змінивши траєкторію польоту кулі. Для простоти розробки була обрана двовимірна графіка, таким чином, це двовірний (2D) шутер з видом зверху на платформу Android в режимі одного.

Концепт-арт (Concept design [4]) героя представлено на рисунках 1.9 та 1.10.

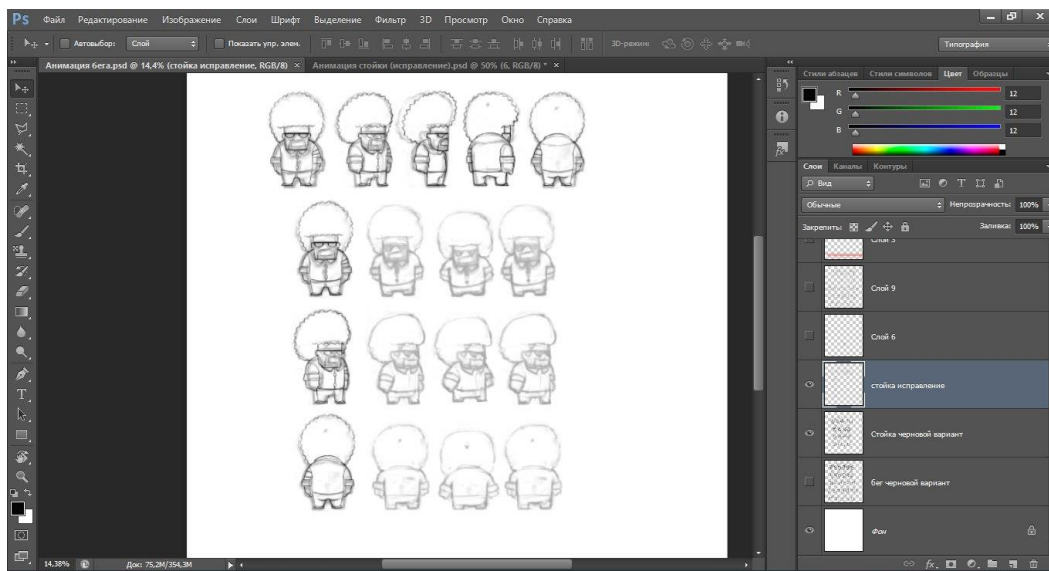


Рис. 1.9. Моделювання головного героя



Рис. 1.10. Біг вниз

Спираючись на готовий концепт-арт героя і обраний жанр, можна сформулювати історію головного героя: «Герой - брутальний чоловік в самому розквіті своїх сил - став на стежку війни. Необхідно допомогти герою перемогти нещадного повелителя арени». Основна ідея гри - протистояння гравця та ігрового штучного інтелекту (ІІ).

При виборі ігрової механіки враховувалася складність програмування, можливості розробника і дизайнера. Унікальною рисою геймплея буде добре спроектований ігровий ІІІ.

Управління буде досить простим. З огляду на адаптивність гри до різної роздільної здатності та глибини екрана, необхідно вибрати зручний варіант управління героєм, як для планшета, так і для мобільного телефону. Зразкове управління героєм представлено на рисунку 1.11.

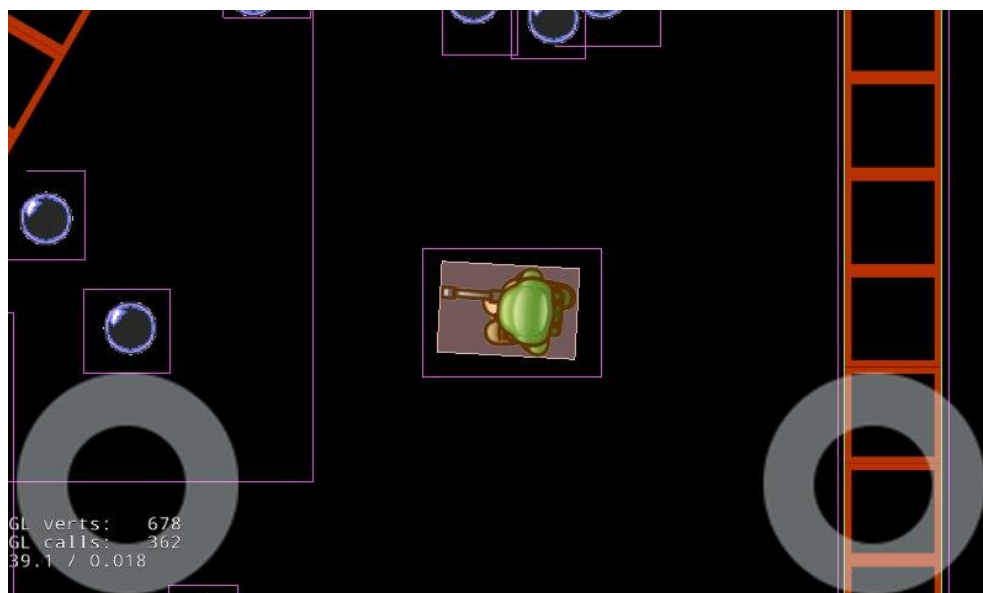


Рис. 1.11. Управління героєм

Ліва окружність відповідає за рух персонажа. При натисканні на нього, обчислюється вектор швидкості, як різниця між точкою дотику і центром кола, права - за стрілянину. При натисканні на праве коло герой стріляє.

Характеризуючи цільову аудиторію, перш за все, потрібно чітко визначити її класифікацію (рис. 1.12). З огляду на жанр і простоту гри, вона орієнтована на казуальну аудиторію для приємного проведення часу.

Параметр	Казуальний		Мідкорний		Хардкорний	
	м	ж	м	ж	м	ж
Співвідношення	40%	60%	55%	45%	72%	28%
Вік	-18	-35	-25	-28	-20	-25
Пристрої	Смартфони/ приставки		Смартфони/ приставки		Смартфони/ приставки	
Щоденно грають	20-60 хвилин		30-120 хвилин		120-300 хвилин	
Мета гри	Гаяти час		Гаяти час та задоволення		Задоволення та покращення навиків	
Жанри	- азартні - настольні	- пригоди - головоломки	- РПГ - екшн	- стратегії - ферми	-ММО - шутери - перегони	-РПГ - екшн -ММО

Рис. 1.12. Класифікація цільової аудиторії

1.4. Огляд аналога проєктованої мобільної гри

На сьогоднішній день існує велика кількість ігор жанру шутер. У контексті даної роботи буде розглянуто один з найпопулярніших аналогів на майданчику Google Play: Age of Zombies від Halfbrick Studios.

У даній грі розробники сконструювали цікаву модель, засновану на красивій графіці, динамічній ігровій механіці, частою зміною сцен. Гра

складається з величезної кількості рівнів, кожен з яких досить різноманітний. Щоб урізноманітнити гру, основним жанром був обраний жанр «пригода». Це і можна вважати особливістю даної гри.

Гра «Age of Zombies» (рис. 1.8) платна. Таким чином, плюси гри:

- приємна деталізована графіка;
- відмінна ігрова механіка;
- дуже динамічна;
- цікава атмосфера;
- інтуїтивне управління;
- хороший сюжет;
- можливість розвивати героя.

мінуси:

- тільки платна версія;
- занадто простий ігровий ШІ.



Рис. 1.13. Гра «Age of Zombies»

Виходячи з плюсів і мінусів даної гри, можна знайти слабкі місця програми, за рахунок яких можна з ним конкурувати. Такими недоліками є відсутність мультиплеєра і слабкі ІІ, що компенсується їх кількістю.

Таким чином, були проаналізовані статистичні дані:

- популярності мобільних платформ - найпопулярнішими платформами в 2022 року виявилися: Android і iOS;

- популярності жанром ігор для смартфонів - найпопулярнішими жанрами виявилися: аркади, головоломки, стрілялки.

Виходячи з аналізу статистичних даних, представлених провідними консалтинговими компаніями, був обраний жанр - двомірний шутер з видом зверху. Так само була сформульована і представлена концепція гри.

1.5. Висновки до розділу

Перший розділ «аналіз предметної області» визначає поняття концепції. На основі статистичних даних по продажах був обраний жанр мобільної гри і дана оцінка популярності мобільної платформи, під яку ведеться розробка. Статистичні дані надані дослідними і консалтинговими компаніями, що спеціалізуються на ринках інформаційних технологій:

- Gartner;
- SurveyMonkey Intelligence;
- Shared2you.

РОЗДІЛ 2

ВИБІР ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МОБІЛЬНОГО ДОДАТКУ ОС ANDROID КОМП'ЮТЕРНОЇ 2D ГРИ

Вибір програмного забезпечення (ПЗ) є важливою складовою перших етапів розробки проєкту. Вибір якісного продукту може значно зменшити зусилля програміста. Як правило, найбільш важливі вимоги, які розглядаються при виборі системи, це:

- призначений для користувача інтерфейс;
- управління даними;
- механізм планування;
- забезпечення спільної роботи.

Для аналізу програмного забезпечення (ПЗ) необхідно розкрити таке поняття як ліцензія MIT. Ліцензія MIT - ліцензія відкритого програмного забезпечення, розроблена Массачусетським технологічним інститутом (МТІ). Ліцензія MIT є однією з найбільш ранніх вільних ліцензій, так як вона відносно проста і ілюструє деякі з основних принципів вільного ліцензування [5]. Вона є дозвільної ліцензією, тобто дозволяє програмістам використовувати ліцензований код в закритому ПЗ за умови, що текст ліцензії надається разом з цим ПЗ. На думку Free Software Foundation, дана ліцензія більш точно називається X11 License, так як в минулому MIT використовував багато ліцензій, і в поточному вигляді вона була написана для X Window System. Використання ПЗ або готових рішень з цією ліцензією не збільшує кінцеву вартість продукту, що розробляється. Велика частина використовуваного ПЗ в розробці даної мобільної гри буде поширюватися саме під цією ліцензією.

Для розробки ігор, програмісти використовують готові рішення - фреймворк (framework). Фреймворк (framework) - програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проєкту. Основною особливістю фреймворка є сильний вплив на архітектуру програми.

Більш об'ємним і функціональним засобом розробки є движок. На відміну від фреймворка, який полегшує роботу програмісту, движок «робить її за нього».

Вибір програмного забезпечення є важливою складовою перших етапів розробки проекту. На вибір ПЗ вплинули такі фактори як:

- вартість і якість ПЗ;
- функціональність ПЗ, тобто здатність вирішувати покладені на нього завдання;
- зручність використання, наприклад, наявність потужної технології автодоповнення в середовищі розробки, що впливає на час розробки;
- різні технічні характеристики.

ПЗ класифіковано за призначенням, для спрощення вибору:

- для роботи з графікою;
- для роботи з кодом;
- для роботи з фізичним світом;
- для роботи з проектом.

2.1. Вибір движка для роботи з графікою

З урахуванням обраної платформи і мови програмування, вибір між середовищами і засобами розробки значно звузився. Для роботи з графікою можна виділити наступні інструменти: cocos2dx, Marmelade sdk і Unreal Engine 4 (UE4).

У 2016 році Marmelade мав безкоштовну Community ліцензію, незначно обмежуючи програміста в наборі використовуваних засобів. Marmelade SDK мав свій збирач проекту, схожий на значно спрощену версію CMake [6], і докладну документацію. Так само плюсом була наявність додаткових інструментів для редагування карт, симулятора девайсів і проста організаційна структура розробляється. Але в 31 березня 2017 року Marmelade був викуплений японською компанією, і ліцензія Community зникла. Вартість ліцензії інді-розробника - 600 \$ в рік.

Відмінною альтернативою для Marmelade SDK став фреймворк cocos2dx (рис. 2.1). Cocos2dx - це крос-платформенний фреймворк з відкритим кодом [7].

Cocos2d-x дозволяє розробляти ігри для різних платформ, включаючи iOS, Android, Windows, Mac OS X і Linux, використовуючи один і той же кодову базу. Він підтримує мови програмування C++, JavaScript і Lua, що дозволяє розробникам використовувати свою улюблену мову для створення ігор.

Основні особливості Cocos2d-x включають:

1. Високоякісна 2D-графіка: Cocos2d-x має потужні засоби для роботи з 2D-графікою, включаючи підтримку спрайтів, анімації, частинок та ефектів.
2. Розширені можливості фізики: Фреймворк включає фізичний двигун Box2D, що дозволяє створювати реалістичні фізичні ефекти в іграх.
3. Аудіо та відео: Cocos2d-x підтримує відтворення аудіо та відео файлів, що дозволяє створювати звукові ефекти та відеоінтеграцію в ігри.
4. Розширення: Є багато розширень, які розробники можуть використовувати для розширення функціональності фреймворку, такі як робота з рекламою, соціальними мережами, аналітикою тощо.

Cocos2d-x є популярним вибором серед розробників ігор, особливо тих, хто працює над кросплатформовими проєктами, оскільки він дозволяє ефективно використовувати код і ресурси між різними платформами. Окрім того, він має велику спільноту розробників, яка надає підтримку, документацію та приклади коду.

Основні переваги використання Cocos2d-x включають:

- Кросплатформовість: Ви можете розробляти ігри для різних платформ, використовуючи один і той же код. Це зменшує затрати часу та зусиль при портуванні гри на різні платформи.
- Швидкість та ефективність: Cocos2d-x базується на C++, що дозволяє отримати високу продуктивність та ефективність. Він оптимізований

для роботи з графікою та обробкою подій, що забезпечує плавну роботу ігор.

- Розширені можливості: Фреймворк має багатий набір функціональності, такий як анімація, фізика, аудіо, частинки, робота з мережею та багато іншого. Ви можете створити різноманітні типи ігор, від простих аркадних ігор до складних стратегічних проєктів.
- Велика спільнота: Існує активна спільнота розробників Cocos2d-x, яка надає підтримку, документацію та приклади коду. Ви можете отримати відповіді на свої питання, поділитися досвідом та отримати допомогу вирішення проблем.

