НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2016р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.050102, 8.050102 Інформаційні технології проектування
7.050103, 8.050103 Системне проектування
(код та назва спеціальності)

на тему: Методи та інструментальні засоби побудови ігор (віртуальна реальність, мультимедія)

Виконав: студент 4 курсу, групи ДА-21
(шифр групи)

_____ Авраменоко Віталій Андрійович_____ _____
(прізвище, ім'я, по батькові) (підпис)

Керівник _____к.т.н., доц. Цурін О.П. _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний проф. док. ек. н. Семенченко Н. В.___ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль _____ст. викладач Бритов О.А._____ _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.050102, 8.050102 Інформаційні технології проектування 7.050103, 8.050103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

«___»_____2016 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Авраменко Віталію Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)) Методи та інструментальні засоби побудови ігор (віртуальна реальність, мультимедія)

керівник проекту (роботи)) Цурін О.П., к.т.н., доц.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 15.06.2016

3. Вихідні дані до проекту (роботи)
   1. Порівняння сучасних популярних інструментальних засобів: XNA, MonoGAMe, Phaser, Construct 2, pixi.js, Blender Game Engine, Easel.JS, Turbulenz, melonJS, PandaJS, Game Maker, Unity, Unreal Engine.
   2. Виділення груп та визначення оптимальних представників.
   3. Детальний аналіз представника кожної групи через написання прототипу гри.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Постановка задачі та аналіз предметної області
2. Аналіз інструментальних засобів.
3. Розробка проектів прототипи для детального аналізу можливостей характерних представників класів.
4. Функціонально-вартісний аналіз програмного продукту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Презентація MS PP.
2. Порівняння доступності інструментальних засобів - плакат
3. Гістограми популярності інструментальних засобів – плакат
4. Порівнянні функціональних можливостей інст. засобів - плакат.

6. Консультанти розділів проекту (роботи)*

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

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Аналіз та класифікація інструментальних засобів	28.02.2016	
4	Постановка задач для прототипів	10.03.2016	
5	Розробка прототипів	15.03.2016	
6	Аналіз отриманих результатів	25.03.2016	
7	Оформлення дипломної роботи	31.05.2016	
8	Отримання допуску до захисту та подача роботи в ДЕК		

Студент _____ (підпис) В.А. Авраменко (ініціали, прізвище)

Керівник проекту (роботи) _____ (підпис) О.П. Цурін (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

Анотація

бакалаврської дипломної роботи Авраменка Віталія Андрійовича

на тему «Методи та інструментальні засоби побудови ігор (мультимедія, віртуальна реальність)»

У даній роботі ставиться завдання розглянути сучасні інструментальні засоби для розробки мультимедійних ігор. Було проаналізовано найпопулярніші сучасні ігрові рушії, фреймворки та конструктори ігор.

Популярність комп'ютерних ігор зростає з кожним роком, ніша ринки дозвілля постійно потребує поповнень новими ігровими тайтлами. Ігрова індустрія постійно розробляє нові інструментальні засоби для полегшення та пришвидшення процесу розробки ігор. Велика різноманітність, конкурентоздатність, специфічність використання, зміна актуальності використаних технологій роблять задачу вибору засобів розробки гри неоднозначною. Отже аналіз сучасних засобів та окреслення сфер їх використання є актуальною задачею, що поліпшить якість та швидкість розробки ігрових проектів.

Результат роботи – порівняльний аналіз та класифікація сучасних популярних інструментальних засобів. Аналіз розроблених з допомогою характерних представників різних класів прототипів-ігор. Окреслення сфери використання, та цільової аудиторії для характерних представників інструментальних засобів.

Загальний обсяг роботи 76 сторінки, 22 рисунків, 8 таблиці, 16 бібліографічних найменувань.

Ключові слова: Ігровий рушій, фреймворк, конструктор ігор, ігровий тайтл, фізичний рушій, коллізія

АННОТАЦИЯ

бакалаврской дипломной работы Авраменко Виталия Андреевича

на тему «Методы и инструментальные средства построения игр (мультимедиа, виртуальная реальность)»

В данной работе ставится задача рассмотреть современные инструментальные средства для разработки мультимедийных игр. Были проанализированы самые популярные современные игровые движки, фреймворки и конструкторы игр.

Популярность компьютерных игр растет с каждым годом, ниша рынки досуга постоянно нуждается пополнений новыми игровыми тайтла. Игровая индустрия постоянно разрабатывает новые инструментальные средства для облегчения и ускорения процесса разработки игр. Большое разнообразие, конкурентоспособность, специфичность использования, изменение актуальности используемых технологий делают задачу выбора средств разработки игры неоднозначной. Итак анализ современных средств и определение сфер их использования является актуальной задачей, улучшит качество и скорость разработки игровых проектов.

Результат работы - сравнительный анализ и классификация современных популярных инструментальных средств. Анализ разработанных с помощью характерных представителей разных классов прототипов-игр. Определение сферы использования, и целевой аудитории для характерных представителей инструментальных средств.

Общий объем работы 76 страницы, 22 рисунков, 8 таблицы, 16 библиографических наименований.

Ключевые слова: Игровой движок, фреймворк, конструктор игр, игровой тайтл, физический движок, коллизия

ANNOTATION

on Vitaly Avramenko bachelor's degree thesis: "Methods and tools for game development (multimedia, virtual reality)"

In this work we consider the task to examine modern tools to develop multimedia games. It analyzes the current most popular game engines, game designers and frameworks.

Popularity of gaming increases every year, niche markets entertainment constant need of replenishment new game titles. Game industry is constantly developing new tools to facilitate and accelerate the process of game development. Great variety, competitiveness and specificity of use, change the relevance of technologies used make the task of selecting the game development tools ambiguous. So the analysis of modern and defining areas of use is an urgent task that will improve the quality and speed of game development projects.

Result of work - a comparative analysis and classification of modern popular tools. The analysis developed using characteristic of different classes of prototype games. The delineation of the scope of use and target audience for specific representatives tools.

The total amount of work 76 pages, 22 figures, 8 tables, 16 bibliographic titles.

