КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра
за спеціальністю 121 Програмна інженерія
на тему:

РОЗРОБКА ІГРОВОГО РУШІЯ
З ФІЗИЧНОЮ СИМУЛЯЦІЄЮ ЧАСТОК

Виконав: студент 4-го курсу
Максим ШАТОХІН _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Ярослав ЛІНДЕР _____
(підпис)

Засвідчую, що в цій курсовій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«____» _____ 2021 р.,
протокол № ___

Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

КИЇВ-2021

РЕФЕРАТ

Обсяг роботи 60 сторінок, 25 ілюстрацій, 27 використаних джерел,1 додаток.

Ключові слова: ГРА, ІГРОВИЙ ДОДАТОК, ІГРОВИЙ РУШІЙ, ФІЗИЧНИЙ РУШІЙ, РОЗРОБКА, КОЛІЗІЯ, ДИНАМІКА, ФІЗИЧНЕ ТІЛО.

Об’єктом розробки програмного забезпечення є побудова робочого крос-платформного ігрового рушія з симуляцією фізики часток твердих рухомих тіл.

Метою кваліфікаційної роботи є ознайомлення з процесом розробки ігрових рушіїв, дослідження таких додатків, представлених на ринку, виявлення їхніх переваг та недоліків. Ознайомлення з етапами розробки ігрового рушія, вивчення його складових компонентів. Вивчення законів руху твердих тіл, алгоритмів пошуку колізій, видів зіткнень та варіантів їх вирішення. Використання отриманих знань при створенні власного додатку.

Інструментом створення є інтегроване середовище розробки програмного забезпечення Microsoft Visual Studio 2019. Мова програмування C++. Інтерфейс для мультимедіа-програмування – безкоштовна, вільно поширювана бібліотека SFML. Допоміжний графічний інтерфейс – imGUI.

Результат роботи: досліджена структура ігрового рушія. Розглянуті етапи обробки об’єктів фізичним рушієм. Вивчені процеси руху твердих тіл, алгоритми пошуку колізій. Розглянуті види зіткнень та варіанти їх вирішення. Отримані теоретичні знання використані для розробки власного фізичного ігровий рушій. Реалізовано взаємодію простих фізичних об’єктів, підтримку сили тяжіння, керування фізичним тілом.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. Індустрія відеоігор — одна з наймолодших на ринку. Здається, ще нещодавно невеличка компанія Atari випустила першу у світі гру Pong [1], і ось вже п’ятдесят років потому студенти створюють значно складніші ігри на своїх лабораторних заняттях. За даними з порталу Review42 [2] станом на дві тисячі шістнадцятий рік, у комп’ютерні ігри зіграло близько двох с половиною мільярдів людей, що на той час становило третину населення Землі. При цьому, більш ніж сімдесят відсотків з них, незважаючи на розповсюджені в нашій країні стереотипи, приходяться на дорослу частину населення. У країнах Європи та в Америці відеоігри вже зовсім перестали сприйматися як щось дитяче. Турніри з кіберспортивних дисциплін, де люди змагаються між собою безпосередньо в комп’ютерних іграх, збирають більше глядачів, аніж світові чемпіонати з футболу, а річна виручка індустрії за даними порталу Statista [3] кожного року пране наздогнати одну з найприбутковіших – індустрію кіно.

Виходячи з наведених даних, можемо з великою впевненістю заявити, що, станом на сучасний день, люди прагнуть насолоджуватися відеоіграми. Це підштовхує до думки про розробку власного конкурентоспроможного додатку, яка зазвичай починається безпосередньо з написання ігрового рушія.

Актуальність роботи та підстави для її виконання. Люди вже півстоліття займаються створенням відеоігор, з кожним разом вдосконалюючи методи їх розробки. До дев’яностих двовимірні ігри у жанрі платформер вважались еталонними, і всі продукти, випущенні до того часу, робились схожими за геймплеєм одна на одну, наслідуючи «перевірену формулу», як зазначено в джерелі [4]. Однак все змінилось, коли двоє ентузіастів написали першу тривимірну гру – Doom [5], серія

якої і по сей день вважається культовою. Наразі ж із кожним новим додатком розробники намагаються додати до свого витвору щось нове.

Втім, розробка ігор довгий і важкий процес, який зазвичай займає декілька років, а написання рушія для гри потребує навіть більшу купу часу. Через це компанії-розробники таких додатків намагаються використовувати зручний їм рушій якнайдовше, аби швидше випускати кінцеві продукти. Проте ігровий рушій має задовольняти вимоги ринку та підтримувати той функціонал, який необхідний для розробки масштабних ААА (великих за бюджетом) проєктів. Це призводить до того, що кожного разу старий рушій, наче будинок з надбудовами та прибудовами, доповнюють новими елементами, перетворюючи їх на амальгаму різного коду. Так, одні з найвідоміших ігрових рушіїв, як Unity [6] чи Unreal Engine [7] існують вже більш як десять років, і рішення, які були використані при їх створенні, уже зовсім не актуальні.

Через це більшість незалежних розробників більше схиляються до написання власного рушія, який зміг би задовольнити усі їх потреби та потреби ринку. До того ж, немалою перевагою є і те, що ніхто краще не розбирається у рушії, аніж його розробник, що для останнього може чимало скоротити необхідний для розробки додатку час.

Мета і завдання роботи. Метою кваліфікаційної роботи є ознайомлення з етапами розробки ігрового рушія та аналіз таких продуктів, що вже існують на ринку, дослідження процесів проєктування та архітектурних рішень для додатка, розширення знань в області фізики руху, взаємодії об’єктів та вищої математики, використання здобутого досвіду при написанні власного проєкту.

Об’єкт, методи й засоби дослідження або розроблення. Об’єктом розроблення програмного засобу є створення коректного фізичного простору, підпорядкованому фізичним законам.

Під час розробки використана система руху твердих тіл, що базується на основних законах механіки, відкритих у сімнадцятому сторіччі Ісаком Ньютоном. Вони називаються Трьома законами руху Ньютона й описують взаємодію між об’єктами у системі, та силами, що впливають на цю систему.

Можливі сфери застосування. Кінцевий продукт може використовуватись як програмне забезпечення для створення ігор або помічник для візуалізації фізичної взаємодії тіл.

РОЗДІЛ 1. ЯК ВЛАШТОВАНІ ІГРИ

1.1 Що таке гра?

Термін «гра» відноситься до тих, яким складно дати визначення, адже середньостатистична людина розуміє його на інтуїтивному рівні. Зазвичай перше, що приходить на згадку, намагаючись все ж таки дати тлумачення терміну – ігри, у які ми грали ще в дитинстві, такі як звичайні схованки чи квач, або деякі з азартних ігор, по типу Покера чи Дурня. Також багато людей намагаються описати їхні найулюбленіші настільні ігри (рисунок 1.1), наводячи за приклад шахи чи Монополію. Звісно, не треба забувати про комп’ютерні ігри, гральні автомати чи рулетку, або ж воєнні ігри, які також мають своїх прихильників.

Рисунок 1.1 – Настільна кооперативна гра Древній Жах

Проходячи курс з теорії ймовірності, студенти стикаються з теорією ігор, підрозділом прикладної математики. Вона намагається математично визначити найбільш успішну поведінку суб’єктів у стратегічних ситуаціях, вибір яких безпосередньо залежить від вибору інших учасників. Ця галузь

дає визначення багатьом термінам, які використовуються в іграх, таких як кооперативні/некооперативні, паралельні/послідовні, або ж ігри з нескінченним числом ходів. Вони відкривають завісу на математичну складову ігор та допомагають розробникам чи гейм-дизайнерам зробити ігровий процес цікавим і захоплюючим.

Граючи, ми розгадуємо ці формули, механіку роботи гри та її принципи, які лежать в основі здобуття перемоги. Саме це приносить найбільшу насолоду і є головною складовою гри. Так вважав і Раф Костер, який у своїй книзі «Теорія Веселого Гейм Дизайну» [8] дає їй визначення, як «інтерактивний досвід, який надає гравцеві зростаючу послідовність викликів, підпорядкованих шаблону, подоланню яких обучається гравець, і в кінці кінців майстерно його опановує». У цьому і є таємниця ігор, адже вони, як і жарти, стають веселими тільки коли починаєш їх розуміти.

1.2 Поняття відеогри

Відеоігри насправді існують вже близько сторіччя. Ще у одна тисяча дев’ятсот сорок сьомому році з’являлися такі їх попередники, як шахова комп’ютерна програма чи розважальний прилад з електронно-променевою трубкою [9]. Їх використовували в рамках військового проєкту, мета якого була створити пристрій, що міг би передбачати дії супротивника. Нерідко відеоігри в повсякденному житті люблять називати комп’ютерними іграми, втім, як ми бачимо, вони не тільки існували раніше за персональні комп’ютери, а навіть сприяли розвитку останніх.

На сьогодні існує велике різноманіття відеоігор, проте, незважаючи на їхні особливості, безпосередню більшість з них можна охарактеризувати визначенням, яке дали їм спеціалісти з програмної інженерії [10], а саме – *інтерактивною агентно-модульованою комп’ютерною симуляцією з системою м’якого реального часу*.

Середовище, у якому відбуваються події під час гри, будь то частина реального всесвіту або ж видуманого – математично змодельований простір. Він являє собою апроксимовано спрощену модель реальності, щоб останньою міг легше маніпулювати комп'ютер, адже відтворення поведінки усіх атомів, кварків чи квантів об'єктів – непід'ємна задача для сучасних комп'ютерів. Отже, за своєю сутністю це і є те, що називається *симуляцією*.

Системами реального часу називають такі програми, які повинні оброблювати інформацію у зовнішньому середовищі за певний час, щоб підтримувати з останнім постійну безперервну взаємодію. Такі обмеження суттєво необхідні для відеоігор, адже вони зобов'язані коректно реагувати на усі можливі дії гравця, щоб не порушувати його враження від симуляції. Для охарактеризування задачі такої системи вводять декілька ключових термінів, розглянутих далі.

a. Дедлайн – критичний термін, коли потрібно відреагувати на запит із зовнішнього середовища. Для ігрових рушіїв це ключові обмеження, необхідні для підтримки коректної роботи відеогри, запобігаючи виникненню затримки відображення, що називаються «лагами», або ж неправильну поведінку гри, також відому як «баги». Наприклад, візуальна складова гри або екран повинні оновлюватись тридцять разів на секунду, щоб створити ілюзію руху об'єктів. Таке значення пов'язане з властивістю людського ока розпізнавати до дванадцяти окремих зображень за секунду, а далі, при збільшенні їх кількості, трактувати їх як рух тіл. Фізичне ж ядро, яке буде розглянуте у наступних розділах, має оновлюватись мінімум сто двадцять разів на секунду, щоб забезпечити стабільний фізичний стан об'єктів.

б. Латентність – час, необхідний на відгук системи. Для ігор він варіюється в залежності від системи, яка оброблює запит. Для інтерфейсу користувача цей час повинен бути мінімальним. Зазвичай такі запити виконуються одразу ж, на тому самому фреймі, а ось для зміни локації гри, тобто перебудови усього ігрового простору, може знадобитись набагато більше часу. У таких ситуаціях розробники спрямовують усі ресурси на обробку важкої задачі, а користувач має змогу бачити смугу завантаження.

в. Джитер – небажана затримка при виконанні запиту із зовнішнього середовища. Зазвичай є великою проблемою, адже негативно впливає на враження від ігрового процесу, тому розробники намагаються запобігати його виявленню. Для цього використовується кешування або ж різноманітні алгоритми оптимізації.

Залежно від впливу порушень цих обмежень на симуляцію, відрізняють системи жорсткого та м’якого реального часу. Для них порушення призводить до повної відмови системи, або ж до зниження її якості роботи, відповідно. Через це відеоігри зачисляють до *м’яких*, адже пропущені дедлайни в них не є катастрофічними і ніяк негативно не впливають на гравця.