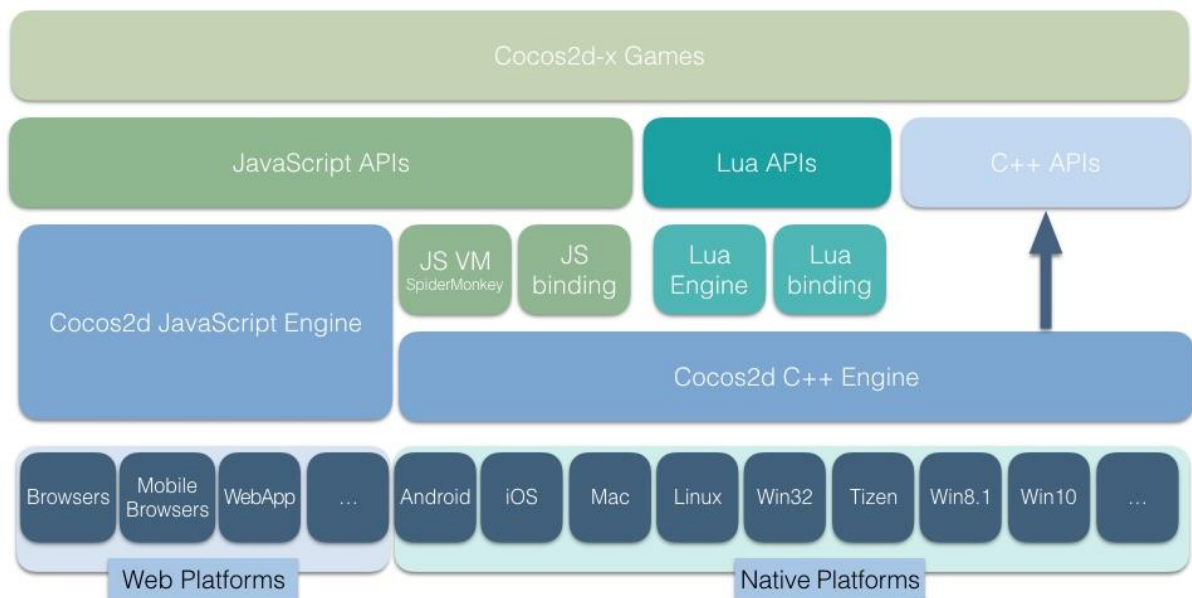


Рис. 2.1. Архітектура cocos2dx

Перевагою розробки за допомогою UE4 є наявність величезної кількості інструментів і шаблонів, які набагато простіше використовувати, ніж фреймворк. UE4 дає можливість створювати гри не програмуючи, а маніпулюючи ресурсами, сутностями і зв'язками (рис. 2.2): Blueprint - редактор візуального скриптинга [8]. Технічно не потрібно писати жодного рядка коду. Це дуже зручно для створення швидких прототипів або ігор. Також 2 березня 2015 року Unreal Engine 4 став безкоштовним, але розробники ігор, повинні передавати 5% від прибутку гри компанії Epic Games, за умови, що доходи гри складають більше \$ 3000 за квартал.

.NET. Це середовище можна використовувати як інтегроване середовище розробки, так і в якості окремих засобів. Visual C++ складається з наступних компонентів:

- **Компілятор C++:** Visual C++ постачається з компілятором, який перетворює код на мові C++ у машинний код, який може бути виконаний операційною системою.
- **Інтегроване середовище розробки (IDE):** Visual C++ має потужне інтегроване середовище розробки (IDE) під назвою Visual Studio. Це середовище надає розширені можливості для розробки, налагодження, тестування та керування проєктами на мові C++. Воно має інтерфейс користувача з багатьма функціями, такими як редактор коду, відладчик, система контролю версій, інструменти тестування та багато іншого.
- **Бібліотеки Windows API:** Visual C++ надає доступ до багатьох бібліотек Windows API, які дозволяють розробникам взаємодіяти з операційною системою Windows. Ці бібліотеки надають функції для керування вікнами, обробки подій, мережевого взаємодії, роботи з файлами та багато іншого.
- **Бібліотеки стандарту C++:** Visual C++ включає бібліотеки стандарту C++, такі як STL (Standard Template Library), які надають реалізації різних алгоритмів, контейнерів та інших корисних компонентів. Ці бібліотеки полегшують розробку програм на мові C++ і забезпечують переносимість коду між різними платформами.
- **Інструменти для налагодження та профілювання:** Visual C++ надає різноманітні інструменти для налагодження та профілювання програм. Це включає точковий налагоджувач (debugger) для виявлення та виправлення помилок, аналізатор пам'яті для виявлення витоків пам'яті.

- **Пакети розробки (SDK):** Visual C++ має різноманітні пакети розробки (SDK), які дозволяють розробникам створювати програми для конкретних платформ або функціональностей. Наприклад, Windows SDK надає набір інструментів та бібліотек для розробки програм під операційну систему Windows.
- **Інструменти для розробки графічного інтерфейсу:** Visual C++ має набір інструментів для розробки графічного інтерфейсу користувача. Включаючи дизайнер форм, який дозволяє створювати і налаштовувати вікна, кнопки, поля введення та інші елементи інтерфейсу.

Ці компоненти разом створюють потужне середовище розробки для програм на мові C++, дозволяючи розробникам створювати різні типи програм з використанням широкого набору інструментів та бібліотек.

Мова C++, що є найпопулярнішою у світі мовою рівня системи, і Visual C++ разом надають розробникові висококласний засіб світового рівня для побудови програмного забезпечення.

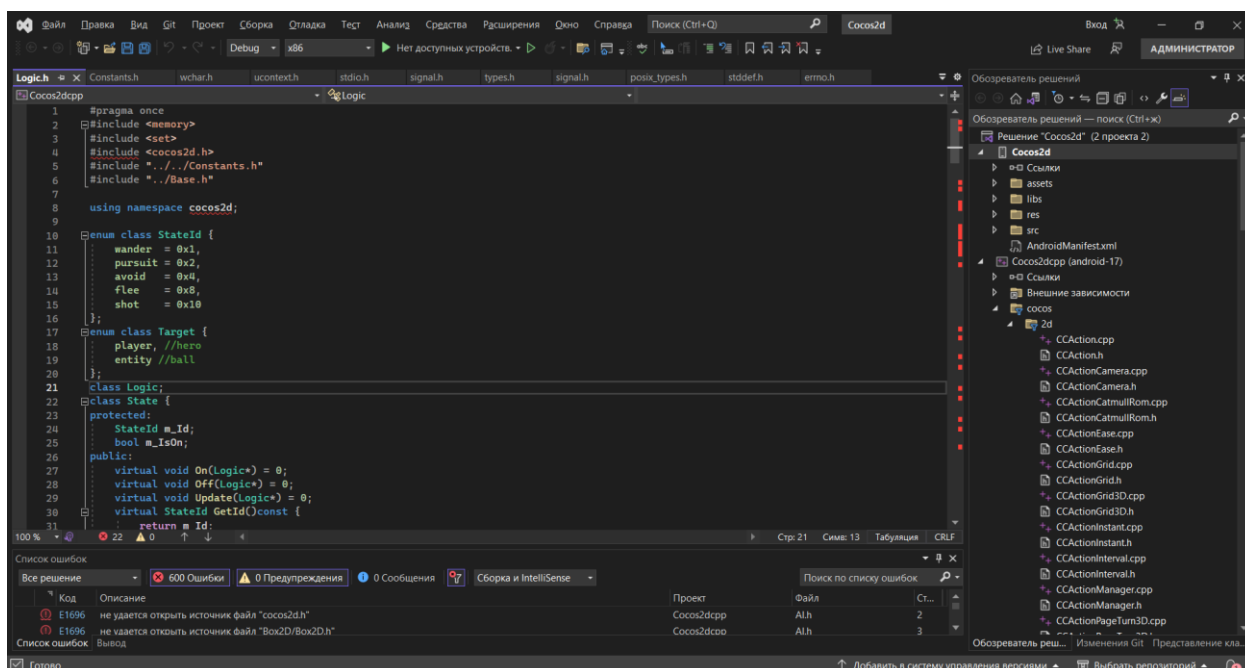


Рис. 2.3. Microsoft Visual Studio 2022

Програма реалізована на мові програмування C++ в середовищі Visual C++ 2022. Вибір середовища програмування обумовлений відносною простотою реалізації графічного інтерфейсу користувача (GraphicUserInterface – GUI).

2.3. Вибір фізичного движка

Так як унікальністю гри є симуляція ідеально пружних зіткнень куль, необхідно використовувати фізичний движок. Фізичний движок (physics engine [11]) - комп'ютерна програма, яка виробляє комп'ютерне моделювання фізичних законів реального світу в віртуальному світі, з тим або іншим ступенем апроксимації. Основними вимогами до вибору движка були: безкоштовна ліцензія і наявність необхідної функціональності для симуляції зіткнень.

Існує два легко інтегруючі в cocos2dx движки, це Chipmunk [12] і Box2d [13]. Обидва движки є безкоштовними, з відкритим кодом, швидкими і прості у використанні, в порівнянні з 3D двигунами. Також, обидва підтримують промальовування в режимі налагодження. Проте, у Chipmunk є один недолік: при досягненні об'єктом певної швидкості руху, не виключено, коли він буде «проходити» крізь інші об'єкти. Для усунення цього ризику був обраний Box2d [14].

2.4. Вибір інструментів розробки для роботи з ресурсами

Для створення простих карт з одного або групи спрайтів, був обраний Tiled Map Editor (рис. 2.4) з безкоштовною ліцензією. Програма дає можливість створювати карти з «плиток» та отримувати на виході файл формату tmx, який підтримується cocos2dx.

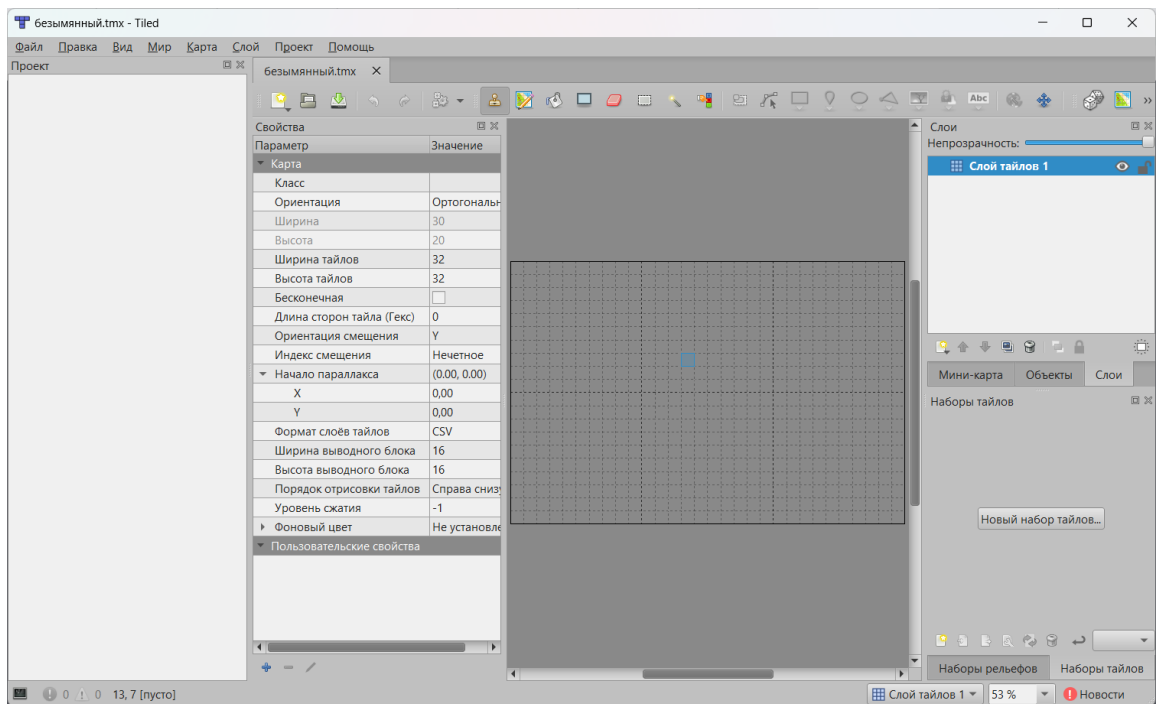


Рис. 2.4. Интерфейс Tiled Map Editor

Для отримання файлу plist для зручного створення анімації був використаний Texture Packer (рис. 2.5). Дана програма містить дві ліцензії - безкоштовна і платна (pro). Основною відмінністю ліцензій є використання додаткових алгоритмів для скорочення кількості полігонів, необхідних cocos2dx для промальовування спрайту. Для завдання даної роботи досить безкоштовного режиму.

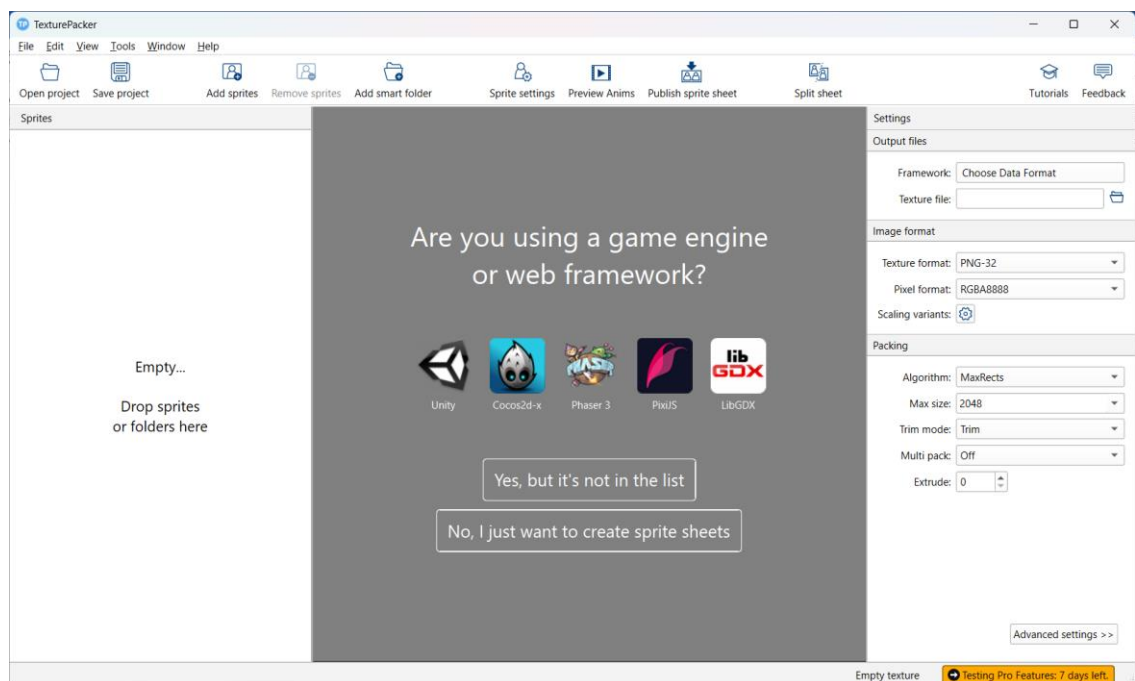


Рис. 2.5. Texture Packer

Для полегшення роботи з кодом і можливості стежити за його зміною, необхідно використовувати засіб контролю версій. Система контролю версій (СКВ) - це система, яка реєструє зміни в одному або декількох файлах з тим, щоб надалі була можливість повернутися до певних попередніх його версій цих файлів. СКВ Git [15; 16] є добре документованої, популярної, функціональної та безкоштовною системою контролю версій з відкритим вихідним кодом.