Keywords: Game engine, framework, designer games, game titles, physics engine, collision

ЗМІСТ

ВСТУП

В наш час неможливо уявити повсякденне житя без постійного притоку інформації. Для задоволення цих потреб використовуються сучасні засоби мультимедіа : телебачення, Інтернет, тощо.

Мультиме́діа — комбінування різних форм представлення інформації на одному носієві. Мультимедіа може бути грубо класифікована як лінійна й нелінійна.

Аналогом лінійного способу подання може бути кіно. Людина, що переглядає даний документ жодним чином не може вплинути на його зміст. Нелінійний спосіб подання інформації дозволяє людині брати участь у поданні інформації, взаємодіючи якимось чином із засобом відображення мультимедійних даних. Участь людини в даному процесі також називається «інтерактивністю». Такий спосіб взаємодії людини й комп'ютера найбільш повно представлений у категоріях комп'ютерних відео ігор

Відеогра́ — це електронна гра, в ігровому процесі якої гравець використовує інтерфейс користувача, щоб отримати зворотну інформацію з відеопристрою. Електронні пристрої, які використовуються для того щоб грати, називаються ігровими платформами. Наприклад, до таких платформ належать персональний комп'ютер та гральна консоль. Пристрій введення, який використовується для керування грою, називається ігровим контролером. Це може бути, наприклад, джойстик, клавіатура та мишка, геймпад або сенсорний екран.

Компютерні відеоігри ігри набули небувалої популярності за останні роки та посіли почесне місце на ринку розваг та дозвілля. Як результат, бурний розвиток індустрії розробки комп’ютерних ігор.

Для задоволення потреб розробників було створенно величезну кількості інструментальних засобів та продовжується розробка нових. Загальна ціль всіх інструментальних засобів - полегшення та покращення розробки, використання

пердовіх технологій обробки графіки, фізики, забезпечення кросплатформенності розробених проектів.

Метою даної роботи є збір теоретичних відомостей про сучасні інструментальні засоби для побудови ігор. Огляд та аналіз популярних представників, розбиття засобів на класи. Постановка задач для тесових проектів ігор-прототипів, у розрізі спрощених вимоги до сучасних ігрових проектів. Детальний аналіз найбільш характерних представників кожного класу, розробка з їх допомогою ігор прототипів. Визначення оптимальних сфер використання для інструментальних засобів спираючись на аналіз процесу розробки прототипів.

1. ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мультимедійні ігри

В сфері мультимедіа не останнє місце займає таке явище, як мультимедійні комп'ютерні ігри. Мультимедійні ігри — такі ігри, у яких гравець взаємодіє з віртуальним середовищем, побудованим комп'ютером. Стан віртуального середовища передається гравцеві за допомогою різних способів передачі інформації (аудіальний, візуальний, тактильний). Наразі всі комп'ютерні ігри відносяться до мультімедийних ігор. В такий тип ігор можна грати як в поодинці на локальному комп'ютері або приставці, так і з іншими гравцями через локальну або глобальну мережу.

У 2011 році відеоігри були офіційно визнані видом мистецтва урядом США та Національним фондом мистецтв США[13].

1.2 Ігрова індустрія

Перша комп’ютерна гра «Зоряні війни» вийшла у світ 1962 року. Її завдання полягало в тому, щоб відбити астероїди і напади ворожих космічних кораблів. Згодом було створено багато інших ігор. А з поширенням у 1970–1980 роках потужніших комп’ютерів електронних ігор побільшало: пригодницькі ігри, ігри-головоломки, стратегічні ігри та ігри «екшн». Багато ігор імітують різні види спорту, як от хокей на льоду чи гольф. Чимало з них здобули високу оцінку громадськості,оскільки вони дуже цікаві й допомагають у навчанні. Стає дедалі популярнішим онлайновий вид комп’ютерної гри. Її персонажами керує не комп’ютер, а гравці, як і через Інтернет одночасно беруть участь у грі. Їх можуть бути тисячі. Популярність таких забав пояснюється можливістю поспілкуватися з іншими. Гравці “розмовляють” одні з одними і відчувають себе частиною всесвітньої родини.

В Україні щороку зростає кількість людей, що купують комп’ютерні ігри. Якщо для гравців це просто забавка, то для розробників, виробників та

розповсюджувачів — досить вигідний бізнес. Світовий ринок комп’ютерних ігор оцінюють в сотні мільярдів доларів. На Заході один ліцензійний ігровий диск коштує $40–50. Популярну гру можуть продати накладом від мільйона до кількох десятків мільйонів примірників. Не дивно, що в розробку гри там можуть легко вкласти кілька мільйонів доларів. Ця індустрія приносить великі прибутки і державній скарбниці.Комп’ютерні ігри стали вже й елементом політики. Парламенти західних країн дискутують щодо законодавчого обмеження насильства в комп’ютерних іграх або ж стимулювання виробництва ігор як такого.

1.3 Класифікація комп’ютерних ігор

Комп'ютерні ігри в основному класифікуються за жанрами, а також за кількістю гравців.

Внаслідок того, що критерії приналежності гри до того чи іншого жанру не визначені однозначно, класифікація ігор недостатньо систематизована, і в різних джерелах дані про жанр конкретного проекту можуть розрізнятися. Проте, існує консенсус, до якого прийшли розробники ігор, і приналежність гри до одного з основних жанрів майже завжди можна визначити однозначно. Ці найбільш популярні жанри (які об'єднують в собі безліч піджанрів) перераховані нижче.

Існують ігри з елементами кількох жанрів, які можуть належати кожному з них (наприклад, серія Grand Theft Auto, Космічні Рейнджери, Rome: Total War і багато інших). Такі проекти зараховують або до одного з жанрів, який в грі є основним, або відразу до всіх, присутніх в грі, якщо вони в рівній мірі становлять геймплей проекту.