Агентно-модульована симуляція означає те, що ми розглядаємо об’єкти з певною поведінкою, іменовані агентами, та їх взаємодію один з одним. Цю модель описують як «об’єднане більше за суму окремих частин». Іншими словам, моделюючи нескладну поведінку окремих суб’єктів, можливо створити повноцінну комплексну систему, яка вже, у свою чергу, підпорядковуватиметься складним законам. Така модель явно відслідковуються у тривимірних рольових іграх, де кожен актор у світі має

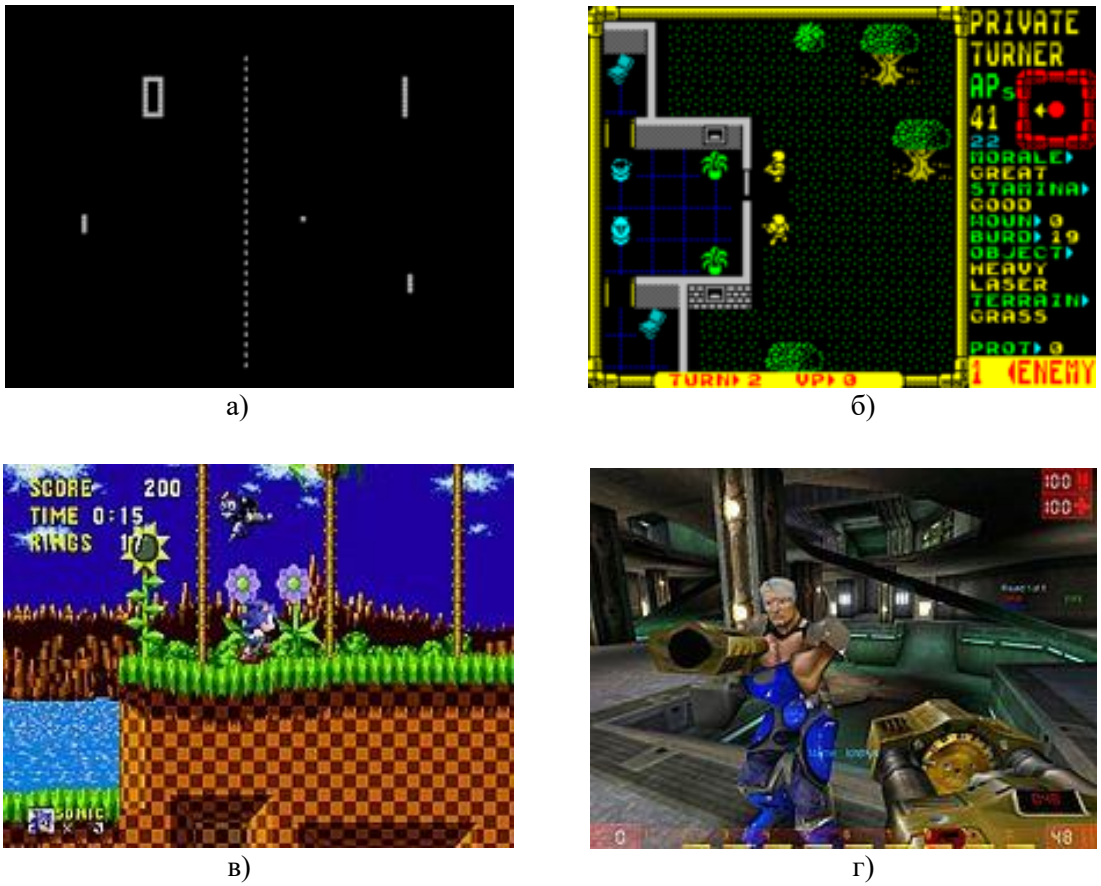
індивідуальну поведінку, що, разом з іншими акторами, доповнює собою екосистему гри.

Здатність адаптуватися, відповідати на дії гравця, підлаштовувати свою історію та світ під глядача у реальному часі робить ігри *інтерактивними*, тобто зосередженими на взаємодії безпосередньо між гравцем і грою.

1.3 Передумови виникнення ігрового рушію

У золоту епоху грального автоматів, коли найпопулярнішими іграми були PacMan та Street Fighter, ніхто не замислювався над можливим відокремленням рушія безпосередньо від гри [11]. Розробка останніх займала небагато часу, максимум до декількох місяців, а випускали їх одноразово на картриджах без можливості модифікації. Технології розробки в ті часи розвивалися з неймовірною швидкістю (рисунок 1.2), і, поспішаючи за новими тенденціями, компаніям було вигідніше писати кожну нову гру з чистого листа.

Термін «ігровий рушій» вперше виник у середині дев'яностих, разом із тривимірною грою Doom, розробленою id Software. Це був другий подібний проєкт компанії, і вони хотіли його розвивати надалі. Через це розробники прийшли до рішення про розподіл додатку на головне ядро та безпосередньо елементи гри. До першого віднесли прорахунок об'єктів у просторі, виведення гри на екран, виявлення зіткнень та аудіо-систему. До другого – ключові механіки та унікальні складові Doom. Це була одна з перших ігор, розроблених під платформу персональних комп'ютерів. Поширювана через мережу інтернет, вона миттєво захопила увагу мільйонів людей по всьому світові. До того ж, при купівлі гри покупець отримував і рушій, який дозволяв будь-кому спробувати себе у ролі розробника відеоігор.



а) б)

в) г)

Рисунок 1.2 – Скріншоти ігор різних десятиріч: а – 70-ті; б – 80-ті; в – 90-ті; г – 00-ві

Надалі почала з’являтися велика кількість ігрових рушіїв, які могли б задовольнити різні потреби розробників, проте ніхто більше не мав бажання викладати їх задарма на увагу публіки. Однак підтримка довголіття гри була можливою тільки за участі фанатів. Наприкінці дев’яностих для гри Quake III була розроблена спеціальна скриптова мова Quake C [12], що дозволило писати гравцям власні скрипти, які могли змінювати логіку оригінальної гри до невпізнаваності. Такі можливості надалі перейняли й інші рушії, і тепер, завдяки системі модифікацій, деякі ігри живуть десятками років.

1.4 Проблема ідеального ігрового рушія

Ігровий рушій – серце і основна частина будь якого ігрового додатку. Він має можливість контролювати всі його складові, через що для розробників спрощується процес створення продукту на низькому, прикладному рівні. Однак межу, що відділяє відеогру від її рушія, не завжди можна виділити прозоро. Один рушій чітко знає, як малювати конкретний ігровий об’єкт, а інший лиш оперує абстракціями, щоб малювати будь-які елементи за допомогою матеріалів або шейдерів. Жорстко вписані елементи в код проєкту робить його подальше використання майже неможливим. Тому при розробці частіше використовують шаблон керованої даними архітектури. Завдяки ньому вихідний код хоч і відокремлений від зовнішніх файлів, проте програма спроєктована так, що її логіка повністю залежить і керується ними. Це те, що зазвичай відрізняє ігровий рушій від безпосередньо гри або ж її елементів.

Наслідуючи архітектуру, керовану даними, розробники намагались зробити ігровий рушій якомога більш абстрактнішим. На жаль, чим більші різновиди ігор вони намагались охопити, тим з більшими проблемами стикалися. Так, наприклад, для коректної роботи гри у закритих приміщеннях треба реалізовувати алгоритм бінарного ділення простору, щоб об’єкти, які знаходяться за стінами або за межами камери, не відображалися і не навантажували процесор. Для ігор на відкритій місцевості треба використовувати інший підхід: зміни рівнів деталізації. Кількість трикутників, що використовують об’єкти поблизу нас, повинна залишатись незмінною, а при віддаленні об’єкта значно зменшуватись, щоб також не навантажувати рушій. Таким чином розробники дійшли до висновку про неможливість створення ідеального ігрового рушія та вирішили, що краще підлаштовувати такі для окремих жанрів ігор (рисунок 1.3).

Рисунок 1.3

На сьогоднішній день різні рушії адаптовані під окремі види ігор, через що ефективніше виконують задачі, необхідні для функціонування відповідних ним додатків. Далі розглянуті найпопулярніші жанри відеоігор.

а. Шутери від першої особи (рисунок 1.4 а) – жанр, у якому, відповідно до його назви, ключовою механікою є орудування та стрілянина зброєю по супротивникам від першої особи. До найвідоміших представників жанру можна віднести Half-Life, Quake, Battlefield, Call Of Duty та серію ігор від українських розробників S.T.A.L.K.E.R. До необхідних технологій при створенні такої гри відносять ефективне опрацювання тривимірного простору, детальну анімацію головного герою та зброї, чутливий контроль камери, миттєвий відгук на введення інформації з контролерів та розвинутий незалежний штучний інтелект.

б. Платформери (рисунок 1.4 б) – жанр, назва якого випливає з головної механіки гри, а саме з вправних стрибків між платформами. Класичними представниками є Super Mario Bros, Celeste та не менш відома Run And Fire. Зі становленням ери тривимірних ігор, асортимент платформерів значно

розширився, включаючи тепер Uncharted, Super Mario World та серію Little Nightmares. До найважливіших задач при створенні такого жанру додатків відносять динамічну фізику об’єктів, підтримку зміни оточення, генерацію випадкових рівнів, кінематичний рух камери та її колізію.

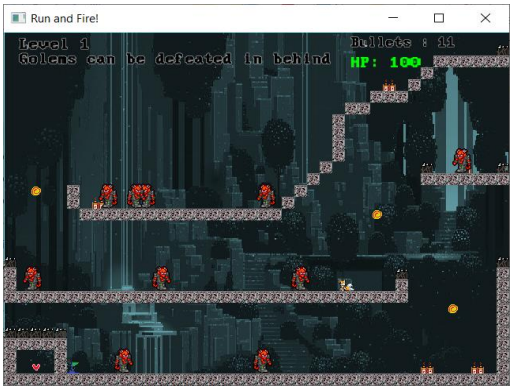
в. Стратегії (рисунок 1.4 в) – ігри, які є нащадками економічних воєнних настільних ігор. У них треба будувати різного роду необхідні споруди, орудуючи своїм військом і доступними ресурсами, та взяти гору над усіма іншими опонентами будь-якими можливими способами. До найвідоміших представників належать Warcraft, Civilization, StarCraft та українська серія Cossacks. До необхідних при створенні технологій відносять підтримку обробки сотень акторів, зміну ландшафту світу та розвинутий штучний інтелект.

г. Масова багатокористувацька онлайн-гра (рисунок 1.4 г) – жанр, до якого можна віднести будь-який додаток, що підтримує одночасне підключення дуже великої кількості гравців на одному сервері. Зазвичай йде в доповнення до іншого виду гри, просто надаючи їй суперництва або ж кооперативності. Найвідомішими прикладами є World of Warcraft, Eve Online, Ever Quest. При створенні таких додатків особливу увагу приділяють одночасній підтримці сесії для багатьох користувачів, синхронізації між ними, наповненню всесвіту та системі динамічних випадкових подій.

д. Інші жанри, такі як симуляції бійок або файтинги, гоночні ігри, додатки з підтримкою віртуальної реальності, пазли та інші. Для кожного з них існують свої необхідні технології та обмеження, такі як точна система пошуку зіткнень для файтингів чи

стереоскопічне відображення для ігор з віртуальною реальністю.

а)



б)

в)

г)

Рисунок 1.4 – Скріншоти відеоігор за жанрами: а – шутер S.T.A.L.K.E.R.;
б – платформер Run And Fire; в – стратегія Cossacks 3; г – МБОГ World of Warcraft

Втім різні за жанром ігри потребують різного часу для їх створення. Відповідно до цього, легких для розробки додатків, таких як головоломки чи платформери, на ринку безліч, а важких для створення проєктів, наприклад, з доповненою реальністю, майже немає. Від цього корелюється і ціна на такі продукти. Згідно порталу HowToMarketGame [13] (рисунок 1.5) прибуток від інноваційних ігор чи якісних ААА проєктів набагато більше, ніж від усіх розроблених платформерів та пазлів разом. За цієї причини більшість розробників рушіїв створюють їх виключно з підтримкою тривимірної графіки, доповненої чи віртуальної реальності та іншими інноваційними інструментами. Останні допомагають

розроблюваним іграм не відступати, або ж навіть опереджати їхніх конкурентів, що стимулює обирати саме цей програмний продукт. Також нерідко в індустрії кіно при відтворенні складних графічних сцен звертаються до ігрових рушіїв і модулюють деякі постанови в них. За цієї причини, багато хто, працюючи над двомірними проєктами, вимушений використовувати великі рушії з безліччю непотрібних для них функцій або ж намагатися створювати свій ідеальний рушій.