Таким чином, для розробки програми використовуються такі інструменти, як Microsoft VS 2022 року, Cocos2d 3.14, Box2d, Tiled, Texture Packer, GitHub. Все програмне забезпечення є безкоштовним або використовується лише його обмежений функціонал, який не потребує покупки платній версії продукту.

2.5. Висновки до розділу

Другий розділ присвячений вибору необхідного програмного забезпечення для розробки гри. При виборі враховувалися ряд вимог до ПЗ і обмеження, які накладає сформульована в першому розділі концепція гри.

РОЗДІЛ 3

ПРОЄКТУВАННЯ ТА КОНСТРУЮВАННЯ 2D ГРИ

При розробці програми для спрощення читання коду і його масштабованості були використані патерни проєктування. Патерн проєктування в розробці програмного забезпечення - повторювана архітектурна конструкція, що представляє собою рішення проблеми проєктування в рамках деякого контексту [17; 18; 19].

Для розширення функціональності класу b2World движка Box2d без внесення змін до його коду, був використаний патерн проєктування декоратор [18] (Decorator). Патерн декоратор динамічно додає нові обов'язки об'єкту. Декоратори є гнучкою альтернативою породження підкласів для розширення функціональності. Прикладом його використання є клас-обгортка PhysicsSystem. Він дає можливість проаналізувати різні підходи до вирішення проблеми руху персонажа і синхронізації фізичного світу з графічної складової.

Використання патерну проєктування Сінглтон (Singleton) необхідно для можливості доступу до ресурсів з будь-якої точки системи. Для чистоти коду та глобальної видимості, відповідно до рекомендацій Скотта Майерса [20] і Стіва Макконнелла [21], використання даного патерну проєктування зводиться до мінімуму.

Також одним із способів контролю ігрового світу, є параметр FPS (Frames Per Second) - кількість кадрів в секунду на екрані, які видаються програмним забезпеченням відеокарти. Розкриття даного визначення відіграє значну роль при вирішенні проблеми неплавного, переривчастого руху героя.

При проєктуванні системи були використані UML-діаграми (Unified Modified Language за стандартом OMG 2015 года [22]) станів (State Machine diagram) - діаграма станів показує, як об'єкт переходить з одного стану в інший і діаграма діяльності (Activity diagram) - діаграма, на якій показані дії. Під діяльністю розуміється специфікація виконуваного поведінки у вигляді координованого послідовного і паралельного виконання підлеглих елементів.

Під час розробки гри були виділені додаткові підзадачі:

- проектування інтерфейсу;
- згладжування синхронізації фізичного і графічного движка;
- адаптація гри під різні дозволи пристроїв;
- проектування ігрового інтелекту;
- налагодження;
- тестування.

Підхід до розробки був обраний ітеративний. Робота виконувалася з безперервним аналізом отриманих результатів і поверненням до попередніх етапів.

3.1. Опис інтерфейсу гри

Інтерфейс гри складається з 9 сцен. Діаграма зв'язків і переходів сцен представлена на рисунку 3.1 UML діаграмою станів.

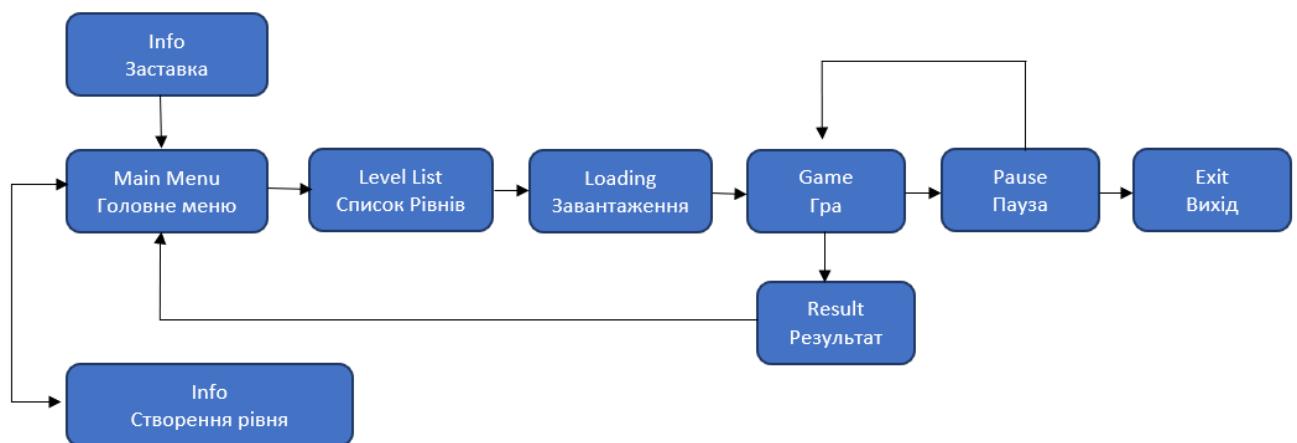


Рис. 3.1. Діаграма станів гри

Ключові стани розробленої гри представлені на рисунках 3.2 - 3.5. Головне меню (рис. 3.2) складається з трьох кнопок: почати гру (Play), побудувати (Build) і вийти (Exit).



Рис. 3.2. Каркас головного меню

Кнопка «вийти» (Exit) дає можливість вийти з гри. Гравець перейде до режиму будівництва карти при натисканні на кнопку «побудувати» (Build). На даний момент режим «будівництва» дуже сирий і потребує доопрацювання. Кнопка «почати гру» (Play) дозволяє перейти на наступну сцену вибору рівня (рис. 3.3). Інтерфейс меню вибору рівня складається з двох стрілок з боків, для перегортання рівнів, опис самого рівня і дві кнопки: назад - повернення в головне меню і вперед - перехід до стадії гри (вибір поточного рівня). Текст на слайді передається з файлу json формату, що описує рівень.



Рис. 3.3. Каркас меню вибору рівня

Перед переходом до гри, підключається додатковий потік, який завантажує ресурси: великі зображення і самі уявлення рівня в json форматі. Під час завантаження ресурсів, на екрані з'явиться вікно завантаження (рис 3.4). Необхідність попереднього завантаження ресурсів в оперативну пам'ять обумовлена тим, що на швидко працюючих пристроях текстури можуть завантажитися повільніше, ніж GPU обробить виклики графічних функцій.

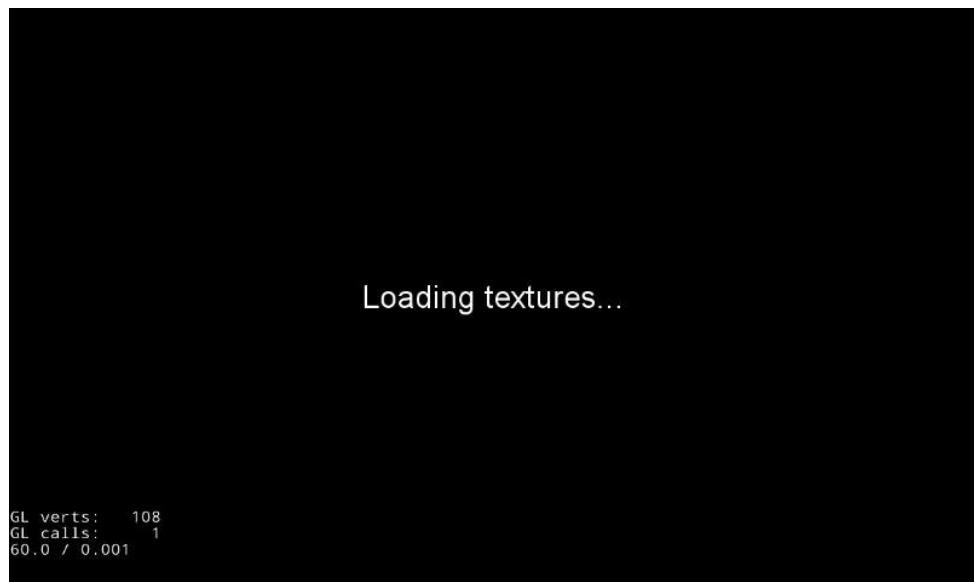


Рис. 3.4. Екран завантаження

Інтерфейс самої гри (рис. 3.5) складається з:

- індикатора життів у вигляді 4 сердець - вони зникають при отриманні шкоди;
- керованого героя, представленого набором анімації;
- двох сенсорів управління: лівий - керування рухом, правий - стріляниною.

Також елементами гри є стіни, кулі (кулі), приціл - вказує напрямок стрільби і смуга перезарядки - з'являється після пострілу. Карта представлена групою однакових текстур.



Рис. 3.5. Скріншот з гри

Керований герой представлений 8 групами спрайтів, де кожна група представляє анімацію руху в одну з 8 сторін. На рисунку 3.6 представлений чорновий варіант анімації героя. На даний момент в гру не введено зброю на рівні графічної частини. Зброю замінює червоний приціл, на місці якого з'являються кулі.

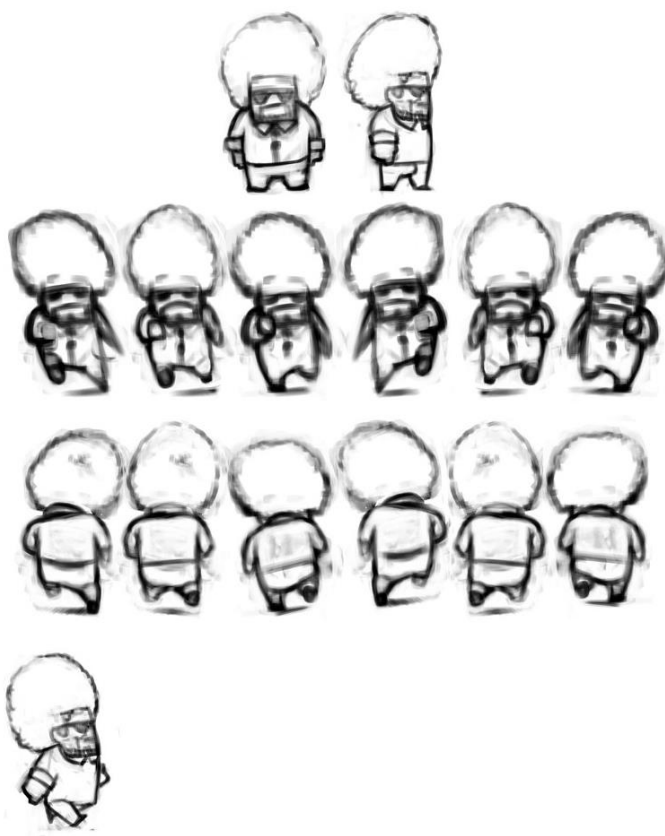


Рис. 3.6. Чорновий варіант анімації руху героя

3.2. Реалізація основного ігрового циклу

Ігровий цикл - принцип, згідно з яким ігрові дизайнери задають головний елемент ігрової механіки, який визначає фундаментальний досвід гравця. Один ігровий цикл являє собою дію гравця, результат цієї дії в ігровому світі, реакцію гравця на результат і запит гри на повторення нової дії [23].

При додаванні фізичного движка Box2d і синхронізації його з графічною частиною виникла проблема - рух персонажа став переривчастим. Причиною проблеми є різний час, що надходить на вхід для поновлення кожного кроку фізичного движка. На рисунку 3.7 представлений графік, що відображає очікуваний час для поновлення і фактичний.

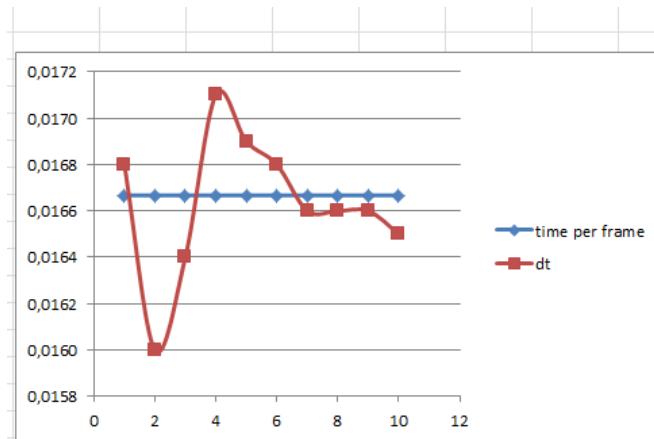


Рис. 3.7. Графік очікуваного і вхідного часу для поновлення ігрового світу

Для вирішення завдання був реалізований клас `PhysicsSystem`, представлений в додатку В, як патерн проектування декоратор.

3.2.1. Змінний тимчасовий крок

До того, як виникла проблема синхронізації фізичного і графічного движка, було використано найпростіше рішення - використання змінного тимчасового кроку.

Лістинг 3.1 - Псевдокод методу поновлення ігрового циклу зі змінним кроком.

```
// time_to_consume - час, який пройшов з попереднього оновлення
// оновити фізичну частину
update_physics(time_to_consume);
// оновити графічну частину
```

```
update_graphics_with_physics(time_to_consume, physics);
```

Позитивними сторонами даного підходу є:

- все передане для поновлення часу (time_to_consume) використано, що дозволяє фізичному движку (Box2d) синхронно слідувати за графічним (cocos2dx);
- спосіб простий в реалізації.