В основі сучасних розподілів відеоігор на жанри лежить вид активності, який найчастіше здійснює гравець в іграх даного жанру. Так відеоігри в загальному можуть поділятися на ігри руху, планування і сюжету або спілкування, дії та контролю. В багатьох класифікаціях визначення жанру відбувається за кількома осями. Наприклад, за двома осями сюжет — свобода

дії, або трьома абстракція — симуляція — свобода[16]. Проте найчастіше використовуваною класифікацією, хоч і не прийнятою усіма, жанри з якої зустрічаються в більшості існуючих, є наведена нижче, яка виключає осі або багаторівневі поділи:

Action

В іграх такого жанру необхідно використовувати рефлекси та швидкість реакції для подолання ігрових обставин. Це один із базових жанрів і водночас найпоширеніший. Як правило екшн-ігри пов'язані із агресивними діями щодо противників і/або оточення. Персонаж гравця повинен битися, стріляти, переслідувати ціль чи самому уникати переслідування. Стосовно екшн-ігор, де наявні значні елементи пригодницьких ігор, застосовується термін Action-adventure. З-поміж них часто виділяється напрямаркадних ігор, ігровий процес яких вирізняється простотою та легкістю освоєння.

Цей жанр поділяється на велику кількість піджанрів, серед яких основними є:

Шутери — вимагають від гравця боротися з противниками шляхом стрілянини. Залежно від перспективи, поділяються на шутери від першої (Wolfenstein 3D) читретьої особи (Макс Пейн). Існують різновиди як тактичні, в яких ігровий персонаж діє у складі команди (SWAT 4), аркади (Alien Shooter), стелс-екшн, метою якого є приховані дії для виконання завдань, без прямого знищення противників (серія Hitman). Щодо ігор, де основою ігрового процесу є знищення великих кількостей ворогів, а сама стрілянина в цьому переважає над тактикою і влучністю, застосовується термін Shoot 'em up (R-Type, Touhou, Contra).

Файтинги — імітують ближній бій, як правило один на один, на спеціальних аренах. Приклади: серія Mortal Kombat, Street Fighter.

Beat 'em up — подібні на файтинги, з тою різницею, що персонажі вільно переміщуються ігровим світом (а не спеціальною ареною) і борються проти багатьох противників одночасно. Приклади: Golden Axe, Battletoads, Double Dragon.

Платформери — персонаж мусить рухатися, стрибаючи по платформах, та долати перешкоди. Приклади: Mario, Spyro the Dragon, Megaman.

Лабіринти — персонаж рухається лабіринтом з метою знайти вихід, зібрати предмети і/або уникнути пасток і небезпек. Часто в іграх цього жанру є обмеження на час. Приклади: Pac-Man, Boulder Dash.

Стратегія

Сенс стратегічних ігор полягає в плануванні дій та виробленні певної стратегії для досягнення якоїсь конкретної мети, наприклад, перемоги у військовій операції. Гравець керує не одним персонажем, а цілим підрозділом, підприємством чи навіть всесвітом. Відповідно до реалізації ігрового часу, стратегічні відеоігри поділяються на два основних різновиди:

Покрокові стратегічні ігри — гравець та його противник здійснюють дії один за одним, покроково, маючи змогу за один ігровий хід виконати певну кількість операцій. Приклади: Heroes of Might and Magic III, Цивілізація.

Стратегічні ігри в реальному часу — і гравець і противник виконують свої дії одночасно, проте часто часто масштаб часу відрізняється від реального. Наприклад, будівництво триває кілька секунд, а ігрова година складає кілька хвилин реального часу. Приклади: StarCraft, Age of Empires III.

Tower Defense — похідний жанр, в якому гравець керує оборонними баштами аби не пропустити хвилі ворогів.

MOBA — похідний жанр, в якому гравець керує одним персонажем з метою захистити свою базу від ворогів та знищити ворожу.

MMORTS — багатокористувацькі он-лайн стратегії в реальному часі, орієнтовані на суперництво/співпрацю з іншими реальними гравцями, щоб досягнути спільної мети.

Видом стратегічних ігор, які відображають суто бойові дії, є варгейми. Ігри, в яких гравець управляє масштабними державами (імперіями, планетами, галактиками), при цьому займаючись всіма аспектами їх життя, включаючи торгівлю, науки та війни, мають назву глобальних стратегій. Близьким до них

є ігри в бога, де гравець виступає в ролі надприродньої істоти, управляє світом через чудеса, правителів, керує силами природи.

Рольова гра

Гравець асоціюється з конкретним персонажем або лідером команди, які діють відповідно до правил своїх ролей. Наприклад, лицар не може того, що чарівник, кожна роль має свої особливості, а часом від неї залежить і розвиток сюжету. Мета ігрового процесу полягає у виконанні різноманітних завдань (квестів) для розвитку одного персонажа або групи. Можливі варіанти дій залежать від обраного образу персонажа, попередньо визначеного чи формованого самим гравцем.

Рольові відеоігри часто поєднуються з іншими жанрами, утворюючи Action-RPG, тактичні рольові ігри, MUD-и і т.п.. Особливим випадком рольової гри є MMORPG, багатокористувацькі рольові онлайн-ігри, в яких живі гравці взаємодіють одне з одним у віртуальному світі через мережу Інтернет, що визначає специфіку ігрового процесу.

Іноді рольові відеоігри поділяються за дизайном і побудовою сюжету. Так існує умовний поділ на рольові ігри західного зразка та східного

Симулятор

В широкому розумінні всі ігри є симуляторами. У вужчому значенні це відеоігри, призначені для складання уявення про дійсність за допомогою відображення певних реальних явищ та властивостей у віртуальному середовищі. Існує чимало піджанрів, як технічні (управління складними технічними пристроями, авіаційною технікою та інші, наприклад гра Іл-2 Штурмовик), аркадні (відрізняються від аркад наявністю спрощеної фізичної моделі. Наприклад, X-Wing), спортивні, економічні, побачень та інші.

Пригоди

В пригодницьких відеоіграх гравець керує ігровим персонажем, який рухається по сюжету та виконує зумовлені сценарієм завдання, покладаючись на свою уважність та логіку, здійснює пошуки підказок і вирішує загадки. Всередині жанру виділяються основні піджанри: інтерактивна

література, інтерактивні фільми та візуальні романи. Часто за аналогією до пригодницьких фільмів пригодницькими називаються ті відеоігри, сюжет яких динамічно розгортається, насичений яскравими подіями, швидкою зміною обстановки, а персонажі проявляють кмітливість та сміливість. Приклади: серія про Індіану Джонса, Disney's Aladdin in Nasira's Revenge, Syberia.