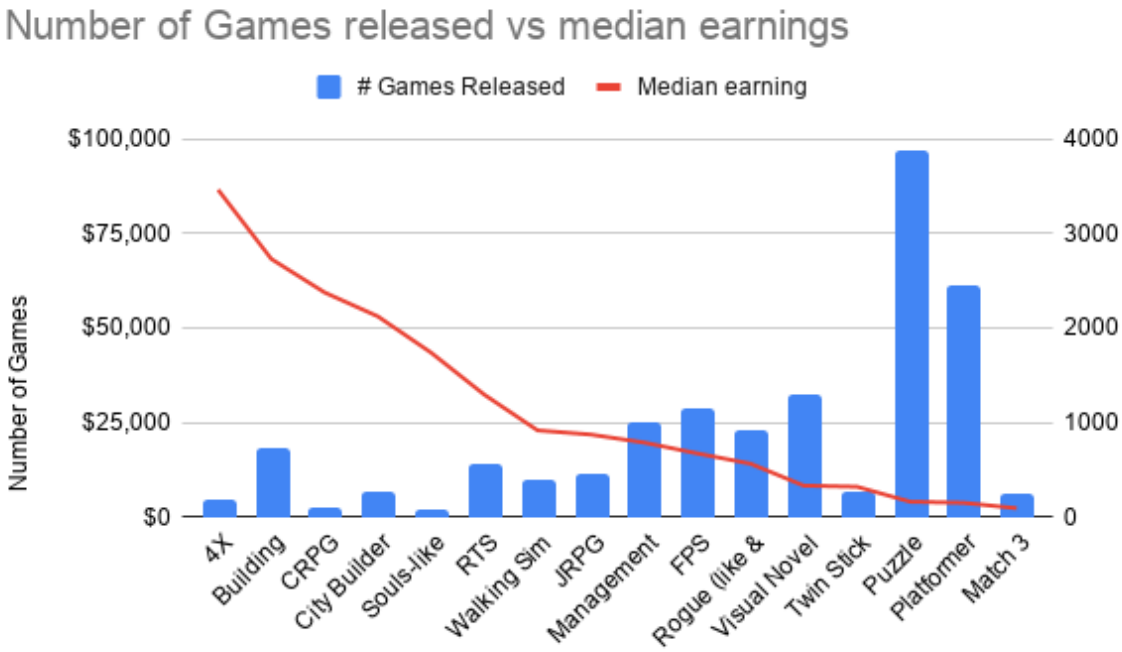


Рисунок 1.5 – Діаграма відповідності прибутку ігор по жанрам

1.5 Огляд існуючих ігрових рушіїв

Сучасна розробка відеоігор завжди починається з обрання чи написання рушія, який міг би задовольнити більшість потреб розробників. Чимало сучасних програм підтримують розробку одразу декількох видів ігор, мають підтримку багатокористувацького режиму, дають можливість писати скрипти у візуальних редакторах та багато іншого. Оскільки метою

даної роботи є створення фізичного рушія для двовимірних ігор, у прикладах були розглянуті популярні додатки, що мають фізичне ядро та підтримують двовимірну графіку.

а. Unreal Engine – крос-платформний рушій, розроблений компанією Epic Games Inc. Перша версія додатку була написана ще у одна тисяча дев’ятсот дев’яносто восьмому році, підтримуючи тільки розробку шутерів від першої особи. Наразі вже існує четверта версія Unreal Engine, а на дві тисячі двадцять другий рік запланований вихід п’ятої. З кожною версією інструменти програми розширюються, додаючи підтримки все більших різновидів ігор, нові технології для роботи з графікою, анімаціями, штучним інтелектом тощо. Одним з головних плюсів додатку є його безкоштовне розповсюдження, що дає змогу спробувати рушій майже будь-кому, а при купівлі ліцензії на рушій розробник отримає доступ до вихідних файлів, які зможе модифікувати згідно його потреб, що дає ще більшу свободу в розробці. Додаток має внутрішню візуальну скриптову мову Blueprints, а також HTML-5 реалізацію для створення браузерних ігор. Він містить інструменти для налаштування звуку, світла, шейдерів, анімацій та фізики. Однак останній підтримує тільки симуляцію жорстких тіл. Ігрова логіка і безпосередньо сам рушій написані за допомогою мови C++. На Unreal Engine написані такі культові ігри як Bioshock і Bioshock Infinite, S.T.A.L.K.E.R. 2, Borderlands та Dishonored.

б. Unity – багатоплатформовий рушій, створений компанією Unity Technologies. Він був випущений у дві тисячі п’ятому році, маючи за мету залучити якомога більше людей до індустрії розробки відеоігор. Через це рушій був доступний майже на всіх

відомих платформах того часу, поширювався умовно безкоштовно, включав дружній графічний інтерфейс та для розробки використовував більш простіший C#, а не C++, на якому був написаний. Окрім головних функцій, як підтримка фізики, анімацій, динамічного освітлення додаток також має багато інших потужних інструментів. Таких, наприклад, як редактор інтерфейсу користувача (рисунок 1.6), генератор ландшафтів, компресія текстур, шейдерів та підтримка компіляції у JavaScript, що дає змогу завантажувати ігри у браузерах. Завдяки зручному інтерфейсу та інтуїтивному дизайну Unity отримав велику популярність серед починаючих незалежних та мобільних розробників. Так, станом на 2018 рік на даному рушії було написано близько шістдесяти відсотків мобільних ігор. Серед найпопулярніших додатків можна відмітити Cuphead, Hearthstone, Monument Valley та Pokémon GO.

Рисунок 1.6 – Користувацький інтерфейс в Unity

в. CryEngine [14] – рушій для створення ігор на персональні комп’ютери та консолі, розроблений компанією Crytek. Він був створений у дві тисячі другому році для демонстрації нових технічних можливостей відеокарт NVIDIA, проте розробники побачили в додатку потенціал і зробили з нього повноцінний рушій з багатьма новими для того часу функціями. Так і нині CryEngine має в своєму розпорядженні чимало унікальних потужних інструментів. До них можна віднести автоматизоване створення маршрутів і доріг, редактор симуляції рідин, інструмент для створення анімацій обличь, систему процедурних анімацій, опромінюючий звук, глобальне освітлення в реальному часі, редактор матеріалів та багато іншого. Однак через таку велику кількість потужних систем зросла і складність для опанування рушія, через що його використовують зазвичай тільки для крупних AAA проєктів, таких як, серія Far Cry, Crysis, Warface, Prey та The Climb. Для написання ігор використовується мова C++, на якій розроблено і сам рушій, а для написання скриптів останній має інтерпретатор мови Lua.

г. UbiArt [15] – крос-платформний рушій, розроблений компанією Ubisoft Montpellier у дві тисяча десятому році. Його ключовою відмінністю була підтримка векторної графіки та шістдесяти кадрів за секунду при розширенні Full HD. Однак через свою незвичність, пов’язану з унікальною обробкою графіки, їм було важко користуватися. Також рушій не передбачав створення скриптів за допомогою спеціальних мов, внутрішнього редагування матеріалів та подібних інструментів, що надавали інші додатки. Фізичне ядро рушія підтримувало симуляцію жорстких тіл, проте через його унікальну графіку він

приглянувся деяким розробникам, і з його допомогою вийшли такі чудові ігри як Gravity Falls, Child of Light, Valiant Hearts: The Great War.

Як ми бачимо, більшість рушіїв зазвичай використовують стандартну фізичну симуляцію жорстких тіл, а упор роблять на графічну складову гри чи інноваційні інструменти розробки.

1.6 Висновки розділу

Людство грає в ігри з давніх-давен, а їх відео адаптації захопили світ менш ніж за сторіччя. Наразі ринок налічує безмежну кількість ігор різних жанрів, і розробка кожної починається з обрання чи написання рушія. Останніх існує дуже багато, у кожного є свої плюси та мінуси, однак майже жоден з них не робить детальну симуляцію фізики, віддаючи перевагу графічній складовій.

РОЗДІЛ 2. АРХІТЕКТУРА ІГРОВИХ РУШІЇВ

2.1 Ігровий рушій та його складові

Для будь якої сучасної ігри рушій грає головну роль. Він бере під контроль відображення сцен, генерацію звуків, імітацію штучного інтелекту, перебіг анімацій, мережеву складову, допомагає гейм-дизайнерам наповнювати ігровий світ життям, створювати сценарні повороти та багато чого іншого. Ігрові рушії є головним програмним забезпеченням для створення відеоігор, настільки важливим, що вони нерідко порівнюються з інтегрованим середовищем розробки для програмістів, без якого було б неможливо писати програми.

Складаються рушії з багатьох складних систем, кожна з яких має свій певний набір функцій та відокремлену область впливу. За головною приналежністю в джерелі [16] їх розділяють на дві головні частини: ядро реального часу та редактор з інструментами. Перше бере під контроль обрахунок ігрових елементів, відтворення різноманітних ефектів та створення усього, що потребується під час гри. Другий, відповідно, надає розробникам інструменти для створення гри, такі як імпортер зовнішніх файлів, редактор світу, редактор матеріалів і шейдерів тощо.

2.2 Системи реального часу

Коли ми запускаємо та граємо в будь-яку гру, все, що ми спостерігаємо на екрані, відбувається за рахунок роботи систем реального часу рушія. Уявімо головного героя шутеру, що висувається з укриття та отримує кулю від ворожого персонажа чи NPC. Для такої простої послідовності дій система вводу-виводу має відреагувати на кнопки, натиснуті гравцем, та передати менеджеру з обробки подій. Він, у свою

чергу, розішле інформацію про введення до менеджеру гравця. Останній надішле команду на початок руху актора, запуск анімації висування з укриття та відтворення звуків кроків. У цей час паралельно буде оброблюватись ядро штучного інтелекту. Воно на кожному такті розсилає всім NPC, що знаходяться недалеко від головного героя запит на оновлення своїх сенсорів почуттів. Один з агентів помітить, що тепер бачить ворожого йому гравця, та відкриє вогонь. Запустяться звукові ефекти пострілу, візуальні світлові ефекти, анімація стрільби. У світі створиться модель кулі, що отримає імпульс для польоту в сторону гравця. Запуститься простежування на колізію кулі й персонажу. Фізичний рушій обрахує, що об’єкти перетинаються, та відправить подію про влучення до менеджеру гравця, який через графічний інтерфейс виведе відповідне повідомлення на екран гри. Усі залученні до прикладу процеси підкорюються системам реального часу. Для спрощення були розглянути лише системи верхнього рівня.

Як і більшість програмного забезпечення, системи реального часу ігрового рушія поділяються на підпорядковані прошарки. Усі верхні шари залежні від нижніх, але не навпаки. Чим система нижче, ти більш низького рівня задачі вона вирішує. Повна ієрархія функціональних частин рушія наведена в додатку А. Далі розглянемо кожну з частин детальніше, починаючи з найнижчої.

2.2.1 Цільове апаратне забезпечення

Цей прошарок відповідає за платформу, на якій плануються випускати ігри, створені на цьому рушії. Прикладами є персональні комп’ютери, смартфони, планшети, консолі та приставки. Більшість програмного забезпечення платформонезалежні, проте цільове апаратне забезпечення треба брати до уваги, адже від нього чимало залежить

користувацький інтерфейс. Для персональних комп’ютерів треба реалізовувати підтримку мишки з клавіатурою, а для консолей – геймпадів та джойстиків. Більшість комерційних ігрових рушіїв підтримують усі наявні платформи та, при появі нових, намагаються своєчасно випускати оновлення.

2.2.2 Драйвери

Драйверами називають програмне забезпечення низького рівня, яке надає змогу зовнішнім програмам керувати апаратним прошарком. Відповідно драйвери залежать від апаратних компонентів, що встановлені на пристрої. Майже кожна складова платформи, будь то відеокарта чи комп’ютерна миша, має драйвер, що надає спрощений доступ до її функцій. Наразі розробники програмного забезпечення намагаються уніфікувати їх. Наприклад, більша частина відеокарт NVIDIA [17] підпорядковані драйверу з однаковою архітектурою, тому, розроблюючи програмне забезпечення під них, беруть до уваги тільки драйвери, незважаючи на відмінності в апаратній складовій.

2.2.3 Операційні системи

Операційною системою називають головну виконуючу програму або комплекс програм, завдяки яким можливе керування апаратною складовою платформи через взаємодію з користувачем. Прикладами таких для персональних комп’ютерів є Linux, Microsoft Windows, MacOS, для смартфонів – IOS та Android. Сучасні операційні системи застосовують примітивну багатопотоковість, яка дозволяє підтримувати одночасне виконання декількох програм. Через це треба враховувати, що не вся обчислювальна потужність буде задіяна для підтримки гри. Також треба

пам’ятати, що системні виклики кожної операційної системи різні. Тому, щоб писати крос-платформний рушій, треба мати реалізацію низькошарових функцій під кожну платформу. Більшість з них мають однакові ядра, наприклад Linux та Android, які є UNIX-системами, через що випускати програми під них одночасно простіше.

2.2.4 Комплекти стороннього забезпечення для розробки

Нерідко для спрощення роботи над проєктом розробники долучаються до стороннього програмного забезпечення – SDK.