Недоліками використання змінного кроку є внутрішня проблема обчислень фізичного движка Box2d, різкі рухи в зв'язці з Box2d.

Різкі рухи - дуже серйозний недолік, що виключає можливість використання даного способу.

Фіксований тимчасовий крок.

Використовувати фіксований часовий крок радить офіційна документація ігрового фізичного движка Box2d [24]. Фіксований тимчасовий крок в грі дорівнює 1/60, що відповідає 60 кадрам в секунду. Таким чином, псевдокод поновлення ігрового циклу представлений в лістингу 3.2.

Лістинг 3.2 - Псевдокод методу поновлення ігрового циклу з фіксованим кроком без згладжування

```
// fixed_time_step фіксований часовий крок
fixed_time_step = 1 / 60.f;
// time_to_consume час, який пройшов з попереднього оновлення
// оновити фізичну частину
update_physics(time_to_consume);
// оновити графічну частину
update_graphics_with_physics(fixed_time_step, physics);
```

При такій реалізації (лістинг 3.2) виникає інша проблема - фізичний і графічний движки оновлюються несинхронно. При низькій частоті зміни кадрів фізичний світ буде оновлено раніше на велику кількість фіксованих тимчасових кроків, при високій - фізичний світ буде відставати.

Для того, щоб була відсутня велика різниця між часом фізичного і графічного світів, прийнято використовувати змінну для збереження часу, яку не

використано для поновлення фізичного світу. У класі `PhysicsSystem` це властивість `m_FixedTimestepAccumulator`. При накопиченні достатньої кількості часу для одного або декількох кроків, фізичний світ оновлюється `nSteps` раз.

Лістинг 3.3 - Обчислення кількості кроків поновлення фізичного світу

```
int nSteps = static_cast<int> (  
    std::floor(std::abs(m_FixedTimestepAccumulator) / MC::FPS)  
);  
m_FixedTimestepAccumulator -= nSteps * MC::FPS;
```

Так вдається зберігати синхронність движків в межах одного фіксованого тимчасового кроку. Кількість максимально можливих кроків обмежена, що б уникнути «спіралі смерті» - загальноприйняте поняття для позначення ситуації, при якій кількість кроків, яке необхідно зробити фізичного світу настільки велике, що їх виконання впливає на продуктивність, що тягне за собою ще більший час затримок [25].

Для використання залишку від часу в `m_FixedTimestepAccumulator`, вводиться властивість `m_FixedTimestepAccumulatorRatio`, яке показує яку частину часу по відношенню до фіксованого кроку не використано.

Таким чином, алгоритм складається з 4 пунктів:

- використовуючи час, переданий для оновлення кадру, і `m_FixedTimestepAccumulator` (минуле необроблене, накопичений час), вважаємо ціле число кроків довжини фіксованого тимчасового кроку (`MC::FPS`), яке повинно бути змодельоване. Залишок зберігається в накопичувачі (`m_FixedTimestepAccumulator`);
- обчислюємо частину від необробленого часу щодо довжини фіксованого тимчасового кроку і зберігаємо його в `m_FixedTimestepAccumulatorRatio`;
- моделюємо необхідне число кроків фізичного світу;
- згладжуємо позицію і кут фізичних тіл.

Згладжування (`smoothing` [26]) використовується, щоб усунути наявність візуальних дефектів руху. Його можна реалізувати двома способами: інтерполяцією і екстраполяцією. Обидва способи мають переваги і недоліки.

Інтерполяція [27] обчислюється по двом відомим станам фізичних тіл (рис. 3.8 рис. 3.8): до симуляції кроку (точка А, рис. 3.8) і після (точка С,). Результат згладжування представлений на точці В рисунка 3.8.

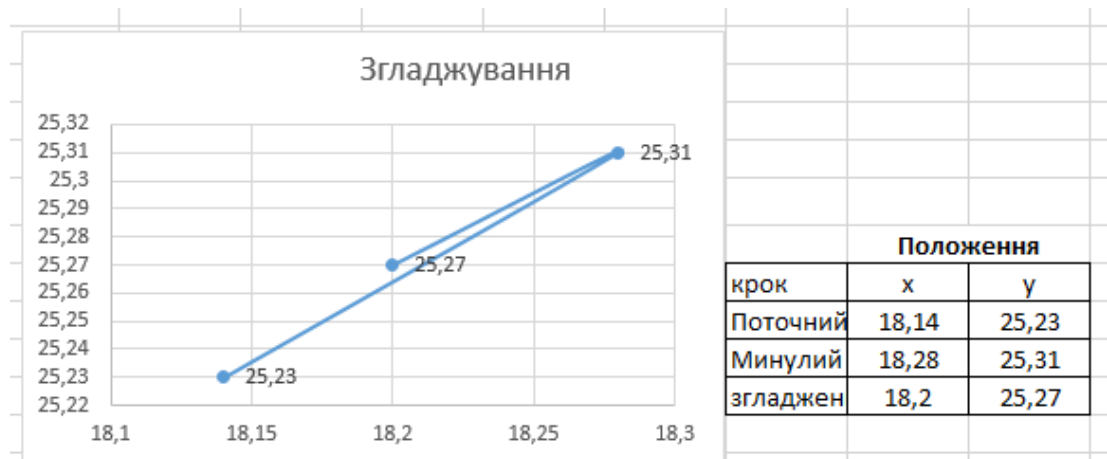


Рис. 3.8. Приклад згладжування

У даній роботі аналізувалося два типи інтерполяції [28]: лінійна, де α - $m_FixedTimestepAccumulatorRatio$ (лістинг 3.4), і методом косинуса (лістинг 3.5). Для досягнення максимальної плавності так само обчислюється «smoothstep» [28] або «плавний крок» - величина між 0 і 1, яка використовується при інтерполяції.

Лістинг 3.4 - Псевдокод лінійної інтерполяції

```
state = currentState * alpha + previousState * (1-alpha);
```

Лістинг 3.5 - Інтерполяція методом косинуса

```
float smoothstep(const float& x) {
```

```
    return (3 - 2 * x) * x * x;
```

```
}
```

```
const float mu2 = (1 -
```

```
cosf(smoothstep(m_FixedTimestepAccumulatorRatio)*MC::PI)) / 2.f;
```

```
m_SmoothPosition = (1 - mu2)*m_PreviousPosition + mu2 * b->GetPosition();
```

Перевага використання інтерполяції полягає в досягненні плавності і зменшенні кількості візуальних дефектів. Але при цьому виникає невелика затримка між станом фізичного світу і графічного. Це видно на малюнку 3.8, де згладжені координати для відображення графіки на екран, не збігаються з

кінцевим положенням фізичного тіла. Так само, кількість візуальних дефектів: скачки, ривки, збільшує той факт, що тіло рухається не по прямій лінії.

Другий спосіб виправлення візуальних дефектів - екстраполяція [29]. Для досягнення згладжування і обробки, що залишився в `m_FixedTimestepAccumulator` екстраполяція ґрунтується на «передбаченні» положення тіла, знаючи останній відомий стан тіла (лістинг 3.6).

Лістинг 3.6 - Псевдокод екстраполяції

```
state = currentState + currentStateDerivative * alpha
```

Перевагою використання екстраполяції є простота реалізації і повна синхронна робота двох двигунів. Недоліком такого підходу є можливе ігнорування зіткнень (подолання перешкод без оповіщення про зіткнення), що виключає можливість використання цього методу для даної гри. Так само, використання методу екстраполяції ще більш неточно, ніж інтерполяція для об'єктів, що рухаються не по прямій лінії.

Результат застосування методів на практиці показав, що найкращий спосіб уникнути візуальних дефектів використовуючи фіксований часовий крок і зберегти FPS 60 кадрів в секунду - інтерполяція. Перевагою даного методу є те, що увесь переданий для поновлення час використано і фізичний движок (`Box2d`) синхронно оновлюється з графічним (`cocos2dx`).

3.2.3. Фільтрація тимчасового кроку

Хорошим варіантом для вирішення даної проблеми є фільтрація вхідного для поновлення часу. Даний спосіб реалізований в движку `BitSquid`. Алгоритм складається з 4 пунктів:

- зберігати в історію кожне вхідне для поновлення час для 11 останніх кадрів;
- виключити з обчислень два з найбільшим і два з найменшим часом;
- обчислити середнє значення для решти семи значень;
- інтерполювати від знайденого середнього значення до останнього кадру для додавання більшої згладжуваності.

Виходячи з алгоритму, можна сказати, що це фільтр нижніх частот - фільтр, що не пропускає частоти нижче і вище певних значень. Фільтр адаптований для роботи з часом оновлення кадру. Реалізація алгоритму представлена класом TimeFilter, патерн Сінглтон, в додатку А.

На практиці, для тіла, яке може змінювати напрямок і швидкість кожен кадр, найкраще себе показав алгоритм фільтрації часу. Згладження він досягає за рахунок обчислення середньої величини часу поновлення останніх 11 кадрів і додаткової інтерполяції між двома кадрами. Також важливою перевагою цього способу є висока ефективність при роботі з мережею. Так як в довгостроковій перспективі гра припускає наявність багато режимів, рішення на користь цього варіанту.

3.3. Адаптація під різні роздільні здатності мобільних пристроїв

Мобільна гра «Скажені кулі» орієнтована на велику кількість різноманітних пристроїв з різними роздільними здатностями та глибиною екрану. Без підтримки різних роздільних здатностей гра буде виглядати на різних пристроях по-різному, наприклад, як представлено на рисунку 3.9. Для підтримки декількох роздільних здатностей екранів необхідно визначити: по-перше, з яким співвідношенням сторін екрану працювати, по-друге, визначити роздільну здатність розробки - DesignResolutionSize. По-третє, створити набори ресурсів для всіх підтримуваних роздільних здатностей. Ця практика детально описується в джерелах [7; 30; 31; 32].



Рис. 3.9. Відображення без рішення

Було вирішено працювати з співвідношенням сторін 1.5 і роздільною здатністю розробки 480x320. Для всіх можливих роздільних здатностей створювати окремі набори ресурсів немає необхідності, досить створити три папки для роздільних здатностей які значно відрізняються один від одного. У програмі це папки `assets / small`, `assets / normal` і `assets / large`. Роздільну здатність, яка відповідає ресурсам кожної папки, наведено в таблиці 3.1.

Таблиця 3.1

Основні категорії роздільних здатностей

Роздільна здатність	Ширина	Висота	Коефіцієнт масштабування (Scale factor)
<code>assets/small</code>	480	320	1
<code>assets/normal</code>	1024	768	2,4
<code>assets/large</code>	2048	1536	4,8
Цільова роздільна здатність (design resolution)	480	320	1
Роздільна здатність тестованого телефону	768	480	2,4

На рисунку 3.10 представлена гра з підтримкою різних роздільних здатностей. За допомогою `DesignResolutionSize` отримуємо коефіцієнт масштабування - число, яке необхідно передати в метод `Director::setContentScaleFactor`, що б движок міг коректно масштабувати:

- значення місця розташування об'єктів на екрані;
- розмірів шрифтів;
- розмірів ресурсів зображень.

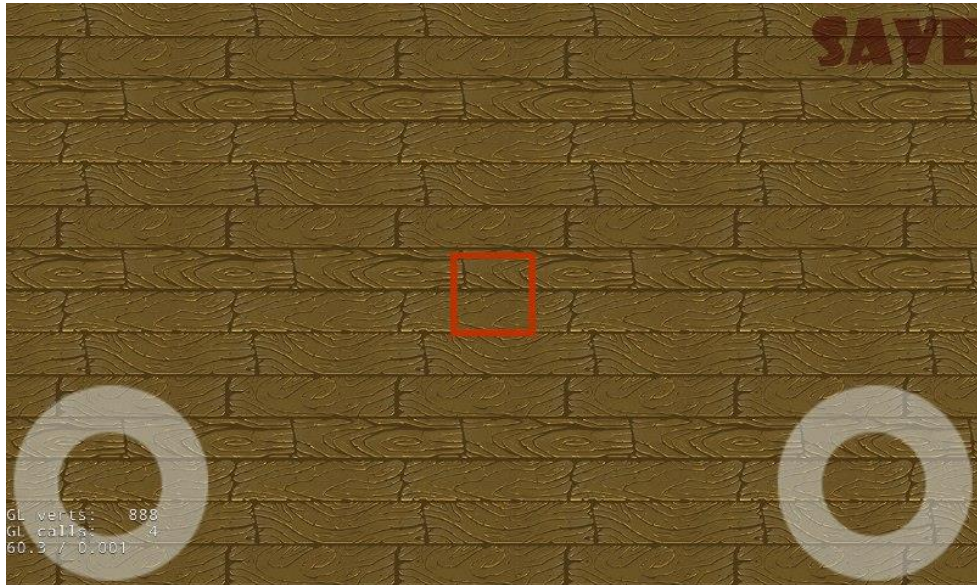


Рис. 3.10. Сцена з використанням рішенням

Фрагмент вихідного коду, який відповідає за реалізацію мульти роздільної здатності, представлений в додатку В.

3.4. Реалізація ігрового штучного інтелекту

Ігровий штучний інтелект (Game artificial intelligence, AI) - це набір програмних методик, що застосовуються в комп'ютерних іграх для створення враження інтелектуальної поведінки комп'ютерно-керованих персонажів [33; 34].

Для розробки ігрового AI були використані:

- діаграми діяльності (Activity Diagram) для опису алгоритму дій;
- діаграма прецедентів (User Case Diagram), що дозволяє описати можливості AI.

Можливі дії бота представлені на діаграмі 3.11.