Інші різновиди

Настільна гра — електронні реалізації настільних ігор. Серед них існують як реалізації класичних ігор (шахи, преферанс), так і специфічних як Magic: The Gathering.

Головоломки — вимагають від гравця вирішення логічних завдань, передбачення можливих ситуацій. Приклади: Тетріс, Angry Birds.

Games with a Purpose — програми, за допомогою яких люди використовують свої знання для вирішення проблем (передусім у наукових дослідженнях), при цьому граючись. Такі ігри є різновидом людино-орієнтованого методу комп'ютерного моделювання. Приклади: Дарвін, C Robots[8].

1.4 Кіберспорт

Кіберспорт — спортивні змагання з відеоігор. Історія електронного спорту почалася з гри Quake, яка мала режим мережевої гри через LAN або інтернет. Завдяки популярності гри Doom, в 1997р. в США з'явилася перша ліга електронного спорту — Cyberathlete Professional League (CPL). Від цього року з'явилися багато нових ліг із кіберспорту.

Змагання з кіберспорту, у тому числі і міжнародні, проводяться по всьому світі. Найзначнішим з них є турнір World Cyber Games, організований подібно до Олімпійських ігор. WCG були започатковані в 2000 році в Південній Кореї, і з того часу проводиться щороку, у тому числі і в інших країнах. У Південній Кореї кіберспорт отримав найбільший розвиток, отже там навіть існують телевізійні канали, які транслюють змагання з електронного спорту.

Великі змагання проводяться в спеціальних місцях, де публіка може спостерігати за гравцями, що сидять за комп'ютерами, а хід змагань відстежувати на великому екрані, де транслюється ігровий процес. У Південній Кореї, через велике число глядачів, подібні змагання проводять на стадіонах. Менш масштабні змагання проводяться в комп'ютерних клубах. Крім того, змагання можуть проводитися через інтернет.

Гра через інтернет володіє деякими недоліками. У різних гравців можуть бути різні затримки передачі інформації через глобальну мережу в зв'язку з її неоднорідністю. Під час гри через інтернет складно виявити шахрайство партнерів. У контрасті, під час гри через локальну мережу всі гравці присутні в одному приміщенні під наглядом організаторів змагання, тому шахраювати набагато важче. Локальна мережа зводить нанівець і проблему затримок, оскільки має достатню і однакову для всіх пропускну здатність.

На великих змаганнях призовий фонд може сягати 1 000 000 доларів або більше. Найбільший виграш за історію кіберспортивних змагань виграли команда Na`Vi, яка перемогла у фіналі чемпіонату The international з дисципліни Dota 2, отримавши 1000000$. Призові фонди спонсорують компанії, пов'язані з виробництвом комплектуючих і периферії для комп'ютерів. [16].

Для кіберспорту підходять такі жанри відеоігор, як шутери від першої особи, стратегії реального часу і спортивні симулятори як найбільш видовищні і динамічні.

Поточні популярні професійні відеоігри включають:Counter-Strike, DotA2, FIFA, Halo 2, Heroes of Newerth, League of Legends, osu!, Point Blank, Quake, Starcraft, Unreal Tournament, Warcraft, World of Tanks

1.5 Розробка комп'ютерних ігор

Розробка відеогри має низку послідовних етапів, загалом їх є три: розробка програмного (джерельного) коду, розробка контенту (малюнки, моделі, музика) та розробка ігрових механік. Їм передує проектування (пре-продакшну) — генерування геймдизайнером ідей щодо майбутньої гри, вибір жанру, тематики,

особливостей ігрового процесу, розробка сценарію та образів персонажів з оточенням. Менеджер координує дії різних людей, залучених до розробки, складає план їхньої роботи, встановлює терміни її виконання, планує витрати. Готова гра в свою чергу має пройти низку етапів, в ході яких потрапляє до гравців і підтримує інтерес до себе. Індустрія відеоігор включає у себе багато людей з різними професіями та ролями: програмістів, які відповідають за технічні можливості гри, художників, моделювальників та аніматорів, які створюють графічний контент, композиторів та звукорежисерів, які створюють звукове оформлення та музичний супровід, який нерідко видається окремим накладом. За успішне завершення роботи над проектом відповідають продюсери. Відеоігри, які розробляються незалежними розробниками чи аматорами називаються інді-іграми. Такі ігри нерідко створюються за допомогою спеціальних програм, які які істотно полегшують (або навіть ліквідують) процес розробки коду або графіки, наприклад, як RPG Maker.

1.5.1 Написаня коду

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

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

Деякі рушії розраховані на створення ігор конкретного жанру, інші — універсальні. Не всі рушії можуть забезпечити однакові внутрішноігрові

можливості та рівень графіки. Частина рушіїв дозволяють створювати ігри для різних платформ, так Unreal Development Kit підтримує розробку інтерактивних творів для PC, Xbox 360, PlayStation 3, Wii та Android.

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

1.5.2 Розробка контенту

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

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

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

Для звукового оформлення пишеться музика і відбувається озвучування персонажів. Крім того для повноцінного звукового оформлення потрібні ефекти, як кроки, звуки пострілів. Вони можуть обиратися з вільних бібліотек чи записуватися окремо. Декотрі композитори спеціалізуються на створенні

музики до ігор. Музика може виконуватися цілими професійними оркестрами, мати пісенний супровід. Діалоги персонажів часто озвучуються спеціальними акторами на студіях озвучування.