Для більш розвинутого використання контейнерів, спрощеного обрахування лінійної алгебри, використання складних алгоритмів зазвичай використовується набір C++ бібліотек Boost. Він охоплює такі розділи програмування, як метапрограмування, графи, юніт-тестування, контейнери, чисельні і математичні алгоритми, обробка тексту й рядків тощо.

Для простішої обробки графіки та роботи із шейдерами зазвичай використовують OpenGL, або більш сучасніший DirectX, який наразі є одним з найпоширеніших графічних комплектів через свою зручність у використанні. Альтернативою до них є Vulkan – бібліотека низького рівню, яка дозволяє розбивати процес рендерингу на партії, мінімізувати використання відеопам’яті та ресурсів.

Найпопулярнішими інструментами для симуляції фізики та кінематики твердих тіл є Havok [18] та PhysX [19], які вже зарекомендували себе як потужні фізичні рушії. Розробники, які ж хочуть більш вільно орудувати фізичними моделями та змінювати їх поведінку, обирають Open Dynamics Engine, адже він є безкоштовно поширюваним інструментом із відкритим для модифікації вихідним кодом.

Також існують готові комплексні бібліотеки, що покривають зв’язок з більшістю систем низького рівня. Найвідомішою серед них для роботи з двовимірною графікою, мережею та звуком є SFML [20]. Вона також безкоштовно поширювана, з відкритим вихідним кодом та адаптована під роботу з деяким іншим програмним забезпеченням.

2.2.5 Крос-платформний прошарок

Для зручного використання функцій операційної системи та апаратних частин, доступ до них виносять у спеціальні бібліотеки низького рівня. У них розробники описують взаємодію з усіма платформами, що мусять підтримувати рушій, реалізовують зручну обгортку для функцій зі сторонніх інструментів та описують необхідні їм для розробки методи. Таким чином, цей прошарок створює програмний інтерфейс додатку або API, який потім використовується в усіх підсистемах рушія.

2.2.6 Базові системи

Кожен рушій – дуже велика комплексна програма, що об’єднана багатьма системами, кожна з яких потребує для коректної роботи великої кількості інструментів. При розробці вони виносяться в окремий програмний розділ, що зазвичай називають утилітами. Розглянемо найважливіші з них.

a. Повідомлення та помилки. Для зручної розробки та налагодження програми розробникам часто потрібно відстежувати некоректну роботу додатку та сповіщати про помилки відповідними повідомленнями. Необхідно описати спеціальні макроси, що будуть змушувати програму призупинити виконування, імітуючи роботу точки зупину. Це

дасть можливість зручно переглянути стек викликів у момент поломки. Необхідно також забезпечити підтримку профілювання інформації, щоб у разі критичного збою програми можна було знайти його джерело.

б. Математична бібліотека. Ігри за означенням математично-модульовані симуляції, через що ігровий рушій повинен надати розробникам зручні інструменти для маніпуляції математичними примітивами. Як мінімум, повинні підтримуватись операції над векторами, матрицями, кватерніонами, кутами Ейлера, лініями, променями, площинами, сферами тощо. Для наближених обчислень описати маніпуляції із кривими та сплайнами, інтерполяції примітивів, інтегрування за Ріманом, і так далі. Обов’язково необхідною також є бібліотека для детермінованого отримання псевдовипадкових чисел.

в. Спеціальні типи даних та алгоритми. Якщо розробник рушія не покладається цілком на сторонні бібліотеки він повинен описати самостійно базисні структури низького рівню. До них відносяться дерева, мапи, динамічні масиви тощо. Слід забезпечити їх надійну роботу, описати алгоритми для роботи з ними та мінімізувати пам’ять для їх використання.

г. Парсери та файли конфігурації. Рушій повинен надавати можливість опрацьовувати зовнішні текстові файли та зчитувати з них спеціальні типи даних. Також додаток зобов’язаний мати файли конфігурації для налаштування константних параметрів рушія, зміни мапи клавіш для вводу, максимальну кількість ресурсів, що він має змогу використати тощо.

2.2.7 Менеджер ресурсів

Рушій повинен надавати можливість керувати будь яким ігровим ресурсом, використовуючи єдиний інтерфейс чи декілька з них. Кожен ігровий об’єкт чи компонент повинен відслідковуватися збирачем сміття, або ж самостійно видалятися, якщо на нього немає активних посилань. Користувач повинен мати змогу запросити як синхронне, так і асинхронне завантаження будь-якого зовнішнього файлу чи створення ігрового об’єкту. Асинхронність дуже важлива для ігрових рушіїв, адже завантажуючи всі об’єкти на старті, ми дуже сильно збільшуємо використання оперативної пам’яті, а при завантаженні ресурсів синхронно під час гри, остання буде простоювати на момент завантаження об’єкту.

2.2.8 Система рендерингу

Система рендерингу є однією з найскладніших та комплексних з усіх наявних у рушії. Вона повинна відображати все необхідне на екрані, відстежуючи об’єкти поза полем зору та ті, що скриті іншими об’єктами, щоб не малювати їх. Рендеринг, як і безпосередньо рушій, складається з декількох підпорядкованих прошарків, які розглянуті далі.

a. Відображення низького рівню. Цей прошарок відповідає за виведення на екран всіх примітивів рендеру. До них відносяться точки, лінії, трикутники, овали та інші прості геометричні фігури. Кожен такий примітив подається до прошарку з відповідним йому матеріалом та кількістю освітлення, що він отримує. Спершу система формує площину з вершин, спрощує її та перетворює на піксельне зображення шляхом растеризації. Потім зчитує матеріал фігури, який описує, як саме текстура накладається на кожну вершину об’єкту, додаючи, за

необхідності, відповідний шейдер. Потім обраховується кількість світла, що припадає на кожну з вершин примітиву, та накладається відповідний ефект освітлення до останнього. Цей процес висвітлений на рисунку 2.1 [21]. Задача цього прошарку – відобразити всі об’єкти найдешевше та найшвидше;

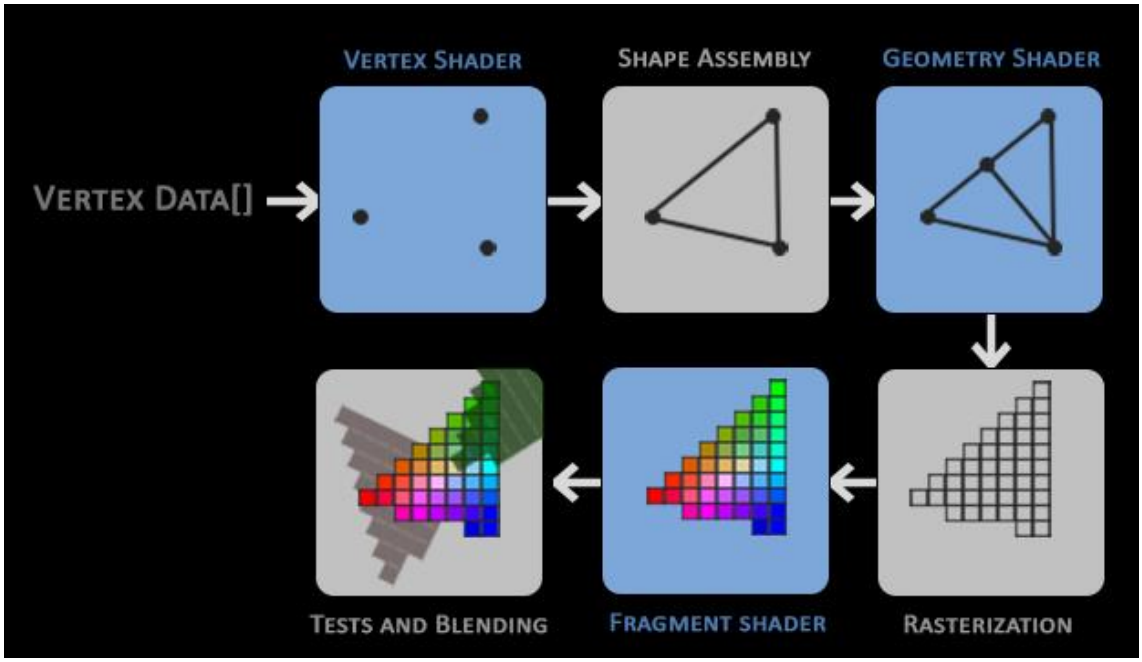


Рисунок 2.1 – Процес обробки примітивів системою рендерингу низького рівня

б. Граф сцени та буферизація. Якщо низький рівень виводить усі надані йому примітиви, замислюючись тільки про те, як зробити це найшвидше, мета цього прошарку – зменшити кількість об’єктів для відображення. Це досягається за рахунок будування графу сцени. Простір рекурсивно розбивається над підпростори, кожен з яких містить відповідні їм об’єкти на сцені. Таким чином, використовуючи спеціальні алгоритми можна швидко відділити об’єкти, що не потрапляють у область бачення та ті, що перекриваються іншими об’єктами. Також цей прошарок відповідає за буферизацію – процес послідовного відображення сцен. Поки відображається перша сцена, рушій малює другу, а

потім міняє їх місцями. Таким чином ми бачимо результат підміни повного зображення новим суцільно готовим зображенням, а не процес малювання примітивів;

в. Візуальні ефекти. Цей прошарок відповідає за відображення тих сутностей, наявність яких не впливає на інші об’єкти у світі. До них відносять частки (дим, вогонь, бризки), наліпки (сліди куль, кроків), динамічні тіні та динамічне освітлення;

г. Графічний інтерфейс. Майже всі ігри містять в собі деяку двовимірну графіку, що називається графічним інтерфейсом. Сюди відносять:

1) прозорий ігровий дисплей, що відображає деякі умовності гри, як кількість здоров’я, патронів тощо;
2) внутрішнє меню гри, що зазвичай містить кнопки обрання рівню гри, складності, зміни конфігурації, старту та інші;
3) вбудований ігровий інтерфейс, який безпосередньо є елементом гри. Сюди відносять відображення інвентарю, вікно купівлі чи продажу предметів, тощо.

2.2.9 Система колізій та фізика

Виявлення колізій – важлива функція для будь якої гри, адже без неї об’єкти будуть просто перетинатися, не взаємодіючи один з одним, через що буде неможливою спроба вплинути на ігровий світ. Більшість ігрових рушіїв зазвичай реалізовують лише динаміку жорстких тіл. Так називається взаємодія об’єктів за допомогою сили, моментів та імпульсів. Для більшості ігор цього достатньо, адже за допомогою наведених параметрів можна легко впливати та фізично рухати тіло у просторі. Наразі майже немає великих ігрових компаній, що реалізовують свою фізичну

систему, а не використовують сторонні бібліотеки. Більш детально фізику в іграх буде розглянуто у наступному розділі.

Рисунок 2.2 – Приклад налаштованої колізії у відеогрі Street Fighter III, фізичні тіла позначені прямокутниками

2.2.10 Анімаційний рушій

Будь-яка сучасна гра, яка містить органічні об’єкти, повинна мати систему анімацій, щоб персонажі в середині гри виглядали природньо. Серед типів анімацій, що використовуються в ігрових рушіях, виділяють наступні:

а. Анімація спрайтами – вид двомірної анімації, коли малюнок об’єкта циклічно змінюється іншим видозміненим малюнком, імітуючи рух тіла. Такий різновид графіки застосовується в усіх двомірних іграх, а також для спрощених тривимірних моделей та рідше для графічного інтерфейсу.

б. Анімація фізичного тіла – вид анімації, яка отримується шляхом надання фізичному об’єкту, що складається з кількох рухомих

частин, імпульсу чи сили. Зазвичай вона використовується для імітації руху рослинності чи знерухомлених персонажів.

в. Скелетна анімація – вид анімації, яка отримується шляхом послідовної зміни суглобів спрощеного скелету персонажа. Кожному актору, якого треба анімувати, призначається скелет з відповідними кістками. Його можна змінювати, обертаючи суглоби, тим самим отримуючи нові пози актора (рисунок 2.2) [22].

г. Анімація вершин – вид тривимірної анімації, при якому змінюється позиція, обертання та масштаб вершин тіла, надаючи йому нової форми. Такий тип використовується для анімації обличь персонажів у тривимірних відеоіграх та для зміни форми статичних ігрових об’єктів.