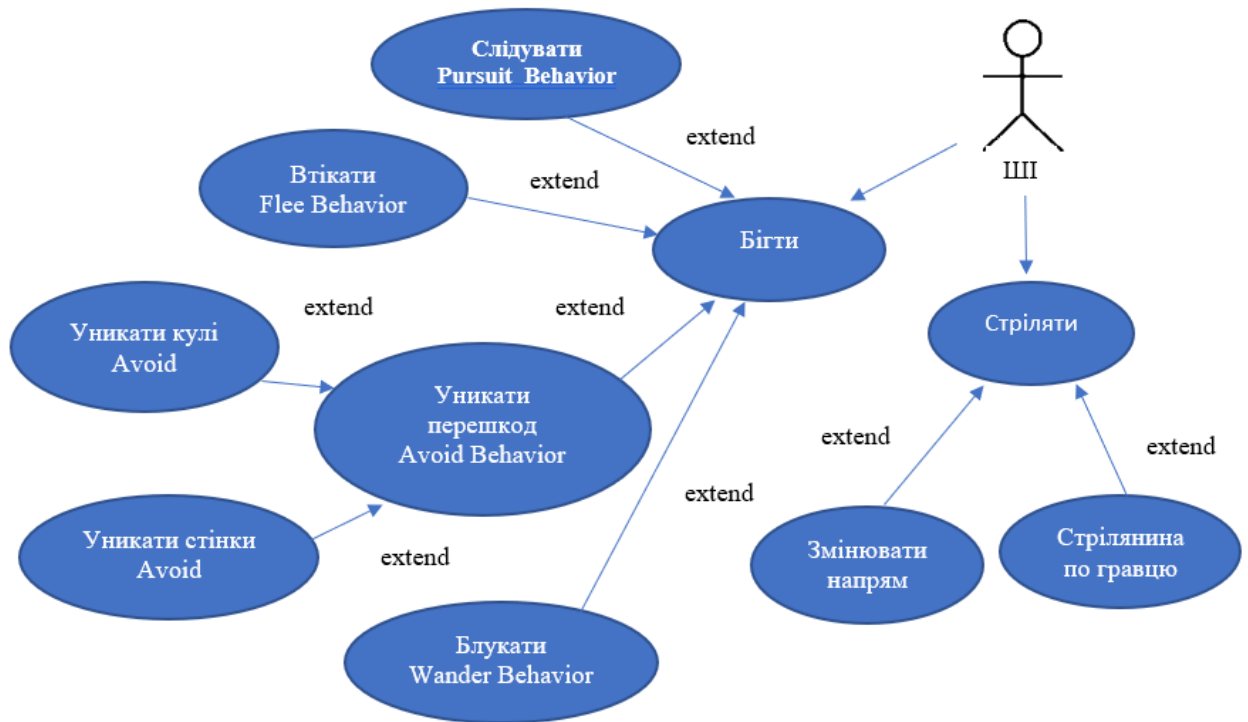


Рис. 3.11. Опис поведінки ігрового інтелекту

Таким чином, для реалізації прецеденту «Бігти» необхідно реалізувати розширяючі його прецеденти. Тобто, були реалізовані такі типи поведінки:

- втеча (flee behavior);
- гонитва (pursuit behavior);
- ухилення (avoid behavior);
- блукання (wander behavior);
- постріл.

Така поведінка називається Стірінг (steering [35]) і часто згадується Стівеном Рабіном [36]. Детальний опис реалізації прецедентів зроблено Фернандо Бевілаккуа [37], але з використанням вектора сили.

При реалізації архітектура ігрового AI була розділена на 3 частини (рис. 3.12):

- бізнес-логіка: незалежний від інших класів тип Logic;
- основний клас Enemy: відповідає за графіку і ініціалізацію Logic і EnemyBody класів;

- клас EnemyBody, інкапсулює тіло П (клас b2Body знаходиться у відносинах композиції з EnemyBody).

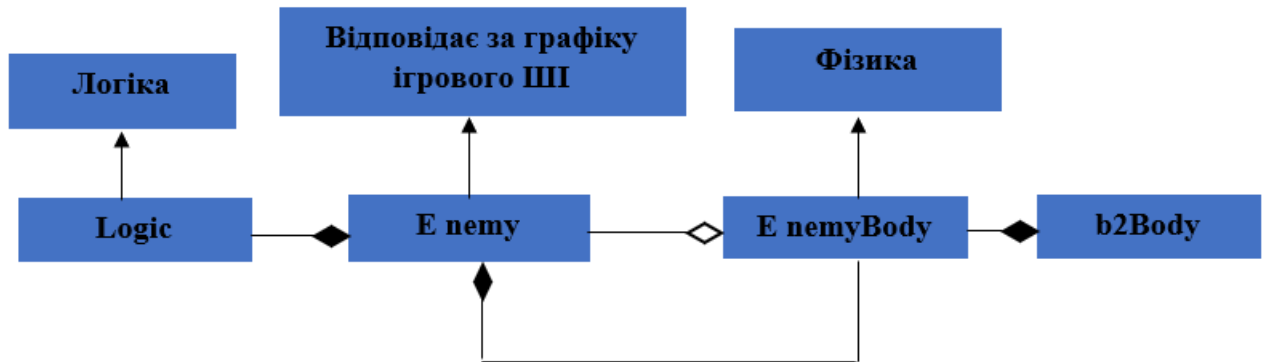


Рис. 3.12. Діаграма класів, що реалізують ігровий інтелект

Клас Logic наведено в файлі Logic.h в додатку Г. Завдання типу Logic - управління, зміна стану П. Logic інкапсулює дані, що відображають поточний стан бота:

- швидкість;
- позиція;
- позиція мети (для стану «Ухилення»);
- показчик на об'єкт класу Entity: сутність, яку необхідно уникнути;
- точка дотику, нормаль до точки дотику взаємодіючих об'єктів (для стану «Ухилення»). Ініціалізується при попаданні об'єкта (кулі, героя) в сенсор;
- тип цілі для стрільби;
- показчик на об'єкт класу Entity: сутність, в яку необхідно вистрілити.

Для реалізації станів використовується ідея патерну проектування стану (рис. 3.13).

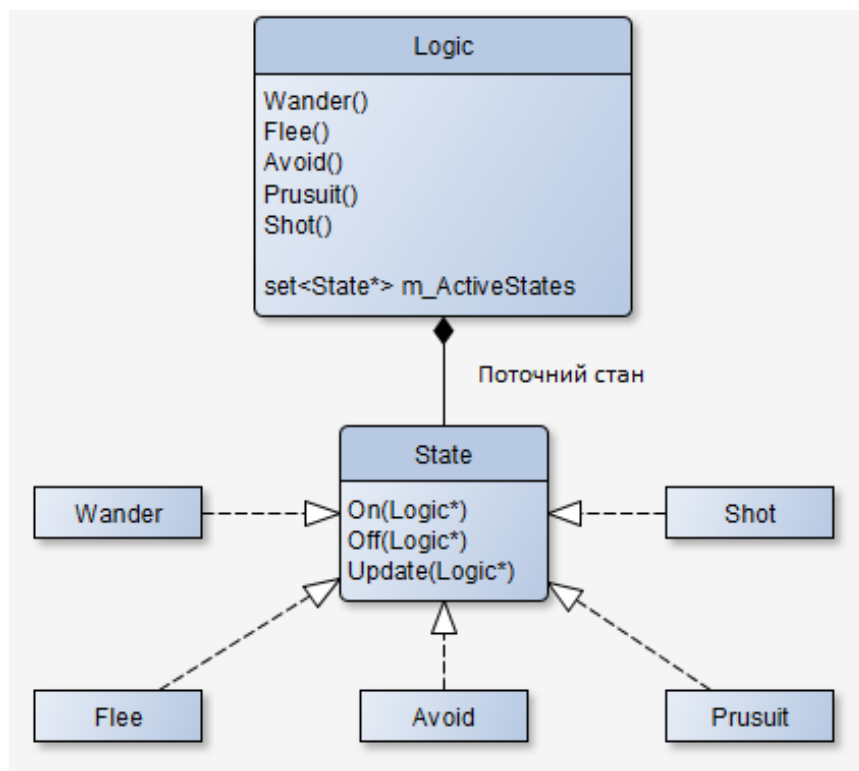


Рис. 3.13. UML-діаграма класів станів

Реалізація відрізняється від патерну стану тим, що може бути активно кілька станів одночасно (таб. 3.2). При переході в стан «стрілянина», все активовані стану відключаються. Стан «ухилення» змінює траєкторію руху інших станів.

Таблиця 3.2

Сумісність станів логіки ігрового ШІ

Стан	Сумісні стани
Стрільба (Shot)	Немає сумісних
Втеча (Flee)	Ухилення (Avoid)
Погоня (Pursuit)	Ухилення (Avoid)
Ухилення (Avoid)	Втеча (Flee), Погоня (Pursuit), Блукання (Wander)
Блукання (Wander)	Ухилення (Avoid)

Причина цієї модифікації пояснюється тим, що в основному дані стану працюють з поточної векторною величиною швидкості ШІ. Наприклад, заздалегідь «побачивши» перешкоду до поточної швидкості поступово буде додаватися вектор швидкості, що обходить перешкоду, що змінює напрямок руху

ШІ. На рисунку 3.14 відображений приклад роботи з векторами швидкостей в реалізації.

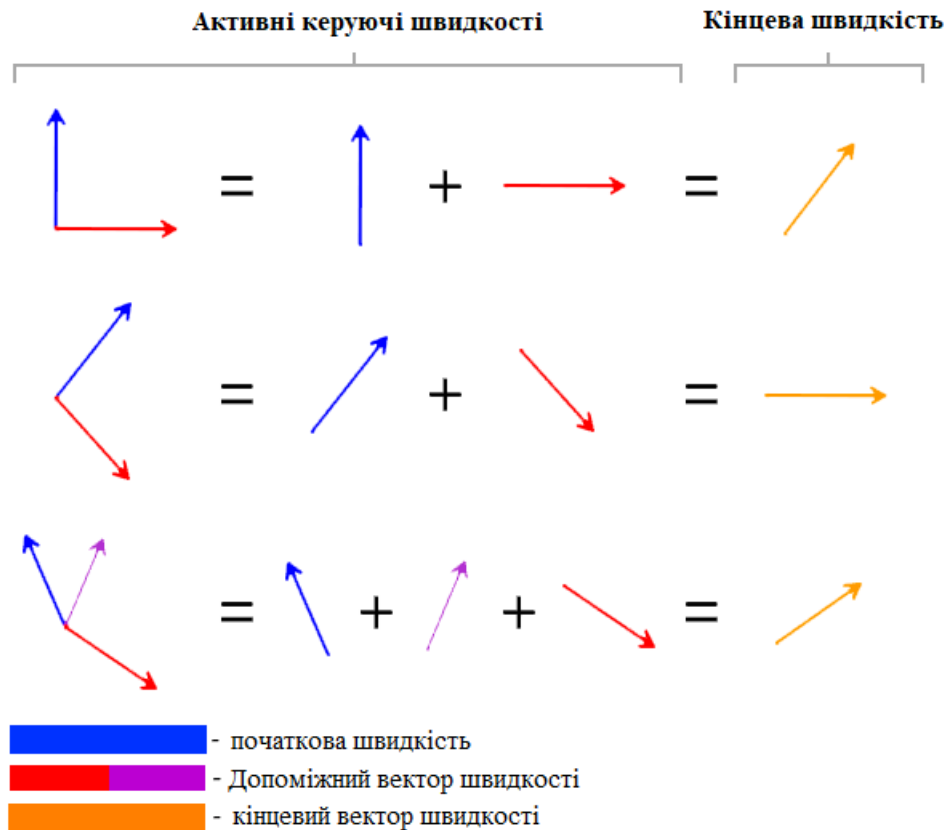


Рис. 3.14. Приклад роботи з вектором швидкості

Кожний стан по своєму модифікує вектор швидкості. Загальний алгоритм роботи ШІ представлений на рисунку 3.15. Інваріантом стану ШІ буде стан блукання. ШІ не бачить гравця, не знає де він і просто блукає в межах карти.

Для взаємодії зі світом бот наділений двома сенсорами:

- сенсор великого радіуса, що реагує на появу кулі;
- сенсор маленького радіуса, що реагує на появу гравця, стіни і кулі.

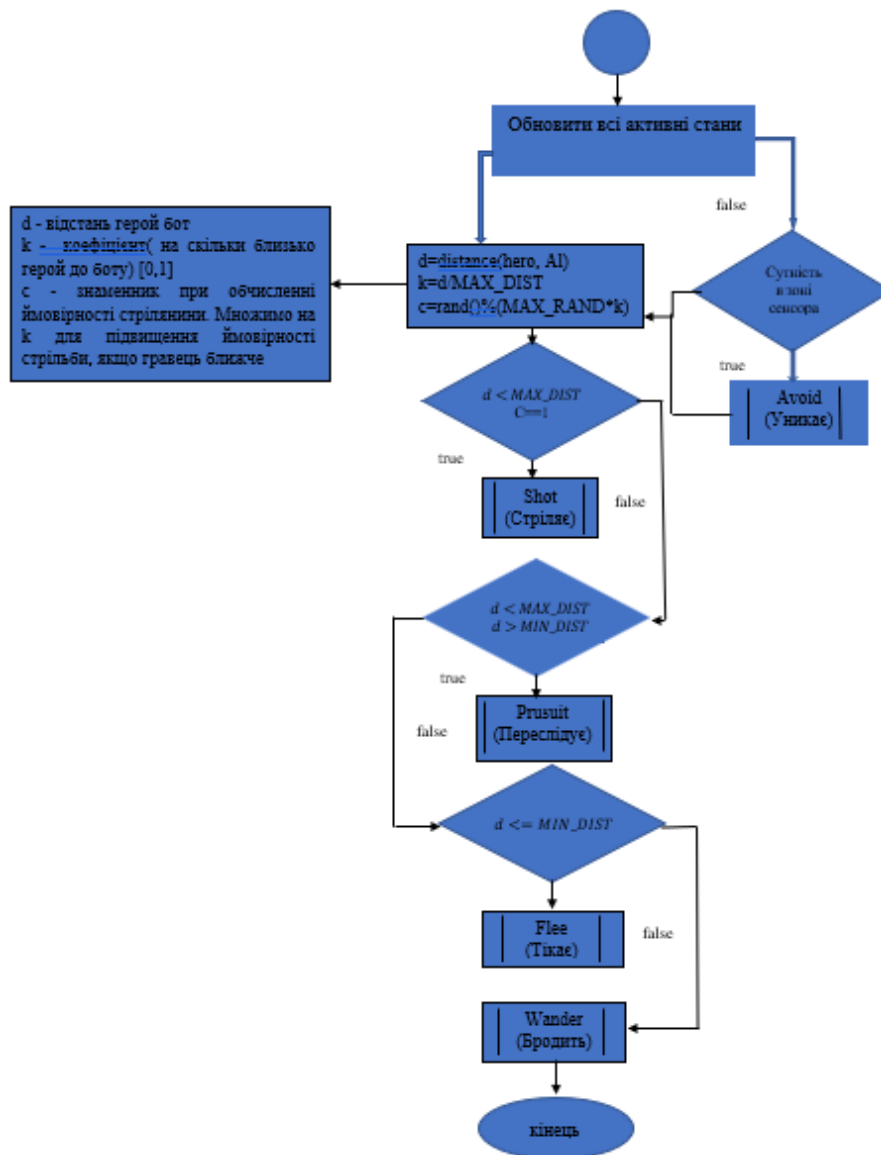


Рис. 3.15. UML-діаграма загального алгоритму поновлення ШІ

При взаємодії з сенсорами викликаються відповідні функції: ContactBegin, ContactEnd, PostSolve, PreSolve, які змінюють стану моделі (бізнес-логіки) викликом методів класу Logic. Методи представлені на рисунку 3.13.