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

1.5.3 Розробка ігрової механіки

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

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

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

1.5.4 Тестування

Після завершення праці над кодом, контентом і механікою, за яких гра може функіонувати, відбувається її доопрацювання. Гра, не зібрана до кінця, але в яку можливо грати, називається альфа-версією. Вона може містити значні помилки і недоопрацювання, як відсутність певних можливостей, музики або об'єктів. Виявленням проблем займаються тестери, котрі грають в цю гру, намагаючись сповна скористатися всіма доступними можливостями в ній. На пізнішому етапі виходить бета-версія, до тестування якої можуть залучатися і потенційні покупці гри. В бета-версії відбувається подальший пошук помилок, перевірка коректності взаємодії об'єктів ігрового світу, управління. Можливі внесення змін в оформлення, зміна ігрового балансу, тощо.

1.6 Інді ігри

Незалежні ігри або інді-ігри (англ. indie games від independent — «незалежний») — відеоігри, створені незалежно від фінансової підтримки великих видавців. Часто ці ігри дешеві або безкоштовні, багато незалежних ігор мають невеликий розмір і тому поширюються через інтернет за допомогою цифрової дистрибуції або як freeware. Більшість спочатку вільних ігор також належить до цієї категорії. Такі ігри, як правило, харектарезуются новаторськими ідеями або революціонними підходами до жанрів, адже їх не обмежують політика компанії тощо.

З 1998 року щорічно проводиться Independent Games Festival в рамках конференції Game Developers Conference. З 2005 року проводиться фестиваль IndieCade. Деякі інді-ігри стали дуже успішним у фінансовому відношенні, наприклад, Braid, World of Goo і Minecraft.

Цікаве рух почався зі стартом платформи KickStarter, де спільними зусиллями збирають кошти на реаліз ацію різних проектів. Розробники зрозуміли можливості цієї платформи, коли Тім Шейфер розмістив там свою нову гру Double Fine Adventure на початку 2012-го року. Тім розраховував

зібрати 400000 $, але в підсумку отримав майже три з половиною мільйони. За ним послідували inXile entertainment з Wasteland 2 і Stainless Games з Carmageddon: Reincarnation. Початківцям розробникам на такий прийом розраховувати не варто, але невелику суму грошей на KickStarter зібрати цілком реально.

За офіційною статистикою KickStarter більш ніж на $ 100 млн було зібранно на проеки ігор, що дало зелене світло 1,476 проектам. За останні чотири роки більш ніж 633,242 розробників пообіцяли більш ніж 4500 ігрових проектів, майже 1000 з них тільки в минулому році.

1.7 Висновки до розділу

Як видно з зазначених матеріалів за останні роки відеоігри зайняли чималу нішу на ринку розваг та дозвілля. А в деяких країнах навіть досягли статусу спорту та мистецтва. Компаній з розробки ігор стає дедалі більше. До того ж зявилося таке поняття як інді-розробник та інді-студія – незалежні розробники що власноруч, без підтримки великих компаній, створюють та продають свої ігри. Як наслідок все більше людей починають захоплюватись розобкою відеоігр. В свою чергу зявився чималий попит на інструментальні засоби що полегшують розробку, покращують графіку, оптимізують фізику тощо. Проте різні засоби мають спецефічні сфери використання залежно від типу, жанру, платформи тощо. А отже потреба в їх аналізі та класифікації виглядає досить актуальною, завдяки моїм дослідженням розробник зможе зберігти час та більш доцільно обрати потрібні йому інструментальні засоби.

2. АНАЛІЗ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

2.1 Дослідження існуючих інструментальних засобів

Більшість інструментальних засобів для розробки ігор можна поділити на 3 группи:

1. Фреймворк - програмна платформа, яка визначає структуру програмної системи; програмне забезпечення, що полегшує розробку і об'єднання різних компонентів великого програмного проекту. Фреймворк відрізняється від поняття бібліотеки тим, що бібліотека може бути використана в програмному продукті просто як набір підпрограм близькою функціональності, не впливаючи на архітектуру програмного продукту і не накладаючи на неї ніяких обмежень. У той час як фреймворк диктує правила побудови архітектури додатку, задаючи на початковому етапі розробки поведінка за умовчанням, каркас, який потрібно буде розширювати і змінювати відповідно до зазначених вимог.

2. Ігровий рушій - центральний програмний компонент комп'ютерних та відеоігор або інших інтерактивних додатків з графікою, оброблюваної в реальному часі. Він забезпечує основні технології, спрощує розробку і часто дає грі можливість запускатися на декількох платформах, таких як ігрові консолі та настільні операційні системи, наприклад, GNU / Linux, Mac OS X і Microsoft Windows. Основну функціональність зазвичай забезпечує ігровий движок, що включає движок рендеринга («визуализатор»), фізичний движок, звук, систему скриптів, анімацію, штучний інтелект, мережевий код, управління пам'яттю і багатопоточність. Часто на процесі розробки можна заощадити за рахунок повторного використання одного ігрового движка для створення безлічі різних ігор.

3. Конструктор ігор - програма, яка об'єднує в собі ігровий рушій і інтегроване середовище розробки, і, як правило, включає в себе редактор рівнів, що працює за принципом WYSIWYG. Такі програми значно спрощує

процес розробки ігор, роблячи його доступним любителям-непрограмістів, і можуть бути використані в початковому навчанні програмуванню [1]

В роботі ми будемо досліджувати різноманітних представників всіх цих группи проте будемо приділяти більшу увагу на інструментам які мали реліз протяком останніх 5 років, оскільки світ мультимедія та ігрової індустрії дуже швидко оновлюется та прогресує, то набор інструментів 5-ти річної давності виявится неактуальним в поточних реаліях .

2.1.1 XNA Framework

Microsoft XNA (англ. XNA's Not Acronymed) — набір інструментів з керованим середовищем часу виконання (.NET), створений Microsoft для полегшення розробки комп'ютерних ігор. Мета XNA в спробі звільнити розробку ігор від написання «повторюваного шаблонного коду» і об'єднати різні аспекти розробки ігор в одній системі. Набір інструментів XNA був анонсований 24 березня 2004 на Game Developers Conference в Сан-Хосе, Каліфорнія. Перший Community Technology Preview XNA Build був випущений 14 березня 2006.