Рисунок 2.3 – Приклади поз для скелетної анімації людини

2.2.11 Системи введення

Оскільки ігри – інтерактивні системи, вони повинні постійно тримати зв’язок із гравцем. Рушій повинен підтримувати доступні пристрої

людського інтерфейсу, або HID. До них відносять клавіатуру з мишею, геймпади з джойстиками та спеціалізовані контролери як кермо, танцювальна підлога тощо. Рушій має забезпечувати можливу зміну відповідності функцій та відповідних їм кнопок на пристроях введення.

2.2.12 Аудіосистеми

Звук оточує нас все життя. Ми звикли прокидатися за покликом будильнику, відволікатися на телефон, коли чуємо сигнал повідомлення, чи задирати голову на шум літаку. Так само й відеоігри, намагаючись підтримувати постійний зв'язок з гравцем, повинні повідомляти його про зміни світу, у який він занурився.

За часів перших рушіїв системи аудіо були дуже простими, вони мали здатність лише відтворювати прості звуки та мелодії. У сучасних додатках звукові системи мають можливість відтворювати різноманітний звук в залежності від приміщення, розподіляти його на декілька шарів та доріжок, щоб потім комбінувати, задавати напрям розповсюдження звукової хвилі та ще багато іншого.

2.2.13 Ігровий світ та його об'єкти

Кожен об'єкт, видимий і невидимий, що нас оточує у віртуальному всесвіті, являє собою окреме тіло. Вони мають свої задачі, які повинні оброблюватися різним чином. Основними ігровими об'єктами за типами є наступні:

а) статичні моделі – будівлі, дороги, ландшафти, рослини та інші нерухомі об'єкти;
б) динамічні моделі – камінці, стільці, чашки, вазони та інші будь-які об'єкти, що повинні бути рухомі гравітацією;

в) неігрові персонажі – будь-які істоти, які повинні рухатися та мати штучний інтелект;
г) головний гравець – модель, якою керує ззовні користувач;
д) об’єкти взаємодії або тригери – сутності, з якими гравець має змогу вступити в контакт, чим вплинути на ігровий процес;
е) джерела світла – об’єкти, що можуть динамічно впливати на освітленість оточення;
ж) камери – об’єкти, які проектують ігровий світ на екран гравця.

2.2.14 Оброблювач подій

Усі системи ігрового рушія тісно пов’язані одне з одною. Нерідко, коли для виконання однієї функції системи, їй необхідно зачекати відповіді від методу іншої. Так, наприклад, ми не можемо змінювати освітлення на нічне та відображати місяць замість сонця поки менеджер часу не сповістить усіх про початок вечору. Через це, ще у перших ігрових рушіїв був менеджер подій, що за єдиним зручним інтерфейсом позволяє передавати будь-які повідомлення до будь-якої системи.

2.2.15 Штучний інтелект

Розробники намагаються зробити ігри більш реалістичними ще з часу їх виникнення. Щоб якомога більше наблизитися до останнього, вони вдосконалюють різні алгоритми й підходи до керування штучними агентами, аби їхні дії походили на людські. Наразі існують мови програмування, що надають можливість командами та простими послідовностями дій керувати штучним інтелектом. До найбільш відомих методів описання поведінки таких відносять Goal Agent Programming

Language [23], Stanford Research Institute Problem Solver, Planning Domain Definition Language та False Discovery Rate.

Поведінка штучних агентів поділяється на декілька етапів, ключовими з яких є:

а) обробка задач – процес виявлення усіх цілей, які необхідно задовільнити агентові;
б) планування – процес, коли визначається задача з найбільшим пріоритетом, яку агент здатен виконати. Створюється послідовність кроків або базових задач, що повинен виконати агент для завершення його головної мети;
в) виконування – послідовне завершення поставлених задач попереднього етапу. У разі неможливості успішного завершення цілі, агент повертається до планування.

2.3 Системи розробки

Для того щоб привести в дію системи реального часу, тобто пограти в гру, спочатку необхідно буде її створити. Прописувати в коді кожну ігрову механіку та взаємодію – дуже складний процес, який потребує багато часу або великої кількості програмістів. Для вирішення цієї проблеми сучасні ігрові рушії надають інструменти внутрішньої розробки або редактор. Надалі розглянемо його основні частини.

2.3.1 Інструмент менеджменту ресурсів

Для роботи багатьох систем рушія необхідні зовнішні файли, адже ми не можемо розроблювати звукову систему без звуку або рендеринг без текстур. Тому ігровий рушій повинен надавати можливість використовувати всі необхідні зовнішні ресурси для розробки. Зберігають

їх у окремих файлах, які враховують можливі унікальні налаштування та параметри, версію вихідного файлу імпорту та шлях до нього, версію рушія та інше. Також такому об’єкту присвоюється унікальний ідентифікатор, за яким інші ресурси можуть отримати швидкий доступ до нього.

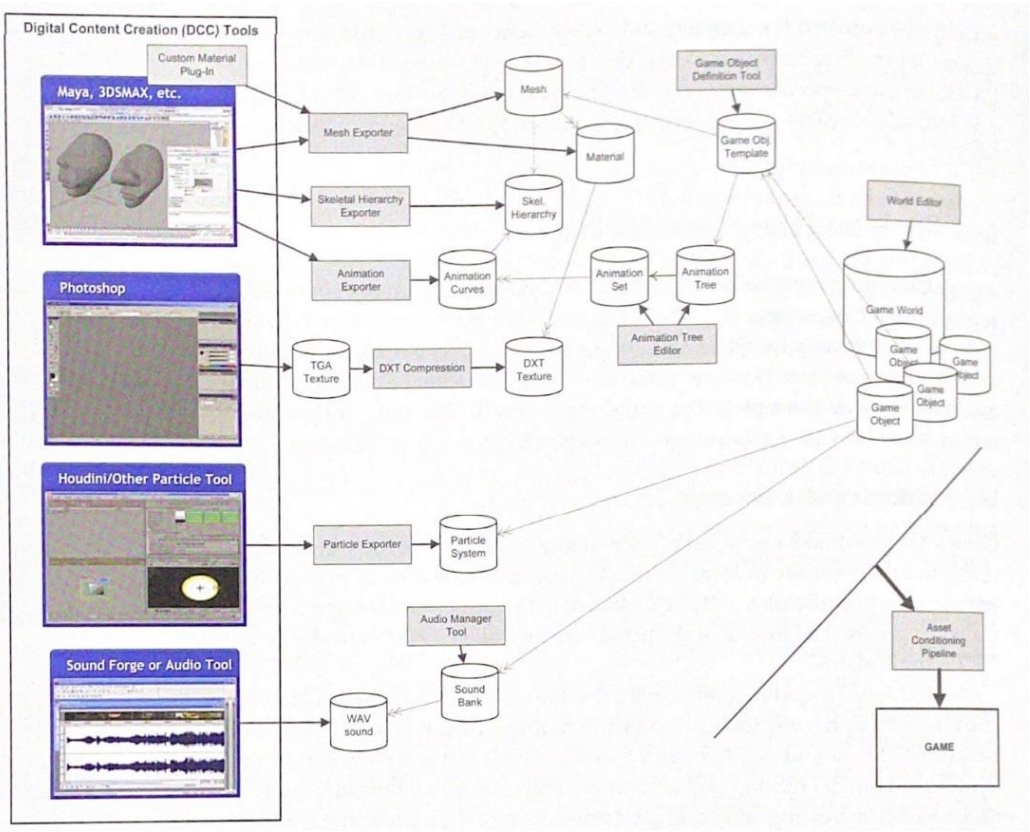


Рисунок 2.4 – Процес експорту до рушія ресурсів із зовнішніх програм

2.3.2 Редактор світу

Ігровий світ це головне місце, де збираються разом усі елементи відеогри, а щоб налаштувати їх, розставити по місцях, наділити властивостями, текстурами, поведінкою необхідний редактор. Він надає розробнику легко знаходити доступ до ключових елементів гри та створювати їх. Завдяки поліморфізму редактор оброблює всі об’єкти

одним чином, дозволяючи без великих зусиль створювати нові компоненти та використовувати їх у редакторі.

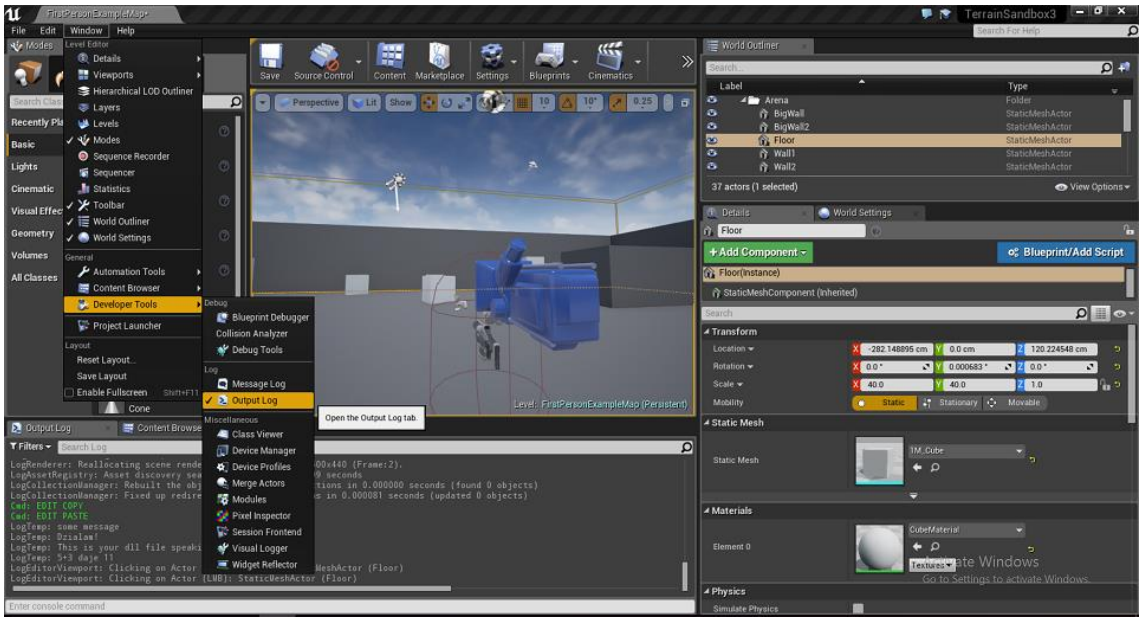


Рисунок 2.5 – Скрішнот редактору світу в рушії Unreal Engine 4

2.4 Висновки розділу

Ігровий рушій – комплексна система, що складається з багатьох прошарків, кожен з яких має свою унікальну важливу функцію. Їх, у залежності від етапу використання, поділяють на системи реального часу та системи розробки. Перші виконуються під час гри, а, відповідно, другі – під час розробки. Охопити та відтворити всі розділи рушія можуть тільки великі компанії із багатьма спеціалістами. Незалежні розробники частіше зосереджують увагу на декількох важливих системах рушія, що необхідні для конкретної гри.

РОЗДІЛ 3. ФІЗИЧНИЙ РУШІЙ

3.1 Задачі фізичного рушія

Реальні фізичні тіла це тверді об’єкти, що мають здатність зіштовхуватися та взаємодіяти на молекулярному рівні один з одним. Моделювати їх детальну поведінку наразі неможлива задача, через що у симуляціях фізику спрощують, надаючи об’єктам для взаємодії більш прості геометричні форми. Проте рушій, наважаючи на такі обмеження, усе ще повинен забезпечувати їх можливістю фізично впливати один на одного. Для відтворення таких симуляцій використовують чисельні методи, що надають наближений результат того, що ми б мали змогу спостерігати в реальному світі. Ще перші комп’ютери, які використовували військові, намагалися вирішувати подібні задачі, наприклад, обчислювати траєкторію польоту ракети. З того часу навички симулювати фізичні тіла значно збільшилися, сьогодні вчені мають здатність відтворювати польоти ракет та супутників у космосі чи навіть на інших планетах.

Фізичним рушієм називають таке програмне забезпечення, що здатне відтворювати фізичну симуляцію [24]. Воно крок за кроком оброблює всі тіла, прикладаючи до них силу, рухаючи та вирішуючи конфлікти при перетині (рисунок 3.1) За рішенням таких проблем стоять дві великі задачі, а саме прорахунок динаміки твердого тіла та виявлення зіткнень.