За одне оновлення гри оновлюються всі включені стану і включаються або відключаються нові відповідно до алгоритму на рисунку 3.15.

Кожний стан перевизначає метод Update, в якому випадковим чином (стан Wander) або на основі отриманих даних (стан Prusuit) створюється вектор швидкості. Алгоритм роботи стану Wander представлений на рисунку 3.16. Створена швидкість не додається відразу до поточної швидкості ШІ. Для того, щоб рух і повороти здавалися плавними, вектор швидкості розбивається на кілька

частин, і до поточної швидкості додається частина згенерованої раніше швидкості - `m_VelocityToConsume`. Схожа техніка згладжування використовується в реалізаціях інших станів.

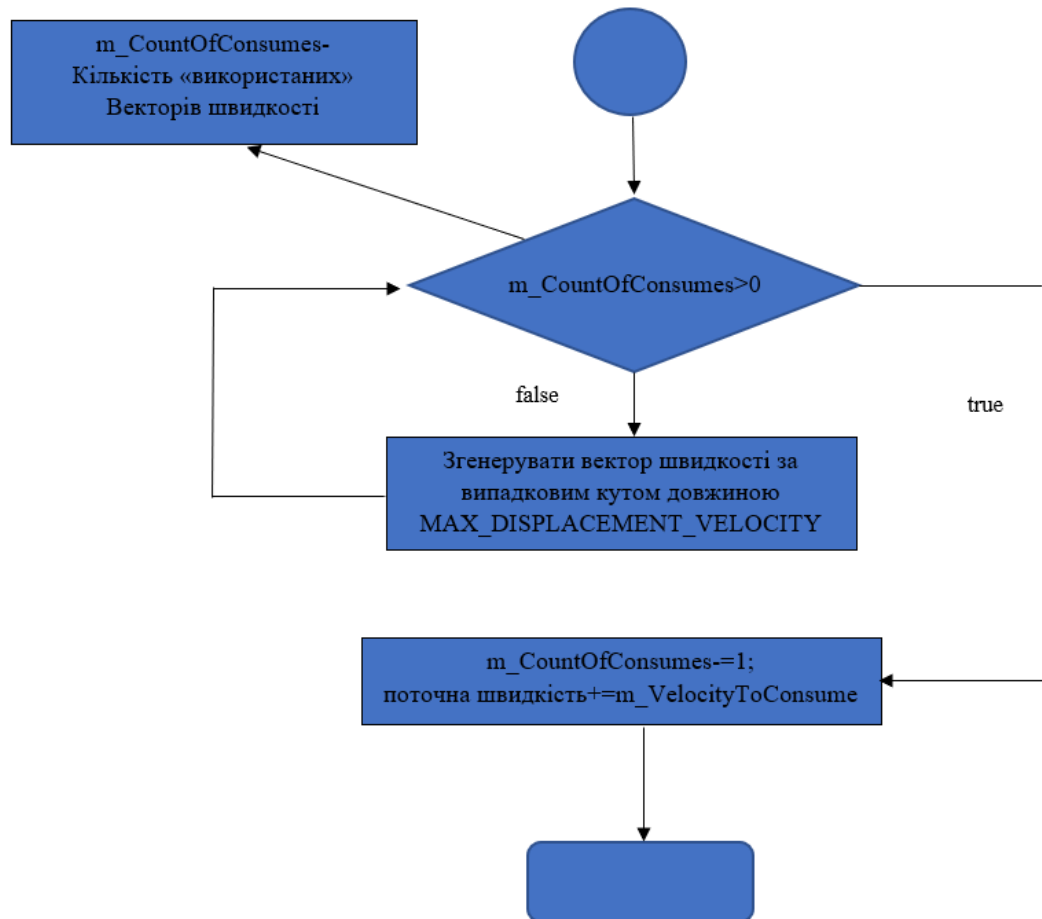


Рис. 3.16. Алгоритм роботи Wander: Update

Таким чином, використана реалізація допомагає автономним персонажам рухатися реалістичним чином в навколишньому середовищі. Ідея не заснована на складних обчисленнях, а використовують мінімум локальної інформації, що робить алгоритм простим і легко реалізованим. Проте, рухи ШІ можуть бути досить складними і непередбачуваними. Мінуси даної реалізації - непередбачувана поведінка при зіткненні з Т-подібними перешкодами.

3.5. Налаштування

Налаштування гри реалізується після компіляції з прапором «-debug» і запуск гри на девайсі через інтерфейс Android Debug Bridge (adb). Досить

тільки дозволити налагодження через USB на девайсі. Так само є можливість підключатися до девайсу через мережу WiFi. Відладчик Visual Studio дозволяє вести спостереження за поведінкою програми під час виконання і виявляти проблеми. Так само відладчик добре працює з cocos2dx. За допомогою відладчика можна переривати або припиняти виконання програми з метою перевірки коду, обчислювати і редагувати значення змінних програми, відстежувати стан реєстрів, переглядати інструкції, створені з вихідного коду, а також переглядати область пам'яті, яка використовується додатком. Отладчик Visual Studio підтримує покрокову налагодження та точки зупинки виконання з заданою умовою.

Для пошуку і виправлення помилок в «фізичному світі» гри необхідно успадковувати клас 2bDraw в Box2d і перевизначити його віртуальні функції. Це дасть можливість побачити чи збігаються розміри і положення фізичних тіл і їх графічний вигляд. Так само слід врахувати, що малювання повинно бути в межах верхнього шару.

Що б уникнути додаткових витрат часу був використаний сторонній модуль [38] для cocos2d-x 2.1.2, використовуючи XCode 4.4 під ліцензією MIT, і змінений для сумісності з cocos2d-x v3.14 і Visual Studio 2022. Графічне представлення фізичних тіл зображено на малюнку 3.17. Заштриховані елементарні фігури (прямокутник, трикутник) представляють фізичні тіла: кінематичні, статичні і динамічні.

У лівому нижньому кутку виводиться додаткова інформація сприяє виявленню багів і налагодженні:

- GL verts - кількість вершин на сцені;
- GL calls - кількість викликів графічних функцій переданих для обробки GPU;
- число перед слешем - кількість FPS в даний час;
- число після слеша - час промальовування одного кадру.

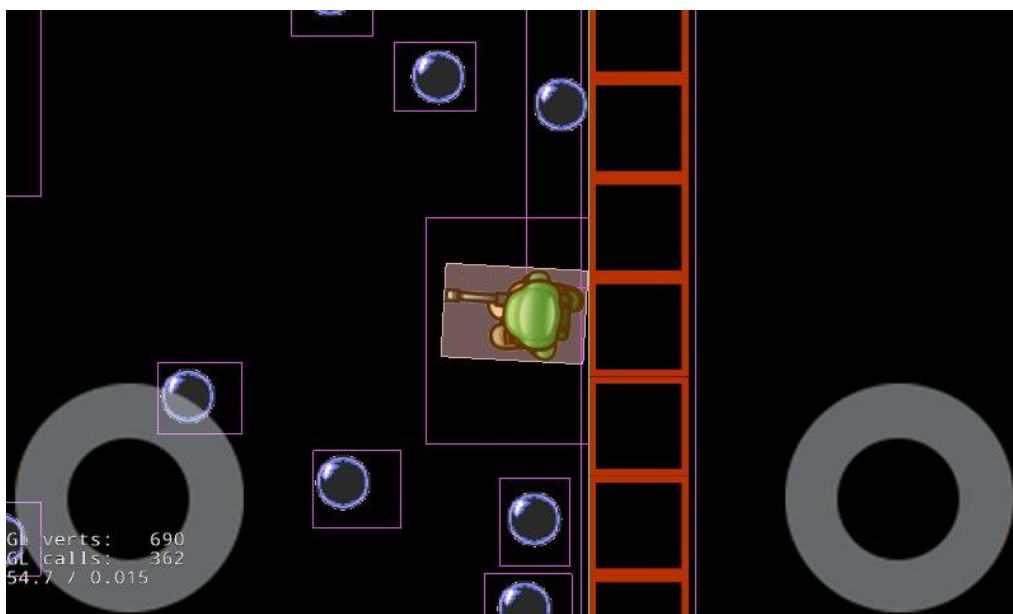


Рис. 3.17. Сцена під час налагодження гри

Додатковим засобом налагодження можна вважати Android Logcat Command-line Tool (рис. 3.18).

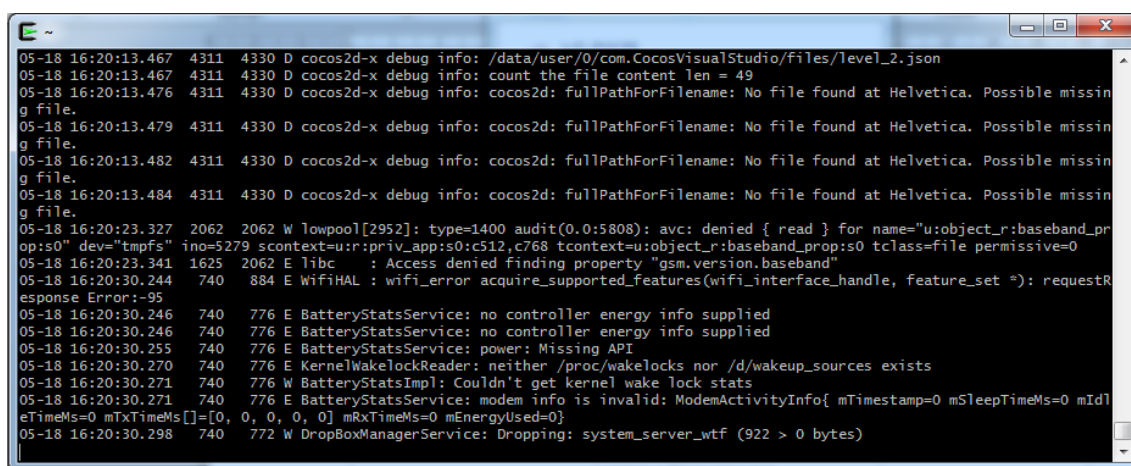


Рис. 3.18. Робота Logcat

Logcat - це інструмент командного рядка, який відображає журнал системних повідомлень, включаючи трасування стека, коли пристрій видає повідомлення про помилку, і повідомлення, які написані за допомогою макросу CCLOG.

3.6. Тестування

Під час розробки гри було написано кілька незалежних від графічної частини модулів. Одним з таких модулів є клас LevelHolder і пов'язані з ним призначені для користувача типи даних. Клас LevelHolder відповідає за

завантаження, обробку та редагування файлів в json форматі. Після його реалізації для виявлення і усунення прихованих проблем на ранньому етапі розробки необхідно провести тестування модуля. Кращим варіантом для тестування модуля було використання сторонніх бібліотек і фреймворків, наприклад, Google C ++ Testing Framework (Google Test). Google Test - бібліотека для модульного тестування на мові C ++. Перевагами цього фреймворка є наявність російської документації і ліцензія BSD, тобто це вільне програмне забезпечення.

Але для тестування модуля з десятком функцій додавати в проєкт додатковий фреймворк не має сенсу. Проблема була вирішена реалізацією додаткової одиниці трансляції MapLoaderTest.h і MapLoaderTest.cpp, де за допомогою макросу IS_EQUEL (x, y) порівнюються одержувані і очікувані результати роботи функцій. Тобто, метою створення одиниці трансляції MapLoaderTest.h було доказ коректності коду.

Також під час розробки гри з'являлася необхідність тестувати проміжні результати на девайсах з різним дозволом екрану, глибиною пікселів, версією операційної системи і API.

Гра була протестована на платформах:

- LG D325 Android 7.1.1;
- Lenovo s90 Android 7.2.0;
- Wileyfox Storm Android 7.2.0;
- Планшет Tab 702 Android 4.2.2;
- Планшет Samsung Galaxy Tab A SM-T355 Android 5.0;
- Lenovo A6010 Android 5.0.2;
- Lenovo A536 Android 4.4.2.

Гра відмінно працювала на всіх тестованих пристроях, показник середньої кількості кадрів в секунду - 60. Критичних помилок не виявлено.

3.7. Підпис гри

При компіляції програми для його тестування і запуску спочатку необхідно його підписати. У режимі налагодження додаток підписується debug ключем.

Термін дії debug - ключа необмежений. Але для публікації додатка Android у Google Play або для запуску release версії гри необхідно створити свій ключ і підписати ним додаток. Будь-який додаток, викладений в магазин, повинен мати підписаний сертифікат. Сертифікат дозволяє ідентифікувати вас як автора програми. Якщо хтось спробує викласти програму з такою ж назвою пакета додатка, йому буде відмовлено через конфлікт імен.

Підпис гри, при компіляції Release-збірки, здійснюється інструментами від Java Runtime Environment (скор. JRE; рос. Виконавча для Java) утилітою jarsigner. Перед тим як підписати додаток необхідно створити утиліту keytool. Обидві утиліти є безкоштовними і встановлюються разом JRE.