XNA Framework ґрунтується на реалізації . NET Compact Framework 2.0 для розробки для Xbox 360 і . NET Framework 2.0 на Windows. Він включає великий набір бібліотек класів, специфічних для розробки ігор, що підтримує максимальне повторне використання коду на всіх цільових платформах. Фреймворк виконується на модифікації Common Language Runtime, що оптимізована для ігор. CLR доступне дляWindows XP, Windows Vista, і Xbox 360. Так як ігри XNA пишуться для CLR, вони можуть бути запущені на будь-якій платформі, яка підтримує XNA Framework з мінімальними змінами або взагалі без них. Ігри, які запускаються на фреймворку, технічно можуть бути написані будь-якою .NET-сумісною мовою, але офіційно підтримується тільки мова програмування C# та середовище швидкої розробки XNA Game Studio Express і всі версії Visual Studio 2008.

XNA Framework приховує низькорівневі технологічні деталі, пов'язані з розробкою гри. Таким чином, фреймворк піклується про різницю між платформами, дозволяючи розробникам приділяти більше уваги смисловому вмісту гри. XNA Framework інтегрується з декількома інструментами, такими як XACT, для допомоги в створенні контенту. XNA Framework надає підтримку створення та двомірних, і тривимірних ігор і дозволяє використовувати можливості контролерів Xbox 360. Десктопні програми можуть поширюватися безкоштовно під поточним ліцензуванням Microsoft.

Існує проект MonoGame, що представляє собою кроссплатформенную open-source реалізацію XNA з додатковими можливостями.

2.1.2 Construct 2

Construct 2 — це заснований на HTML5 конструктор 2D ігор, розроблений компанією Scirra. Конструктор спрямовано в першу чергу на людей, які не розуміються в програмуванні, дозволяючи швидко створювати ігри способом Drag-and-drop з використанням візуального редактора та логічної системи, заснованої на принципі поведінки та реакції. Construct 2 є прямим нащадком попередньої версії програми, Construct Classic.

Функціональні особливості:

Система «Події та Дії» Як і в Construct Classic, основним методом програмування ігор та додатків у Construct 2 є використання «листів подій» (англ. event sheets), що схожі на файли рушія, які використовуються у мовах програмування. Коли виконується умова, задана користувачем в листі подій, слід за нею виконується дія чи функція.

Система «Поведінки» Особливістю Construct 2 в порівнянні з іншими конструкторами є так названі «поведінки» (англ. behaviors). Поведінка — це заздалегідь заготовлений набір (шаблон) властивостей об'єкту. Поведінки потрібні для прискорення процесу розробки гри, коли користувач не задає всі властивості сам в листі подій, а просто користається необхідним шаблоном.

Прикладом поведінки є поведінка 8 direction, яка дозволяє переміщувати об'єкт у восьми напрямках за допомогою клавіш.

Підтримка сторонніх плагінів. Розробники Construct 2 забезпечують та навіть заохочують створення плагінів від сторонніх розробників. Так, на офіційному сайті можна знайти поради та уроки з написання та налаштування плагінів для коректної роботи. Всі плагіни Construct 2 написано на Javascript.

2.1.3 Blender Game Engine

Blender — пакет для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, вимальовування, після-обробки відео, а також створення відеоігор.

Особливостями пакету є малий розмір, висока швидкість вимальовування, наявність версій для багатьох операційних систем — FreeBSD, GNU/Linux, Mac OS X, SGI Irix 6.5, Sun Solaris 2.8 (sparc), Microsoft Windows, SkyOS, MorphOS та Pocket PC. Пакет має такі функції, як динаміка твердих тіл, рідин та м'яких тіл, систему гарячих клавіш, велику кількість легко доступних розширень, написаних мовою Python. Починаючи з версії 2.61 з'явились функції "відстеження камери" (англ. camera tracking), та "захоплення руху" (англ. motion capture або mocap).

Програма є вільним програмним забезпеченням та розповсюджується під ліцензією GNU GPL.

Ігровий рушій Blender Game Engine (BGE), є вбудованим компонентом Blender. У своєму складі він має вбудований фізичний движок Bullet, підтримку мережі через скрипти Python, графічний рендер з повноцінною підтримкою шейдеров. Blender надає чудовий інтерфейс конструктора, з можливістю легко налаштувати ігрову логіку Або як альтернатива написання скриптів Python безпосередньо звертатися до API.Для створення гри немає необхідності виходити з програми, більше того наявні інструменти що полегшують тестування , наприклад перегляд роботі шейдерів без запуску графічного рушія, тощо.

Ігровий рушій Blender використовує систему графічних «логічних цеглин» (поєднання «датчиків», «контролерів» і «приводів») для контролю руху й відображення об'єктів у рушії. Ігровий рушій також може бути розширений за допомогою набору Python палітурки. Як Blender, він використовує OpenGL як шар крос-платформної графіки, для зв'язку з графічним обладнанням.

2.1.4 EaselJS

EaselJS - Javascript бібліотека що надає можливість працювати з Canvas у графічному режимі включаючи повний список ієрархічного відображення, модель взаємодії основного і допоміжних класів, щоб полегшити роботу з з 2D-графікою в Canvas. EaselJS забезпечує рішення для роботи з багатою графікою і інтерактивністю в HTML5 Canvas.

EaselJS також має вбудовану підтримку:

- Особливостей Canvas, такі як тіні і CompositeOperation
- Ticker, глобальний heartbeat, на яке об'єкти можуть підписатися
- Фільтри, в тому числі ColorMatrixFilter, AlphaMaskFilter, AlphaMapFilter і BlurFilter.
- Утиліта ButtonHelper, полегшує створення інтерактивних кнопок
- SpriteSheetUtils і SpriteSheetBuilder, щоб допомогти побудувати і керувати функціональністю SpriteSheet під час виконання.

Всі сучасні браузери, які підтримують Canvas будуть підтримувати EaselJS. Проте продуктивність може різнитися між платформами, наприклад, Android Canvas має погану апаратну підтримку, і прцює , в середньому, повільніше ніж більшість інших браузерів.