3.2 Динаміка твердого тіла

Рух твердого тіла може бути змодельований за правилами Ньютонівської механіки [25]. Вони називаються Трьома законами руху

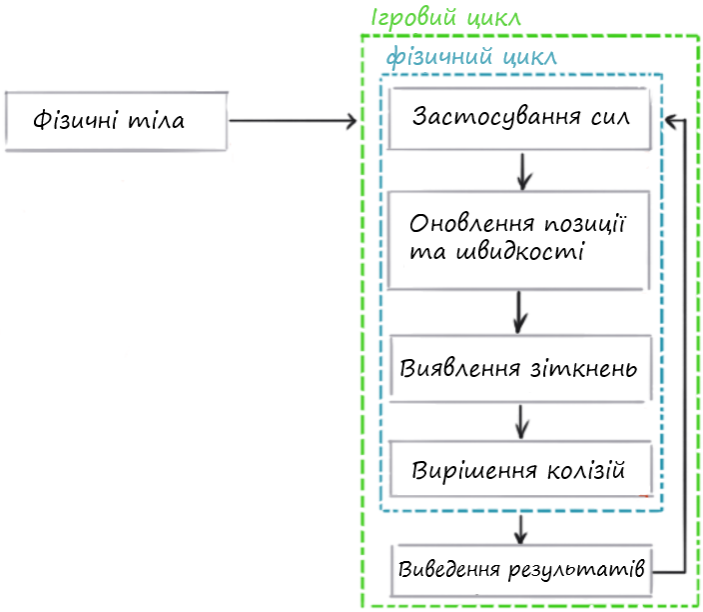


Рисунок 3.1 – Етапи роботи фізичного рушія

Ньютона й описують взаємодію між об’єктами у системі, та силами, що впливають на цю систему:

а) інерція – якщо до тіла не прикладають жодної сили, то його швидкість та напрям не повинні змінюватися;

б) прискорення, сила та маса – векторна сума сил, що впливають на тіло, дорівнює масі цього тіла, помноженій на його прискорення;

в) дія та протидія – кількість сили, що прикладає один об’єкт до другого, завжди дорівнює силі, з якою другий об’єкт діє на перший.

Дотримуючись цих трьох законів, можна описати фізичну систему, що буде подібна до реальності, тож розглянемо її об’єкти детальніше.

3.2.1 Симуляція часток

Симуляція руху часток – одна з найлегших задач фізичного рушія, адже вони мають лише позицію у просторі, швидкість та масу. Користуючись першим законом Ньютона, маємо, що поки на частки не діють зовні сили, то її швидкість буде змінюватися лише при взаємодії з іншими об’єктами, інакше ж її позиція буде змінюватися постійно.

Для симуляції створюється масив часток, кожна з яких має свої параметри: фіксовану масу, початкову швидкість та позицію у просторі. Далі, на кожному кроку рушія, обраховуються сили, що діють на частку, за другим законом Ньютона знаходиться прискорення та оновлюється швидкість. Використовуючи нові параметри частки, обраховується її позиція. Нехай час на оновленні рушія дорівнює t_i, маса частки позначається як m, позиція як $p(t_i)$, сила $F(t_i)$ та швидкість $v(t_i)$. Тоді маємо формулу для оновлення тіла:

$$dt = t_{i+1} - t_i$$

$$v(t_{i+1}) = v(t_i) + \left(\frac{F(t_i)}{m}\right) dt$$

$$p(t_{i+1}) = p(t_i) + v(t_{i+1})dt.$$

Таким чином задається рух простих часток у просторі (рисунок 3.2). Цей метод обрахунку називають неявним Ейлеровим, його використовують більшість фізичних рушіїв, зважаючи на його простоту та відносну точність.

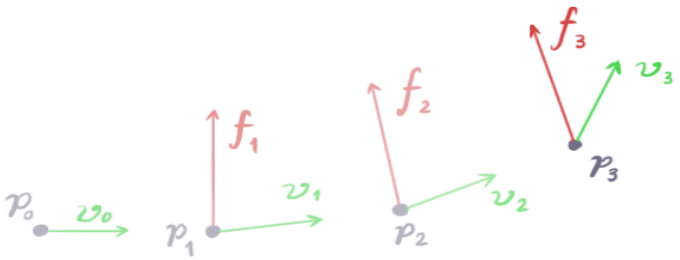


Рисунок 3.2 – Рух фізичної частки в просторі

3.2.2 Симуляція руху твердих тіл

Твердим тілом називають суцільний об’єкт, що не має змоги змінювати своєї форми. Такі об’єкти не існують у реальності, адже навіть найтвердіші тіла підвладні деформації при достатньо великій кількості сили. Проте для симуляції їх наявність допускається, адже в такому разі ми нехтуємо зміною форми і спрощуємо обрахунки.

Тверде тіло схоже з часткою, адже теж має позицію, масу та швидкість із прискоренням, проте, на відміну від частки, воно також має форму і здатність обертатися. Точкою, навколо якої відбувається обертання об’єкта, називають його центром маси, і зазвичай від неї рахують позицію цього тіла у просторі. Однак центр мас обраховується за складною формулою, через що в симуляціях частіше використовують поняття центроїду – геометричного центра тіла. Для об’єктів із суцільною густиною його положення співпадає з центром мас, через що вводять припущення, що в симуляції беруть участь саме такі тіла. Для кожної простої геометричної фігури вже існують описані формули знаходження положення центроїду, наприклад для трикутника це

$$x_c = \frac{b + b1}{3}$$

$$y_c = \frac{h}{3},$$

де b це основа трикутника, h – висота, а

$$b_1 = \frac{h}{\tan{(a)}},$$

де a це кут між основою трикутника та стороною, що знаходиться ближче до початку координат. Складні поліноми можна розбити на прості фігури, зазвичай це трикутники, та знайти їх геометричний центр за формулою:

$$C_x = \sum_i^n A_i y_{c,i}$$

$$C_y = \sum_i^n A_i x_{c,i},$$

де A_i це площина простої фігури, а n – їхня кількість.

Для представлення обертання, як і в аналогії зі швидкістю, вводять поняття сили, яку називають моментом оберту τ та подібно масі – моментом інерції I. Виходячи з другого закону Ньютона, маємо формулу:

$$\tau = I \propto,$$

де $\propto$ це кут, на який відбувається обертання.

Момент інерції обраховується через центр мас, тобто центроїд для фізичних симуляцій. Тому, аналогічно до обрахунку геометричного центру, у симуляціях використовують уже описані формули для пошуку моменту інерції простих фігур. Для прямокутника, наприклад, це:

$$\mathrm{I} = \frac{m(h^2 + w^2)}{12},$$

де h це висота прямокутника, а w – його довжина.

Момент оберту ж можна знайти за формулою:

$$\tau = \big||r|\big|\,\big||F|\big| \sin\theta = r_x F_y - r_y F_x,$$

де θ це найменший кут між векторами, а r – вектор, до якого прикладається зовнішня сила.

Використовуючи дані формули, можна виразити $\propto$, і таким чином знайти кут, на який повинно обертатись тіло кожне оновлення фізичного рушія (рисунок 3.3).

3.3 Виявлення колізій

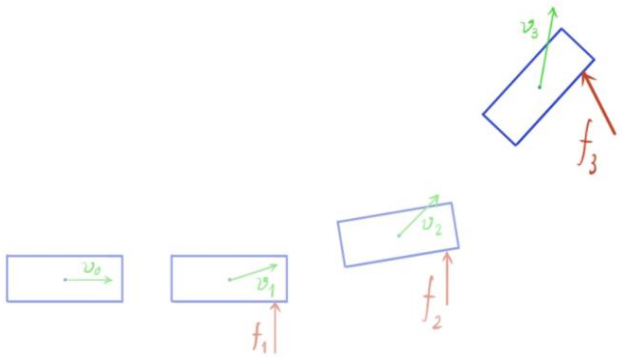


Рисунок 3.3 – Рух фізичного тіла в просторі

Для того, щоб об’єкти у відеогрі виглядали природньо, вони повинні відштовхуватися, змінювати швидкість один одного та напрям. Тому задача рушія – кожен раз, коли тіло рухається, перевірити, чи зіткнулось воно з іншими, і в разі чого надати нову силу та швидкість об’єктам колізії.

Якщо у нас на екрані n фізичних тіл, то на перевірку зіткнень між усіма займе $O(n^2)$ часу, що порівняно досить багато. Для оптимізації цього процесу зазвичай розробники прибігають до розділу простору. Це процес відокремлення об’єктів на різні підпростори, що не перетинаються, тим самим зменшуючи кількість об’єктів, що мали б перевірятися. Цей процес називають широкою фазою [26]. За ним іде вузька фаза, де об’єкти безпосередньо перевіряються на зіткнення, та вирішується, як повинен змінитися їх стан після колізії.

3.3.1 Широка фаза

Для спрощення перевірки на перетин вводиться поняття граничного простору тіла – простої геометричної фігури, що обмежує собою тіло. Якщо граничні простори не перетинаються, то не перетинаються і самі тіла. Одним з найвідоміших способів задавати перші називають

вирівнювані за осями граничні рамки (рисунок 3.4). Для полігонів їх можна знайти простим перебором усіх вершин фігури.

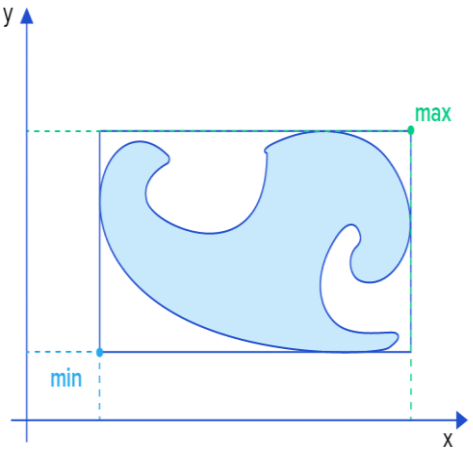


Рисунок 3.4 – Вирівняні за осями граничні рамки складної фігури

Таким чином, пошук колізії спрощується до знаходження перетину прямокутників. Проте час перебору усіх фігур на сцені цим алгоритмом не змінюється, через що розробники застосовують певні алгоритми розбиття сцени на підпростори.

а. Алгоритм сортування та перегляду. Цей метод, також відомий як «підмітати та обрізати», полягає в тому, що ми проектуємо крайні вершини прямокутників на координатні вісі, а потім, на наступній фазі, порівнюємо лиш ті, що перетинаються на проекціях (рисунок 3.5). Усі переглянуті вершини вносяться у список, який при кожному оновленні можна змінювати та сортувати. Зазвичай використовують метод сортування вставками, адже він ліпше за все підходить для впорядкування майже відсортованих елементів.

б. Дерева динамічних граничних просторів. Цей метод, також відомий як ієрархія динамічних просторів, базується на створенні дерев, лист якого являє собі тіло на сцені. Він є вдосконаленням попереднього алгоритму, адже спочатку, як

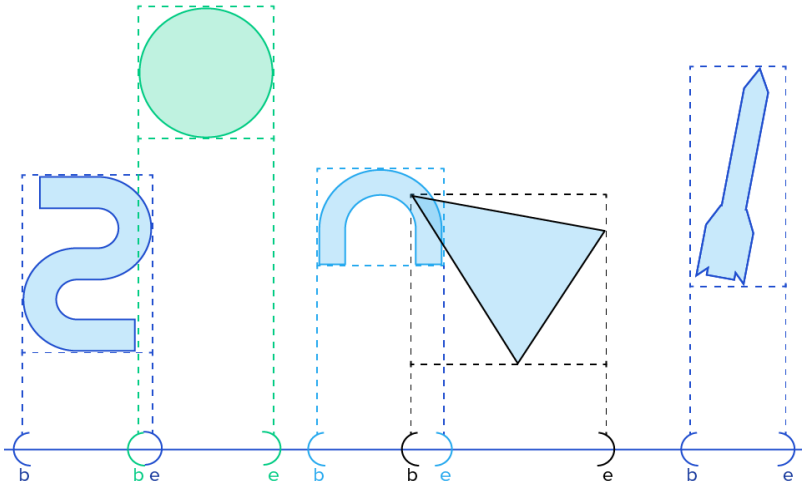


Рисунок 3.5 – Алгоритм сортування та перегляду по осі абсцис

і метод сортування та перегляду, розподіляє вершини фігур на координатах, а потім об’єднує їх у більші граничні простори, який назначається їх батьком. Далі він об’єднує і батьків, і так поки вони не об’єднаються у дерево ієрархії підпросторів (рисунок 3.6).