3.8. Висновки до розділу

У третьому розділі, присвяченому проєктуванню і конструюванню гри, були виділені додаткові підзадачі. При вирішенні таких підзадач, як «реалізація ігрового штучного інтелекту» і «розробка основного ігрового циклу» були порушені аспекти з інших областей знань: робота з векторами і згладжування.

ВИСНОВКИ

У магістерській роботі було розглянуто аналіз методів і технологій розробки комп'ютерної 2D гри для платформи ANDROID.

В роботі проведено аналіз статистичних даних, пов'язаних з розробкою мобільних ігор, а також виконано проєктування і розробка двомірної гри на мобільну платформу Android.

В ході аналізу було дано оцінку популярності жанрів і мобільних платформ на основі даних сучасних статистичних консалтингових компаній: Gartner, Shared2you, SurveyMonkey Intelligence. На основі обраного жанру (шутер - топ 3 скачуваних жанрів в 2023 році), була сформульована концепція розробляємої гри. Концепція закріплює платформу, жанр, назву, стиль графіки, дає короткий опис ідеї і геймплея гри і виділяє цільову аудиторію.

До стадії проєктування і програмування, програмне забезпечення було класифіковано за призначенням: робота з графікою (cocos2dx, Marmelade sdk, Unreal Engine 4), робота з кодом (Microsoft VS22, Eclipse, CLion) і робота з фізичним світом (Chipmunk, Box2d). З огляду на обмеження, відображені в концепції гри, основними вимогами були вартість, функціональність і зручність використання. Для мінімізації часу розробки були обрані додаткові інструменти для роботи з ресурсами (TexturePacker) і з вихідним кодом (GitHub). Таким чином, для розробки програми використовуються такі інструменти, як Microsoft VS 2022 року, Cocos2d 3.14, Box2d, Tiled, Texture Packer, GitHub.

Для ефективної розробки були виділені і вирішені наступні підзадачі:

- проєктування інтерфейсу;
- розробка основного ігрового циклу;
- адаптація гри під різні дозволи пристроїв;
- проєктування ігрового інтелекту;
- налагодження;
- тестування.

При вирішенні задачі про розробку основного ігрового циклу були порушені такі аспекти з інших областей знань, як:

- фільтр нижніх частот (low pass filter);
- згладжування за допомогою інтерполяції та екстраполяції.

Для реалізації циклу був використаний фільтр нижніх частот. Кожен розглянутий в роботі аспект являє собою незалежне самостійне рішення поставленої підзадачі, закріплений в заголовки `PhysicsSystem.h` і може бути використаний для обробки отриманого часу поновлення ігрового циклу в інших проєктах. Тема синхронізації фізичного і графічного движків популярна і перспективна в розвитку, так як її актуальність зростає зі збільшенням популярності ігор.

Особливу увагу було приділено адаптації гри під різні пристрої. Графічні ресурси були розділені на три категорії: `assets / small`, `assets / normal` і `assets / large` для зменшення втрати якості зображень при масштабуванні. В результаті, з мінімально можливим збільшенням необхідної пам'яті для гри, вдалося отримати якісне зображення на девайсах з різною роздільною здатністю екрану. Рішення, використане в даній роботі, активно застосовується й іншими розробниками.

Для створення ігрового штучного інтелекту, було використана і адаптована ідея управління рухом бота впливом на нього додатковими векторними величинами - швидкістю (в рамках цієї роботи). Використане рішення є коротким, простим, швидким (використовує мінімальну кількість даних) і реалістичним: уникає перешкоди, може переслідувати і тікати від цілі, стріляє в ціль та в кулі, що летять в нього. Завдяки використанню патерну проєктування стану і техніки поділу логіки і графіки реалізація ШІ легко підтримувана і розширюється.

При налагодженні програми був клонований і адаптований під `socos2d-x v3.14` і `Visual Studio 2022` незалежний модуль для роботи з фізичним движком `Vox2d - DebugDraw`. Він дозволяє промальовувати поточний стан фізичних тіл на екран, що значно прискорює налагодження. Модуль може бути використаний в інших проєктах.

Таким чином, результатом виконання завдання розробки двомірної гри на платформу Android стала гра «Скажені кулі» і два незалежних модуля: DebugDraw і PhysicsSystem. Робота гри була успішно протестована під систему Android 4.2.2 і вище. Створена гра узгоджується з описаною концепцією і володіє такими характерними рисами:

- відсутність необхідності в навчанні;
- швидкий і унікальний геймплей;
- просте управління;
- можливість створення рівня;
- простий і зрозумілий інтерфейс;
- непередбачуваність противника.

Подальший розвиток проєкту включає поліпшення функції додавання рівнів, додавання яскравого графічного дизайну, додавання музичного супроводу і введення статистики успіху гравця.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Галёнкин Сергей. Маркетинг игр / Сергей Галёнкин. – 2014. – 78 с.
2. Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016 [Электронный ресурс] / 2016. – Режим доступа: <https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems>.
3. New mobile game statistics every game publisher should know in 2016 [Электронный ресурс] / 2016. – Режим доступа: <https://www.surveymonkey.com/business/intelligence/mobile-game-statistics>.
4. Scott Robertson. How to design: Concept Design Process, Styling, Inspiration, and Methodology. – Titan Books, 2014. – 176 с.
5. Andrew M. Understanding Open Source and Free Software Licensing / Andrew M. St. Laurent. – O'Reilly Media, 2004. – 207 с.
6. About CMake [Электронный ресурс]. – Режим доступа: <https://cmake.org/overview>.
7. Engelbert Roger. Cocos2d-x by Example Beginner's Guide / Roger Engelbert. – Packt Publishing Ltd, 2013. – 246 с.
8. Shah Ryan. Blueprints Master the Art of Unreal Engine 4 – Blueprints / Ryan Shah. – CreateSpace Independent Publishing Platform, 2014. – 122 с.
9. ISO/IEC 14882:2014. Information technology – Programming languages – C++ / ISO, 2014. – Режим доступа: <https://www.iso.org/standard/64029.html>.
10. Интегрированная среда разработки Visual Studio [Электронный ресурс] / Microsoft, 2017. – Режим доступа: <https://www.visualstudio.com/ru/vs/>.
11. Bourg David M. Physics for Game Developers, 2nd Edition / David M Bourg, Bryan Bywalec. – O'Reilly Media, 2013. – 578 с.
12. Korth Andy. Chipmunk 7 released – Pro tools open sourced [Электронный ресурс] / Andy Korth. – Howling Moon Software, 2015. – Режим доступа: <http://howlingmoonsoftware.com/wordpress/chipmunk-7-released-pro-tools-open-sourced>.

13. Feronato Emanuele. Box2D for Flash Games / Emanuele Feronato. – Packt Publishing, 2012. – 166 с.
14. Parberry I. Introduction to Game Physics with Box2D / Ian Parberry. – CRC Press, 2013. – 275 с.
15. Scott Chacon. Pro Git [Электронный ресурс] / Scott Chacon, Ben Straub. – 2014. – Режим доступа: <https://git-scm.com/book/en/v2>.
16. Чакон Скотт. Git для профессионального программиста / Скотт Чакон, Бен Штрауб. – Питер, 2016. – 496 с.
17. Фримен Элизабет. Паттерны проектирования / [Элизабет Фримен, Эрик Фримен, Кэти Сиерра, Берт Бейтс]. – Питер, 2013. – 656 с.
18. Смит Джейсон Мак-Колм. Элементарные шаблоны проектирования / Джейсон Мак-Колм Смит. – М.: «Вильямс», 2012. – 304 с.
19. Гамма Эрих. Приемы объектно-ориентированного проектирования. Паттерны проектирования / [Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес]. – Питер, 2016. – 366 с.
20. Макконнелл Стив. Совершенный код. Мастер-класс / Стив Макконнелл. – Русская Редакция, 2017. – 896 с.
21. Scott Meyers. Effective Modern C++ / Scott Meyers. – 2014. – 334 с.
22. OMG. *OMG Unified Modeling Language (OMG UML), Superstructure Version 2.5* / OMG, 2015. – Режим доступа: <http://www.omg.org/spec/UML/2.5/PDF>.
23. Despain. 100 Principles of Game Design / Despain, Wendy. – New Riders, 2012. – 240 с.
24. Erin Catto. World Class. Simulation [Электронный ресурс] / Erin Catto // Box2D v2.3.0 User Manual. – 2013. – Режим доступа: <http://box2d.org/manual.pdf>.
25. Fiedler Glenn. Fix Your Timestep! [Электронный ресурс] / Glenn Fiedler. – 2016. – Режим доступа: <http://gafferongames.com/game-physics/fix-your-timestep>.
26. O'Haver T. Smoothing [Электронный ресурс] / O'Haver T. – 2012. – Режим доступа: <http://terpconnect.umd.edu/~toh/spectrum/Smoothing.html>.

27. Corless Robert. A Graduate Introduction to Numerical Methods – From the Viewpoint of Backward Error Analysis / Robert Corless, Nicolas Fillion. – Springer Science & Business Media, 2013. – 869 с.
28. Abel Gomes. Geometric Computing. Lecture 4 – Interpolation methods [Электронный ресурс] / Abel Gomes. – 2010. – Режим доступа: <http://www.di.ubi.pt/~agomes/tcg/lectures/04-lecture.pdf>.
29. Matsuura Akihiro. Cocos2d-x Cookbook / Akihiro Matsuura. – UK: Packt Publishing, 2015. – 250 с.
30. Siddharth Shekar. Cocos2d Cross-Platform Game Development Cookbook – Second Edition / Shekar Siddharth. – UK: Packt Publishing, 2016. – 384 с.
31. Hernandez Raydelto. Building Android Games with Cocos2d-x / Raydelto Hernandez. – UK: Packt Publishing, 2015. – 147 с.
32. Kehoe Donald. Designing Artificial Intelligence for Games [Электронный ресурс] / Donald Kehoe. – 2015. – Режим доступа: <https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1>.
33. Heaton Jeff. Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms / Jeff Heaton. – Heaton Research, 2013. – 147 с.
34. Aung Sithu Kyaw. Unity 4.x Game AI Programming / Aung Sithu Kyaw, Clifford Peters, Thet Naing Swe. – Packt Publishing, 2013. – 232 с.
35. Rabin Steven. Game AI Pro: Collected Wisdom of Game AI Professionals / Steven Rabin. – A K Peters/CRC Press, 2013. – 626 с.
36. Bevilacqua Fernando. Understanding Steering Behaviors [Электронный ресурс] / Fernando Bevilacqua. – 2013. – Режим доступа: <https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-movement-manager--gamedev-4278>.
37. Box2d debug draw for Cocos2d-x [Электронный ресурс]. – Режим доступа: <https://github.com/vinova/B2DebugDraw>.

ДОДАТОК А

Вихідний код класу фільтра часу

```
class TimeFilter {  
    /*  
1.    Keep a history of the time step for the last 11 frames.  
2.    Throw away the outliers, the two highest and the two lowest values.  
3.    Calculate the mean of the remaining 7 values.  
4.    Lerp from the time step for the last frame to the calculated mean (adding more  
smoothness)  
    */  
  
    static const int    m_Size = 11;//history size  
    int                m_Count;  
    float              m_LowFilter;  
    float              m_History[m_Size];  
  
public:  
    static TimeFilter& GetInstance() {  
        static TimeFilter timefilter(0.2f);  
        return timefilter;  
    }  
  
    float GetTimestep(float dt) {  
        //first 11 calls: find the mean  
        assert(m_Count <= 11);  
        float mean = 0.f;  
        if (m_Count == m_Size) {  
            //Throw away the outliers, the two highest and the two lowest values.  
            float temp[m_Size];  
            std::copy(m_History, m_History + m_Size, temp);  
            std::sort(temp, temp + m_Size);
```

```

        mean = std::accumulate(temp + 2, temp + m_Size - 2, 0.f)/(m_Size -
4.f);

        //change history
        for (int i = 0; i < m_Size - 1; i++) {
            m_History[i] = m_History[i + 1];
        }
        m_History[m_Size - 1] = dt;
    }
    else {
        m_History[m_Count++] = dt;
        mean = std::accumulate(m_History, m_History + m_Count, 0.f) /
m_Count;
    }
    //smoothing
    float timestep = m_LowFilter * dt + (1.f - m_LowFilter) * mean;
    // CCLOG("dt - filtered: %.4f %.4f", dt, timestep);
    return timestep;
}

private:
    TimeFilter(float lowfilter) :
        m_Count(0),
        m_LowFilter(lowfilter) {}
    TimeFilter(const TimeFilter& ) = delete;
    TimeFilter& operator=(const TimeFilter& ) = delete;

```

ДОДАТОК Б

Вихідний код вирішення проблеми змінення роздільної здатності екрана

```
#include "AppDelegate.h"
#include "MainMenuScene.h"
#include "Tests/MapLoaderTest.h"
#include "Constants.h"

using namespace cocos2d;

static Size designResolutionSize = cocos2d::Size(480, 320);
static Size smallResolutionSize = cocos2d::Size(480, 320);
static Size mediumResolutionSize = cocos2d::Size(1024, 768);
static Size largeResolutionSize = cocos2d::Size(2048, 1536);

AppDelegate::AppDelegate() {}

AppDelegate::~AppDelegate() {}

// if you want a different context, modify the value of glContextAttrs
// it will affect all platforms

void AppDelegate::initGLContextAttrs() {
    // set OpenGL context attributes: red,green,blue,alpha,depth,stencil
    GLContextAttrs glContextAttrs = {8, 8, 8, 8, 24, 8};
    GLView::setGLContextAttrs(glContextAttrs);
}

// if you want to use the package manager to install more packages,
// don't modify or remove this function
static int register_all_packages() {
    return 0; //flag for packages manager
}

bool AppDelegate::applicationDidFinishLaunching() {
    // initialize director

    auto director = Director::getInstance();
    auto glview = director->getOpenGLView();
    if (!glview) {
```

```

        #if (CC_TARGET_PLATFORM == CC_PLATFORM_WIN32) ||
(CC_TARGET_PLATFORM == CC_PLATFORM_MAC) ||
(CC_TARGET_PLATFORM == CC_PLATFORM_LINUX)

            glview = GLViewImpl::createWithRect("Shooter", cocos2d::Rect(0,
0, designResolutionSize.width, designResolutionSize.height));

        #else

            glview = GLViewImpl::create("Shooter");

        #endif

        director->setOpenGLView(glview);

    }

    // turn on display FPS
    director->setDisplayStats(true);

    // set FPS. the default value is 1.0/60 if you don't call this
    director->setAnimationInterval(MC::FPS);

    director->setDepthTest(false);

    director->setProjection(Director::Projection::_2D);

    // Set the design resolution
    glview->setDesignResolutionSize(designResolutionSize.width,
designResolutionSize.height, ResolutionPolicy::NO_BORDER);

    Size frameSize = glview->getFrameSize();

    float scaleFactor = 1.f;

    if (frameSize.height > mediumResolutionSize.height) {
        FileUtils::getInstance()->addSearchPath("large");//2048, 1536
        scaleFactor = largeResolutionSize.height /
designResolutionSize.height;
    }

    else if (frameSize.height > smallResolutionSize.height) {
        FileUtils::getInstance()->addSearchPath("normal");//1024, 768