2.1.5 Game Maker

Game Maker — один з найвідоміших конструкторів ігор. Написаний на Delphi Доступний для ОС Windows. Будучи професором Утрехтського університету Марк Овермарс почав розробляти Game Maker як навчальний

посібник для своїх студентів. Створення ігор в ньому не потребує попереднього знайомства з будь-якою з мов програмування.

Гра в Game maker будується як набір ігрових об'єктів. За їх зовнішній вигляд відповідають спрайт, а поведінка задається шляхом опису реакцій на події. Для цього можна використовувати графічне представлення програм (близьке до блок-схемами) у вигляді послідовності іконок-дій. Програмування за допомогою дій відбувається в режимі drag-n-drop. Наприклад, для того щоб почати умовний оператор, потрібно перетягнути на панель дій восьмикутник з іконкою, що означає тип перевірки, а потім, можливо, ввести будь-які значення в форму, що з'явилася. Для більш просунутих користувачів є скриптова мова GML схожий на JavaScript і C++, є можливість створення власних бібліотек дій, використовуючи Library Maker.

Розрахований в основному на створення двовимірних (2D) ігор будь-яких жанрів. Також підійде для створення різних презентацій і т. п. Game Maker розповсюджується на умовах Shareware, безкоштовна версія обмежена в функціональності, а при запуску ігор на стрічці завантаження гри показується логотип Game Maker.

Остання версія 8.1. Більше Game Maker не підтримується, його місце зайняло кроссплатформений розвиток проекту - Game Maker: Studio.

2.1.6 Unity

Unity — багатоплатформовий інструмент для розробки дво- та тривімирних застосунків та ігор, що працює на операційних системах Windows і OS X. Створені за допомогою Unity застосування працюють під системами Windows, OS X, Android, Apple iOS, Linux, а також на гральних консолях Wii, PlayStation 3 і XBox 360.Є можливість створювати інтернет-застосунки за допомогою спеціального під'єднуваного модуля для браузера Unity, а також за допомогою експериментальної реалізації в межах модуля Adobe Flash Player. Застосування, створені за допомогою Unity, підтримують DirectX та OpenGL.

Редактор Unity має простий Drag & Drop інтерфейс, який легко налаштовувати, що складається з різних вікон, завдяки чому можна проводити налагодження гри прямо в редакторі. Рушій підтримує три сценарних мови: C #, JavaScript (модифікація). Проект в Unity ділиться на сцени (рівні) - окремі файли, що містять свої ігрові світи зі своїм набором об'єктів, сценаріїв, і налаштувань. Сцени можуть містити в собі як, об'єкти (моделі), так і порожні ігрові об'єкти – тобто ті які не мають моделі. Об'єкти, в свою чергу містять набори компонентів, з якими і взаємодіють скрипти. У об'єктів з видимої геометрією також за замовчуванням присутній компонент Mesh Renderer, що робить їх модель їх видимою.

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

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

У Unity вбудована підтримка мережі.Ігровий рушій повністю пов'язаний із середовищем розробки. Це дозволяє випробовувати гру прямо в редакторі. Має вбудований генератор ландшафтів

2.1.7 Unreal Engine 3

Unreal Engine 3 - Написаний мовою C++, рушій дозволяє створювати ігри для більшості операційних систем і платформ: Microsoft Windows, Linux, Mac OS і Mac OS X, консолей Xbox, Xbox 360, PlayStation 2, PlayStation Portable,

PlayStation 3, Wii, Dreamcast і Nintendo GameCube. У березні 2010 робота рушія була продемонстрована на комунікаторі Palm Pre, що базується на мобільній платформі webOS.

Для спрощення портування рушій використовує модульну систему залежних компонентів: підтримує різні системи рендерингу (Direct3D, OpenGL, Pixomatic), відтворення звуку (EAX, OpenAL, DirectSound3D), засоби голосового відтворення тексту, розпізнавання мовлення (тільки для Xbox360, PlayStation 3, Nintendo Wii і Microsoft Windows), модулі для роботи з мережею й підтримка різних пристроїв вводу.

Для гри у мережі підтримуються технології Windows Live, Xbox Live, і GameSpy, включаючи до 64 гравців (клієнтів) одночасно. Попри те, що офіційно засоби розробки не містять у собі підтримки великої кількості клієнтів на одному сервері, рушій використовувався для створення MMORPG-ігор. Один з найвідоміших представників жанру, Lineage II, використовує рушій Unreal Engine.

5 листопада 2009 року був випущений пакет Unreal Development Kit, безкоштовна версія Unreal Engine 3 для некомерційного використання з можливістю купівлі дешевої комерційної ліцензі

Для описання логіки гри використовується C++ подібний UnrealScript. Усі елементи ігрового рушія представлені у вигляді об'єктів, що мають набір характеристик, і клас, який визначає доступні характеристики. У свою чергу будь-який клас є «дочірнім» класом object. Серед основних класів і об'єктів можна виділити наступні:

Актор (actor) — базовий клас, що містить усі об'єкти, які мають відношення до ігрового процесу й мають просторові координати.

Пішак (pawn) — фізична модель гравця або об'єкта, керованого штучним інтелектом. Метод керування описаний спеціальним об'єктом, такий об'єкт називається контролером. Контролер штучного інтелекту описує лише загальну поведінку пішака під час ігрового процесу, а такі параметри як «здоров'я» (кількість пошкоджень, після яких пішак перестає функціонувати) або,

наприклад, відстань, на якій пішак звертає увагу на звуки, задаються для кожного об'єкта окремо.

Світ, рівень (world, game level) — об'єкт, що характеризує загальні властивості «простору», наприклад, силу тяжіння й туман, у якому розташовуються всі актори. Також може містити в собі параметри ігрового процесу, як, наприклад, ігровий режим, для якого призначений рівень.