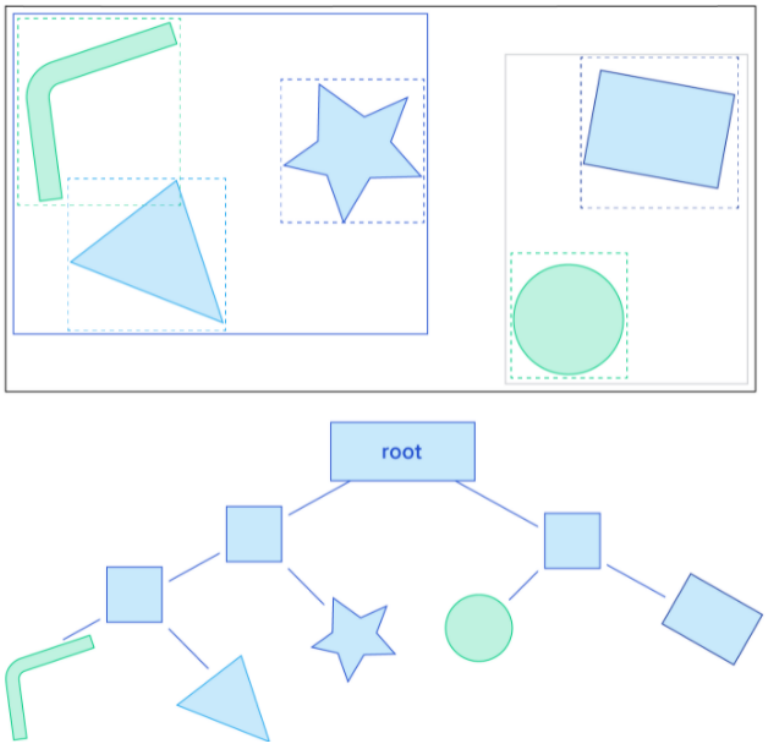


Рисунок 3.6 – Дерево динамічних граничних просторів

3.3.2 Вузька фаза

Після попередньої фази ми маємо деякі списки об’єктів, що можуть потенційно перетинатися. Для виявлення колізії об’єктів існує декілька алгоритмів, проте більшість з них працює лише з опуклими фігурами, тому в симуляціях увігнуті тіла ще при створенні розбивають на декілька простих опуклих. Іншим варіантом є спрощення шляхом наближення увігнутого тіла до опуклого, наприклад, алгоритмом пошуку швидкої оболонки, складність якого дорівнює $O(n \log n)$, де n це кількість вершин фігури (рисунок 3.7).

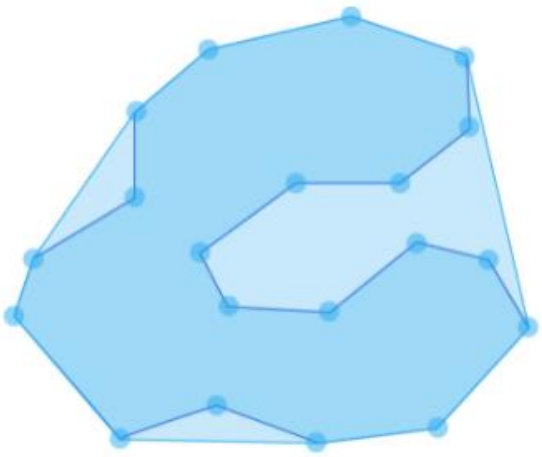


Рисунок 3.7 – Результат роботи алгоритму пошуку швидкої оболонки

Для пошуку перетину між опуклими тілами зручно застосовувати теорему про вісь, що розділяє. Вона каже про те, що дві фігури перетинаються тоді і лише тоді, коли немає жодної такої прямої, ортогональна проекція крайніх вершин фігур на яку перетиналися б (рисунок 3.8). Таких прямих може бути нескінченна кількість, проте для перевірки полігонів достатньо зробити порівняння для кожного його ребра: чи лежать вершини іншої фігури перед ним. Для цього використовується формула:

$$(v - a) * n > 0,$$

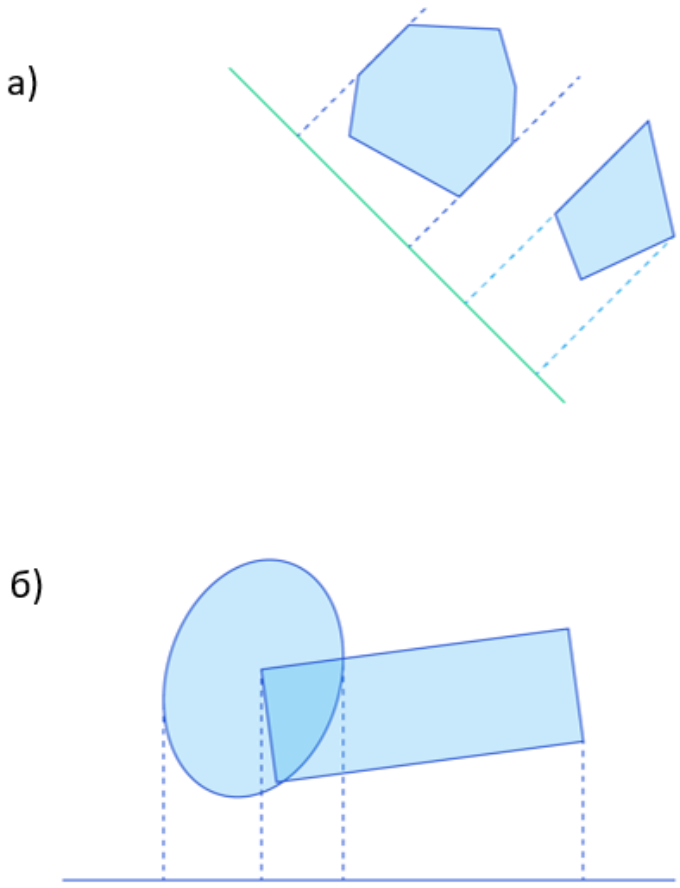


Рисунок 3.8 – Знаходження колізії за теоремою про вісь, що розділяє:
а) не перетинаються б) перетинаються

де a це точка на ребрі, n – вектор нормалі від ребра до іншої фігури, та v – вершина, яку ми перевіряємо. Для збільшення продуктивності останнє обране ребро зазвичай зберігають у пам’яті, адже при невеликих обертаннях фігури воно не буде змінюватися.

3.3.3 Проблемні колізії

Описані методи виявлення зіткнень працюють лише для тіл, що мають невелику швидкість. При великому ж прискоренні об’єктів може виникнути ситуація, коли одне тіло за ітерацію проходить крізь інше. Таку ситуацію називають довгою колізією або проблемою швидкої кулі, та частіше за все вирішують спеціальними методами:

а) метод спільного граничного простору – спосіб вирішення, при якому будується найменша фігура, що включає в себе тіло у поточному стані та попередньому, далі йде процес обробки зіткнення вже між новими фігурами (рисунок 3.9 а);

б) метод маленьких кроків – спосіб вирішення проблеми, при якому для тіл додатково обраховується колізія між їхніми проміжними станами, він надає більш точний результат, аніж попередній, проте потребує більше часу на виконання (рисунок 3.9 б).

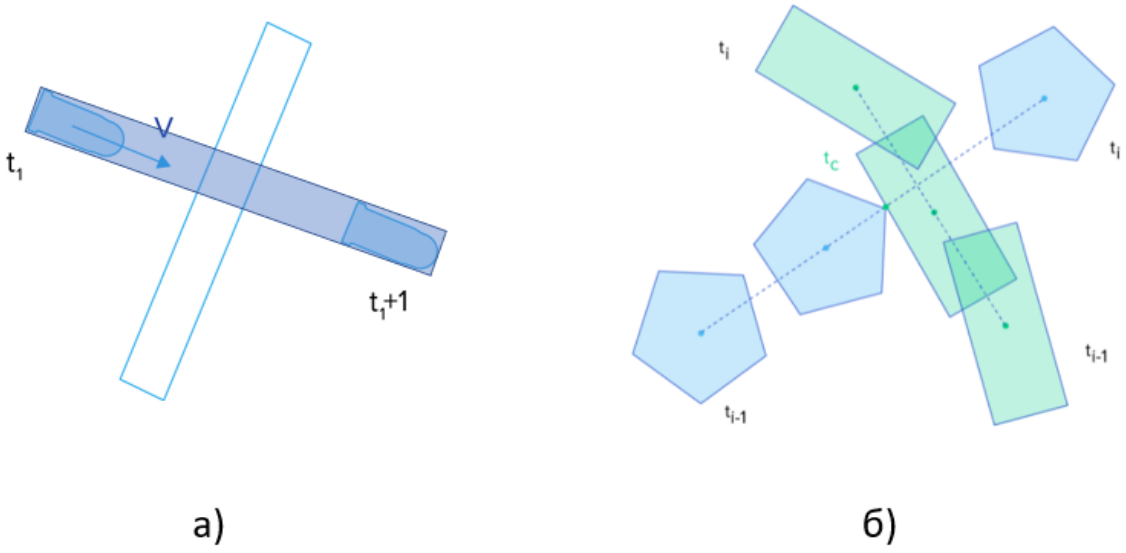


Рисунок 3.9 – Способи вирішення проблеми швидкої кулі:
а) метод спільного граничного простору б) метод маленьких кроків

3.4 Вирішення колізій

Після виявлення зіткнень задача фізичного рушія відокремити тіла у разі їх перетину та надати їм нових сил та швидкостей. Уявімо два тіла A та B, що зіткнені. Нехай $p_a, p_b, \propto_a, \propto_b$ та r_a, r_b їхні позиції, кути та вектори від центроїду до точки зіткнення відповідно. Позначимо як n вектор

нормалі від точки зіткнення, до фігури B (рисунок 3.10). Введемо $R(\theta)$ – матрицю оберту на кут θ, що обраховується за формулою:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

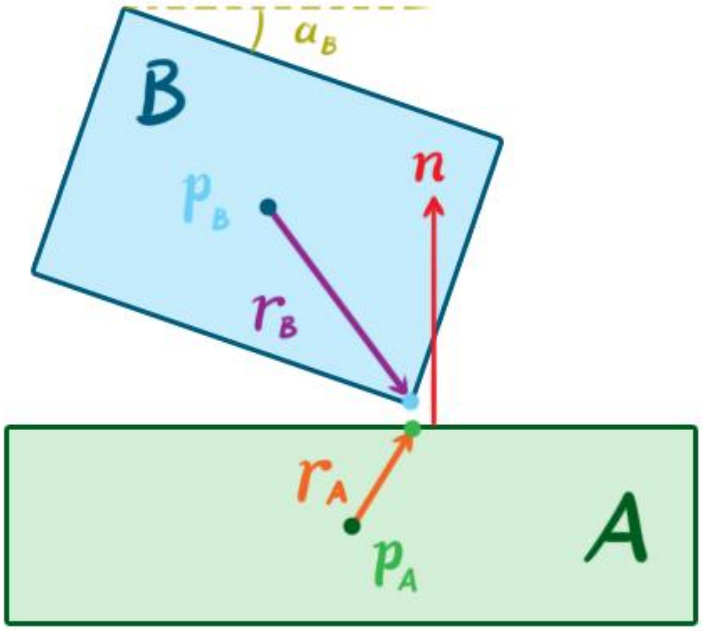


Рисунок 3.10

Використовуючи ці дані, ми можемо знайти довжину зсуву B відносно точки зіткнення:

$$C = (p_b + R(\propto_b) r_b - p_a - R(\propto_a) r_a)\, n.$$

Надалі обраховуються нові швидкості та прискорення тіл. Введімо для використання поняття вектору стану:

$$q = \begin{bmatrix} p \\ \propto \end{bmatrix},$$

де p та $\propto$ відповідно позиція та кут тіла, а за M позначимо матрицю мас:

$$M = \begin{bmatrix} m & & \\ & m & \\ & & I \end{bmatrix},$$

де m це маса тіла, а I – його момент інерції. Таким чином, зміну стану об’єкта можна описати за формулою:

$$v_2 = v_1 + \Delta t M^{-1} F$$

$$q_2 = q_1 + \Delta t v_2$$

Оновлення фізики – важкий процес з багатьма обрахунками, тому для збільшення продуктивності рушію вводять поняття островів та сплячих тіл:

а) островами називають групу об’єктів, яка не впливає на тіла, які не входять до неї, а сила, що буде прикладена до одного об’єкту, розповсюджуватиметься на всю групу, що дає змогу спростити обчислення (рисунок 3.11 а);

б) сплячим тілом називають об’єкт, позиція якого залишається незмінною впродовж симуляції, що дає змогу не обраховувати його фізичний стан, поки на нього не вплинуть ззовні (рисунок 3.11 б).