```

```

        scaleFactor = mediumResolutionSize.height /
designResolutionSize.height;
    }
    else {
        FileUtils::getInstance()->addSearchPath("small");//480, 320
        scaleFactor = smallResolutionSize.height /
designResolutionSize.height;
    }
    director->setContentScaleFactor(scaleFactor);
    register_all_packages();
    // create a scene. it's an autorelease object
    director->runWithScene(MainMenuScene::createScene());

    //tests
    //TEST::RUN_ALL();

    return true;
}

// This function will be called when the app is inactive. Note, when receiving a
phone call it is invoked.

void AppDelegate::applicationDidEnterBackground() {
    Director::getInstance()->stopAnimation();
    // if you use SimpleAudioEngine, it must be paused
    // SimpleAudioEngine::getInstance()->pauseBackgroundMusic();
}

// this function will be called when the app is active again
void AppDelegate::applicationWillEnterForeground() {
    Director::getInstance()->startAnimation();
    // if you use SimpleAudioEngine, it must resume here
    // SimpleAudioEngine::getInstance()->resumeBackgroundMusic();
}

```

ДОДАТОК В

Вихідний код класу PhysicsSystem

```
class PhysicsSystem {
    float m_FixedTimestepAccumulator;
    float m_FixedTimestepAccumulatorRatio;
    int m_VelocityIterations;
    int m_PositionIterations;
    b2World* m_World;
public:
    PhysicsSystem(
        const b2Vec2 gravity,
        b2ContactListener* listener,
        bool doSleep,
        int velocityIters,
        int positionIters
    ):
        m_FixedTimestepAccumulator(0),
        m_FixedTimestepAccumulatorRatio(0),
        m_VelocityIterations(velocityIters),
        m_PositionIterations(positionIters)
    {
        m_World = new b2World (gravity);
        m_World->SetAllowSleeping(doSleep);
        m_World->SetAutoClearForces(false);
        m_World->SetContactListener(listener);
        m_World->SetContinuousPhysics(true);
    }
    ~PhysicsSystem() {
        if(m_World)
            delete m_World;
        m_World = nullptr;
    }
    b2World* GetWorld()const {
        return m_World;
    }

    void fixedUpdate(float dt) {
        m_World->Step(dt, m_VelocityIterations, m_PositionIterations);
        m_World->ClearForces();
        resetSmoothStates_();
    }
    /*
```

The algorithm is simple:

1. using the requested time to consume (input parameter dt) and the previous, unprocessed accumulated time (fixedTimestepAccumulator_), computes the natural number of steps, each of length FIXED_TIMESTEP, that must be simulated; the remainder is stored back in the accumulator (fixedTimestepAccumulator_).
2. computes the fraction of unprocessed time, respect to FIXED_TIMESTEP, and stores it into fixedTimestepAccumulatorRatio_.
3. simulates the required number of physics steps.
4. calls b2World::ClearForces() method to reset applied forces.
5. smooths positions and orientations of physics objects.

```

*/
void update(float dt) {
    // Maximum number of steps, to avoid degrading to an halt.
    const int MAX_STEPS = 5;
    m_FixedTimestepAccumulator += dt;

    int nSteps = static_cast<int> (
        std::floor(std::abs(m_FixedTimestepAccumulator) / MC::FPS)
    );

    // To avoid rounding errors, touches fixedTimestepAccumulator_ only if
needed.
    if (nSteps > 0) {
        m_FixedTimestepAccumulator -= nSteps * MC::FPS;
    }
    else {
//        CCLOG("Engine can't make any step");
    }
    m_FixedTimestepAccumulatorRatio = m_FixedTimestepAccumulator /
MC::FPS;

    const int nStepsClamped = std::min(nSteps, MAX_STEPS);
    for (int i = 0; i < nStepsClamped; ++i) {
        if(i == nStepsClamped - 1) resetSmoothStates_();
        m_World->Step(MC::FPS, m_VelocityIterations,
m_PositionIterations);
    }
    m_World->ClearForces();

//    CCLOG("Smoothing");
    smoothStates_();
}
private:

```

```

void smoothStates_() {

    const float ratio = m_FixedTimestepAccumulatorRatio;
    const float mu = smoothstep_(ratio);

    for (b2Body * b = m_World->GetBodyList(); b != nullptr; b = b-
>GetNext()) {
        if (b->GetType() == b2_staticBody) continue;
        this->interpolation_(b,mu);
        /*
        CCLOG("current: %.4f,%.4f; previous: %.4f,%.4f; smooth: %.4f,%.4f;",
            b->GetPosition().x, b->GetPosition().y,
            e->m_Component.m_PreviousPosition.x, e-
>m_Component.m_PreviousPosition.y,
            e->m_Component.m_SmoothPosition.x, e-
>m_Component.m_SmoothPosition.y
        );
        */
    }
}

void resetSmoothStates_() {
    for (b2Body * b = m_World->GetBodyList(); b != NULL; b = b-
>GetNext()) {
        if (b->GetType() == b2_staticBody) continue;
        Entity* e = static_cast<Entity*>(b->GetUserData());

        e->m_Component.m_SmoothPosition = e-
>m_Component.m_PreviousPosition = b->GetPosition();
        e->m_Component.m_SmoothAngle = e-
>m_Component.m_PreviousAngle = b->GetAngle();
    }
}

float smoothstep_(const float& x) {
    return (3 - 2 * x) * x * x;
}

void interpolation_(b2Body* b, const float& smoothstep) {
    /*e->m_Component.m_SmoothPosition =
    mu * b->GetPosition() +
    (1 - mu) * e->m_Component.m_PreviousPosition;

    e->m_Component.m_SmoothAngle =
    mu * b->GetAngle() +

```



```

    (1 - mu) * e->m_Component.m_PreviousAngle;
    */
    /*interpolate
    double mu2;
    mu2 = (1-cos(mu*PI))/2;
    return(y1*(1-mu2)+y2*mu2);
    */
    Entity* e = static_cast<Entity*>(b->GetUserData());
    const float mu2 = (1 - cosf(smoothstep*MC::PI)) / 2.f;

    e->m_Component.m_SmoothPosition =
        (1 - mu2) * e->m_Component.m_PreviousPosition +
        mu2 * b->GetPosition();

    e->m_Component.m_SmoothAngle =
        (1 - mu2) * b->GetAngle() +
        mu2 * e->m_Component.m_PreviousAngle;
}
void extrapolation_(b2Body*b) {
    Entity* e = static_cast<Entity*>(b->GetUserData());
    const float time = m_FixedTimestepAccumulatorRatio * MC::FPS;
    e->m_Component.m_SmoothPosition = b->GetPosition() + time * b-
>GetLinearVelocity();
    e->m_Component.m_SmoothAngle = b->GetAngle() + time * b-
>GetAngularVelocity();
}
};

```

ДОДАТОК Г

Вихідний код файлу Logic.h

```
#pragma once
#include <memory>
#include <set>
#include <cocos2d.h>
#include "../Constants.h"
#include "../Base.h"

using namespace cocos2d;

enum class StateId {
    wander      = 0x1,
    pursuit = 0x2,
    avoid = 0x4,
    flee   = 0x8,
    shot   = 0x10
};

enum class Target {
    player, //hero
    entity //ball
};

class Logic;

class State {
protected:
    StateId m_Id;
    bool m_IsOn;
public:
    virtual void On(Logic*) = 0;
    virtual void Off(Logic*) = 0;
```

```

virtual void Update(Logic*) = 0;
virtual StateId GetId()const {
    return m_Id;
};
virtual bool IsOn()const {
    return m_IsOn;
}
};

namespace HelperFunctions {
    extern int toMask(const StateId&s);
    extern int random(int a1, int a2);
    extern float toRad(float degs);
    extern float toDeg(float rads);
}

struct LogicImpl {
    Vec2 m_Velocity, //calcs like sum of prev and desired velocity
        m_Position, //current position
        m_Target;
    Entity* m_AvoidEntity;
    b2WorldManifold m_Manifold;

    Target m_ShotTarget;
    Entity* m_ShotEntity;
};

template<MC::BulletType _type> class Gun;

class Logic {
    bool m_ShotInQueue;
public:
    Logic(Vec2 pos);
    void Wander();

```

```

void Pursuit();
void Flee();
void Avoid(Entity* avoid, b2WorldManifold manifold);
void Shot();

void Update(float dt);
//=====

int      GetStateMask()const {
    int mask = 0;
    for(auto&s: m_ActiveStates)
        mask|= static_cast<int>(s->GetId());
    return mask;
}
//=====

Vec2 GetVelocity()const {
    return m_LogicImpl->m_Velocity;
}

void SetVelocity(Vec2 velocity) {
    m_LogicImpl->m_Velocity = velocity;
}
//===== in PTM =====

Vec2 GetPosition()const {
    return m_LogicImpl->m_Position;
}
//===== in PTM =====

void SetPosition(Vec2 pos) {
    m_LogicImpl->m_Position = pos;
}
//=====

Vec2 GetTargetPosition()const {

```

```

        return m_LogicImpl->m_Target;
    }
    void SetTargetPosition(Vec2 target) {
        m_LogicImpl->m_Target = target;
    }
    //=====
    Entity* GetAvoidEntity()const {
        return m_LogicImpl->m_AvoidEntity;
    }
    b2WorldManifold& GetManifold()const {
        return m_LogicImpl->m_Manifold;
    }
    //=====
    void SetShotEntity(Entity*e) {
        m_LogicImpl->m_ShotEntity = e;
    }
    Entity* GetShotEntity() const{
        return m_LogicImpl->m_ShotEntity;
    }
    //=====
    void QueuedShot() {
        m_ShotInQueue = true;
    }
    void CleanQueuedShot() {
        m_ShotInQueue = false;
    }
    bool HasQueuedShot()const {
        return m_ShotInQueue;
    }
    //=====

```

```

void SetShotTargetType(Target type) {
    m_LogicImpl->m_ShotTarget = type;
}
Target GetShotTargetType()const {
    return m_LogicImpl->m_ShotTarget;
}

```

private:

```

std::shared_ptr<LogicImpl> m_LogicImpl;
std::set<std::shared_ptr<State>> m_ActiveStates;
std::shared_ptr<Gun<MC::BulletType::Ball>> m_Gun;
//
void InitStates();
};

```

```

class Wander : public State {
    static const int MAX_VELOCITY = 6 * MC::PTM_RATIO;
    static const int MAX_DISPLACEMENT_VELOCITY = 3 *
MC::PTM_RATIO;
    Vec2 m_VelocityToConsume;
    int m_CountOfConsumes;
    int m-CompatibleMask;
public:
    Wander() {
        m_VelocityToConsume = Vec2::ZERO;
        m_CountOfConsumes = 0.f;
        m_Id = StateId::wander;
        m-CompatibleMask |= (int)StateId::avoid;
    }
    virtual void On(Logic* pLogic) override;

```

```

        virtual void Off(Logic* pLogic) override;
        virtual void Update(Logic* pLogic) override;
};

class Avoid : public State {
    Vec2 m_VelocityToConsume;
    int m_CountOfConsumes;
    int m-CompatibleMask;

    static const int MAX_AVOID_VELOCITY = 4 * MC::PTM_RATIO;
public:
    Avoid() {
        m_VelocityToConsume = Vec2::ZERO;
        m_CountOfConsumes = 0.f;
        m_Id = StateId::avoid;
        m-CompatibleMask |= (int)StateId::flee;
        m-CompatibleMask |= (int)StateId::pursuit;
        m-CompatibleMask |= (int)StateId::wander;
    }

    virtual void On(Logic* pLogic) override;
    virtual void Off(Logic* pLogic) override;
    virtual void Update(Logic* pLogic) override;
};

class Prusuit : public State {
    Vec2 m_VelocityToConsume;
    int m_CountOfConsumes;
    int m-CompatibleMask;

    static const int MAX_VELOCITY = 5 * MC::PTM_RATIO;
public:
    Prusuit() {

```

```

        m_VelocityToConsume = Vec2::ZERO;
        m_CountOfConsumes = 0.f;
        m_Id = StateId::pursuit;
        m-CompatibleMask |= (int)StateId::pursuit;
        m-CompatibleMask |= (int)StateId::avoid;
    }

    virtual void On(Logic* pLogic) override;
    virtual void Off(Logic* pLogic) override;
    virtual void Update(Logic* pLogic) override;
};

class Flee : public State {
    Vec2 m_VelocityToConsume;
    int   m_CountOfConsumes;
    int   m-CompatibleMask;
    static const int MAX_VELOCITY = 5 * MC::PTM_RATIO;

public:
    Flee() {
        m_VelocityToConsume = Vec2::ZERO;
        m_CountOfConsumes = 0.f;
        m_Id = StateId::flee;
        m-CompatibleMask |= (int)StateId::avoid;
    }

    virtual void On(Logic* pLogic) override;
    virtual void Off(Logic* pLogic) override;
    virtual void Update(Logic* pLogic) override;
};

class Shot : public State {
    Vec2 m_SavedVelocity;
    float m_Time;
    int   m-CompatibleMask;

```



```
public:
    Shot() {
        m_Id = StateId::shot;
        m_Time = 0.f;
    }
    virtual void On(Logic* pLogic) override;
    virtual void Off(Logic* pLogic) override;
    virtual void Update(Logic* pLogic) override;
};
```