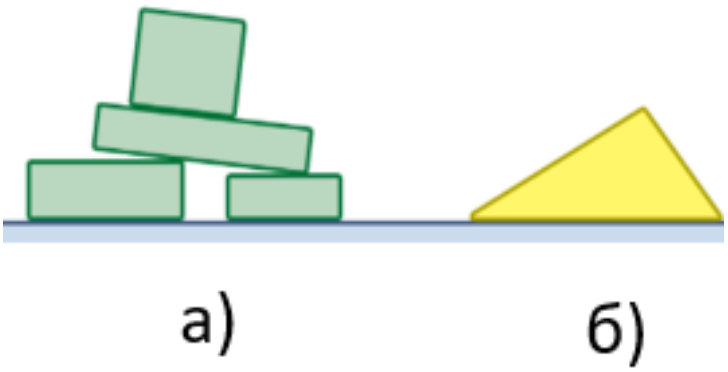


Рисунок 3.11 Спрощені фізичні об’єкти:
а) острів б) спляче тіло

3.5 Висновки розділу

У цьому розділі були розглянуті фізичний рушій, задачі, що він вирішує, та способи їх розв’язку. Вивчені процеси руху твердих тіл, алгоритми пошуку колізій. Розглянуті види зіткнень та варіанти їх вирішення. Досліджені процеси спрощення симуляції та оптимізації. Розглянуті складні колізії та проблеми, що можуть виникати при їх розвитку. Описані способи їх уникнення.

РОЗДІЛ 4. РОЗРОБКА ІГРОВОГО РУШІЯ

4.1 Проектування та розробка

Багато сучасних ігрових рушіїв використовують систему компонентів та сутностей як основний шаблон проектування рушія через її зручність та масштабованість, тому її було вирішено використовувати при розробці [27]. Шаблон вводить поняття систем, компонентів і сутності та описує взаємодію між ними (рисунок 4.1). Компонентами називаються відокремлені об’єкти, що містять відокремлені дані. Вони, у свою чергу, оброблюються системами, кожна з яких має унікальну функцію та область впливу. Сутністю називається контейнер, який об’єднує в єдиний об’єкт декілька компонентів.

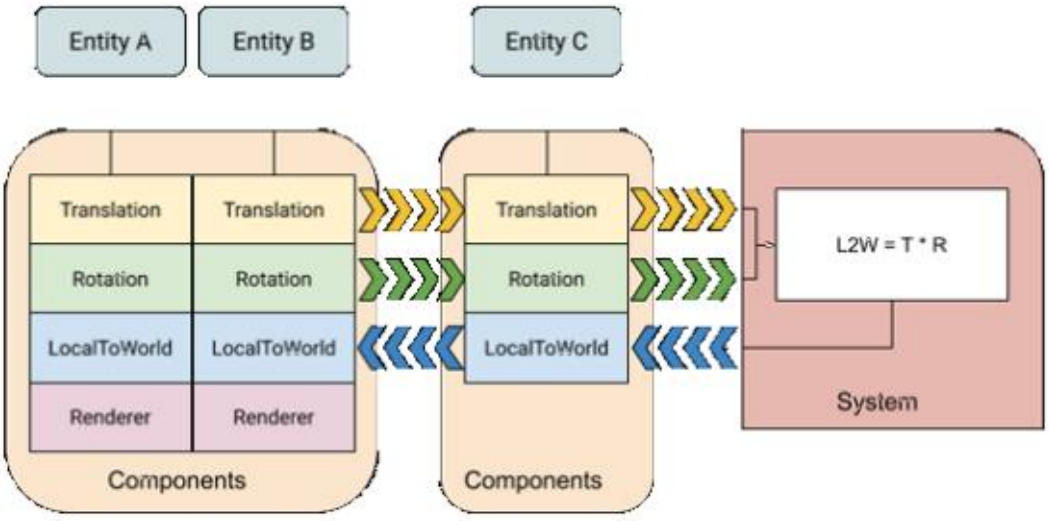


Рисунок 4.1 Приклад взаємодії між сутностями, системами та компонентами

Для розбиття на структуровані частини та зручного керування між ними було використано шаблон одинак, мета якого забезпечити глобальну

точку доступу до конкретного методу чи ресурсу завдяки єдиному екземпляру відповідного класу.

Для спрощеного керування графікою було обрано портативну багатоплатформну безкоштовно поширювану бібліотеку Simple and Fast Multimedia Library (SFML). Вона забезпечує використання двомірної графіки з апаратним прискоренням на базі OpenGL, має розширення для управління пристроями вводу та виводу, а також аудіо та мережеві модулі.

Для зручного тестування була підключена бібліотека Immediate Mode GUI (рисунок 4.2), що дозволяє змінювати деякі параметри відеогри не перериваючи симуляцію.

Рисунок 4.2 Скріншот параметрів стандартного Immediate Mode GUI

У фізичному рушії було відтворено симуляцію багатокутних твердих об’єктів та кругів, враховуючи їхні сили, швидкості на моменти обертання. Була використана оптимізація алгоритмом сортування та перегляду, а також введені поняття сплячих тіл. Додана симуляція дії сили тяжіння. Інтегрована можливість керувати фізичними тілами. Скріншот робочої програми наведений на рисунку 4.3.

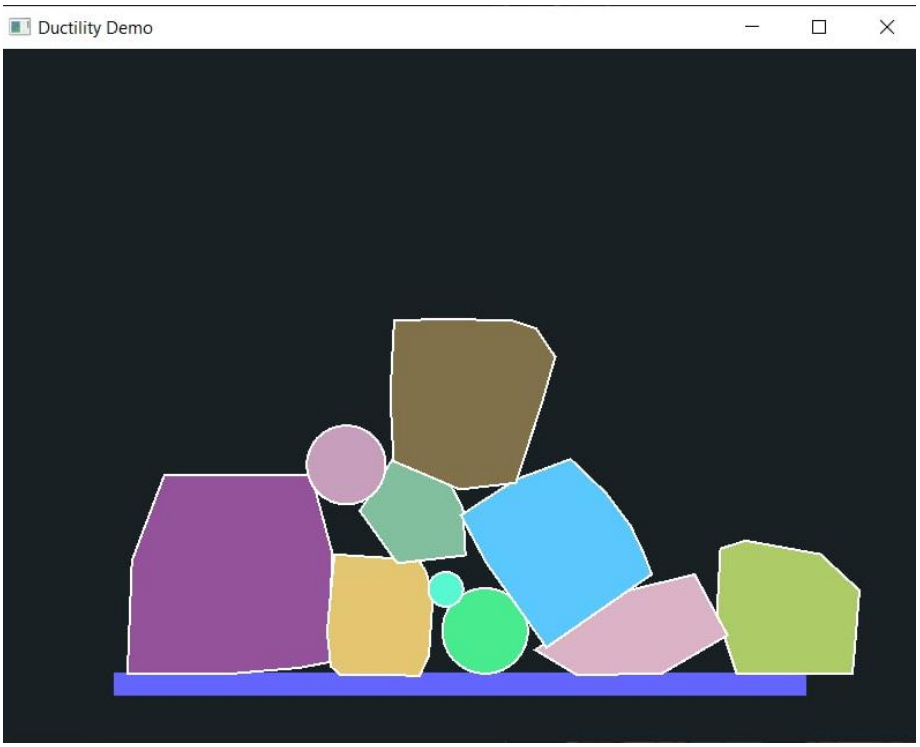


Рисунок 4.3 Фізична симуляція в розробленому додатку

4.2 Подальший розвиток

Надалі планується додати підтримку більшої кількості геометричних фігур, інтегрувати можливість керувати одразу декількома фізичними тілами. Додати нетверді фізичні об’єкти, такі як рідини та гази. Інтегрувати підтримку систем фізичних об’єктів. Додати можливість налаштовувати силу тертя між об’єктами.

У подальшому для більшої оптимізації планується виведення обрахунку фізики часток до шейдерів відеокарти, що дасть змогу зменшити навантаження на процес та потенційно збільшити кількість одночасно підтримуваних фізичних об’єктів.

ВИСНОВКИ

а. У результаті роботи було розглянуто поняття гри, досліджено, що робить ігри цікавими та чому люди в них грають вже протягом століть.

б. Розглянуто поняття відеогри, їхній розвиток та становлення відео ігрової індустрії. Досліджені жанри комп’ютерних ігор, їхні ключові відмінності.

в. Дано означення ігровому рушію, розглянуті передумови його появи. Вивчені види ігрових рушіїв та досліджена статистика популярності додатків за цими видами. Розглянуті існуючі ігрові рушії, порівняно їхні переваги та недоліки.

г. Детально переглянуті усі складові рушія, їхні задачі, функції. Наведені процеси їх виконання, області використання та розглянуті приклади їх взаємодії.

д. Дані означення поняттям фізичного рушія, тіл, колізії. Розглянуті етапи обробки об’єктів фізичним рушієм. Вивчені процеси руху твердих тіл, алгоритми пошуку колізій. Розглянуті види зіткнень та варіанти їх вирішення. Досліджені процеси спрощення симуляції та оптимізації. Розглянуті складні колізії та проблеми, що можуть виникати при їх розвитку. Описані способи їх уникнення.

е. Отримані теоретичні знання використані для розробки власного фізичного ігрового рушія. Реалізовано взаємодію простих фізичних об’єктів, підтримку сили тяжіння, керування фізичним тілом.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Шрейєр Д. Кров, пот і пікселі / Джейсон Шрейєр., 2019. – 368 с. – (Бомбора). – (Легендарні комп'ютерні ігри).
2. Video Game Statistics, Click the “Start” Button [Електронний ресурс] // Review42. – 2021. – Режим доступу до ресурсу: https://review42.com/.
3. Video game industry - Statistics & Facts [Електронний ресурс] // Statista. – 2021. – Режим доступу до ресурсу: https://www.statista.com/.
4. Bycer J. Game Design Deep Dive / Joshua Bycer., 2019. – 152 с. – (CRC Press). – (Programming / Games).
5. Doom [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу https://www.idsoftware.com/.
6. Unity [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://unity.com/.
7. Unreal Engine [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.unrealengine.com/.
8. Koster R. Theory of Fun for Game Design / Raph Koster., 2013. – 300 с. – (O'Reilly Media). – (Game Theory (Books)).
9. Що таке відеогра [Електронний ресурс] // DTF. – 2021. – Режим доступу до ресурсу: https://dtf.ru/.
10. Jason G. Game Engine Architecture / Gregory Jason., 2018. – 1240 с. – (A. K. Peters/CRC Press). – (Third Edition).
11. Game Engines and Game History [Електронний ресурс] // KINEPHANOS. – 2014. – Режим доступу до ресурсу: https://www.kinephanos.ca/.
12. The Quake-C Language [Електронний ресурс] // PGP key. – 1997. – Режим доступу до ресурсу: https://www.gamers.org/.

13. WHAT GENRES ARE PLAYERS LOOKING FOR ON STEAM? [Електронний ресурс] // HOW TO MARKET A GAME. – 2020. – Режим доступу до ресурсу: https://howtomarketagame.com/.
14. CryEngine [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.cryengine.com/.
15. UbiArt Framework [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.ubiart.com/.
16. Nystrom N. Game Programming Patterns / Nystrom Nystrom., 2014. – 345 с. – (Genever Benning). – (Computer games).
17. NVIDIA [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.nvidia.com/.
18. Havok [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.havok.com/.
19. NVIDIA PHYSX SYSTEM SOFTWARE [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.physx.com/.
20. SFML [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.sfml-dev.org/.
21. Hello Triangle [Електронний ресурс] // Learn OpenGL. – 2014. – Режим доступу до ресурсу: https://learnopengl.com/.
22. OpenFlipper [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: https://www.graphics.rwth-aachen.de/.
23. Sardina S. A BDI agent programming language with failure handling, declarative goals, and planning / S. Sardina, L. Padgham. – Melbourne, VIC, Australia: School of Computer Science and IT, RMIT University, 2010. – 53 с. – (Springer). – (BDI agent-oriented programming).
24. Video Game Physics Tutorial [Електронний ресурс] // Developers. – 2015. – Режим доступу до ресурсу: https://www.toptal.com/.

25. Physically Based Modeling: Principles and Practice [Електронний ресурс] // SIGGRAPH. – 1997. – Режим доступу до ресурсу: http://www.cs.cmu.edu/.
26. realtimecollisiondetection.net [Електронний ресурс]:[Веб-сайт] – Режим доступу до ресурсу: http://realtimecollisiondetection.net/.
27. Unity User Manual [Електронний ресурс] // Unity Technologies. – 2020. – Режим доступу до ресурсу: https://docs.unity3d.com/.

ДОДАТОК А

Діаграма структур ігрового рушія