2.1.8 Turbulenz

Бібліотеки двигуна реалізовані та оптимізовані на JavaScript для підтримки швидких ітерації ігрового коду і даних. Двигун виконується безпосередньо в браузері, і включає в себе деякі з наступних функцій:

Асинхронне завантаження ресурсів і обмінювати; Лінивий оцінка оновлень сцени; Багатопоточная перевірка і виконання

Візуалізація ефектів і частинок . Shader на основі режиму негайного відправлення; Підтримка мульти-техніка, багатоходової, мульти-матеріали; Динамічна вершина, індекс і обробки текстур буфера; Відкладений надання підтримки необмежені вогні; Знімні ефекти POST і колекція ефектів; Експонентний карти тіней і оклюзія запитів; Великі частинок і ефект системи; GUI система / HUD підтримки декількох мов і шрифтів

Фізика, Зіткнення і анімація. Жорсткі тіла, зіткнення примітиви і обмежень; Рей і опуклі запити розгортки; Велика колекція вбудованих контролерів анімації; Скелет анімації з кватерніонів

Аудіо та Переферія. 3D джерел звуку і до 7.1 об'ємного звуку; Многопоточная потокової передачі і змішування; Доступ до HID, що дозволяє зовнішні контролери і периферійні пристрої

Мережа, Мультіплеер і соціальні мережі. Стиснення, шифрування, надійної і ненадійною передачі повідомлень; Компенсація мережу лаг, клієнт / сервер і p2p архітектури; Інтеграція з популярними соціальними мережами, включаючи Facebook; Автоматичний програвач входу і доступу автоматично обробляються

Сцена і управління ресурсами. Видимість запитів через портали, усіченого або перекриттям коробки; Сортування і угрупування для видимості і оптимальної обробки; Пропускна здатність і апаратне масштабування за допомогою динамічного вибору активів

2.2 Підбиття проміжних підсумків аналізу інструментальних засобів

Для полегшення аналізу зберемо данні по доступності, актуальності, популярності та основних функціональних можливостей спираючись на данні зібрані з офіційних порталів та ресурсів інструментальних засобів у таблиці та гістаграми.

Таблиця 2.1 Аналіз доступтоності та актуальності інструментальніх засобів

Назва	Доступність	Мова написання гри	Дата останнього релізу
XNA	Безкоштовна	C#	06.10.2011
MonoGAMe	Безкоштовна	C#	29.04.2015
Phaser	Безкоштовна	JS/Typo script	22.04.2016
Construct 2	Безкоштовна тріальна версія	none	26.05.2015
pixi.js	Безкоштовна	JS/HTML 5	22.12.2015
Blender Game Engine	Безкоштовна	None / python	31.10.2013
Easel.JS	Безкоштовна	js	15.11.2015
Turbulenz	Безкоштовна	js	22.12.2015
melonJS	Безкоштовна	js	03.02.2016
PandaJS	Безкоштовна	js	17.02.2015
Game Maker	Безкоштовна	none /GML	13.08.2013
Unity	Умовно безкоштовна[1]	C# /js/ Python	15.03.2016
Unreal Engine	Умовно безкоштовна[1]	Unreal Script	31.03.2016

[1]

1 Умовно безкоштовна - можна використовувати інструменти для розробки ігор в комерційних цілях доки дохід з них не перевищує певної межі, далі потрібно сплачувати відсоток від доходів

Таблиця 2.2 Аналіз функціональних можливостей інструментальніх засобів

Назва	Функціональні можливості				
	2D - графіка	3D - графіка	Обробка колізії	Обробка фізики	Вбудовані механізми відладки
XNA	так	так	ні	ні	так
MonoGAME	так	так	ні	ні	так
Phaser	так	ні	так	так	ні
Construct 2	так	ні	так	так	так
pixi.js	так	ні	ні	ні	ні
Blender Game Engine	так	так	так	так	так
Easel.JS	так	ні	ні	ні	ні
Turbulenz	так	так	так	так	так
melonJS	так	ні	так	так	ні
PandaJS	так	ні	так	так	так
Game Maker	так	так	ні	ні	ні
Unity	так	так	так	так	так
Unreal Engine	так	так	так	так	так

2.2.1 Аналіз популярності інструментальних засобів

У якості характеристики популярності серед спільноти будемо використовувати кількість щоденних відвідувачіа офіційного ресурсу

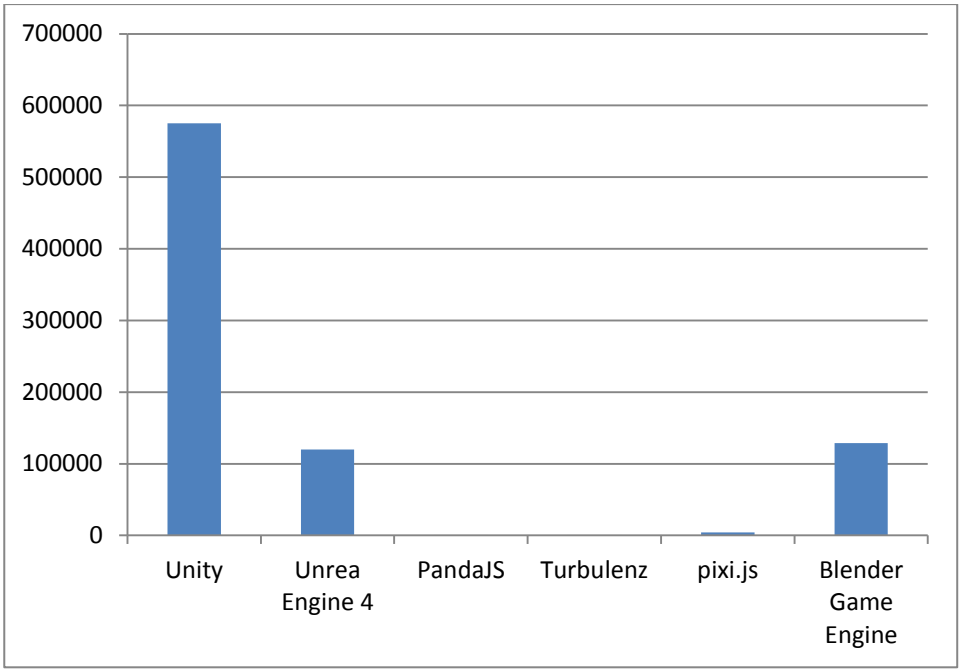

Рис 2.1 Гістограма кількості щоденних відвідувачів на офіційного ресурсу ігрового рушія

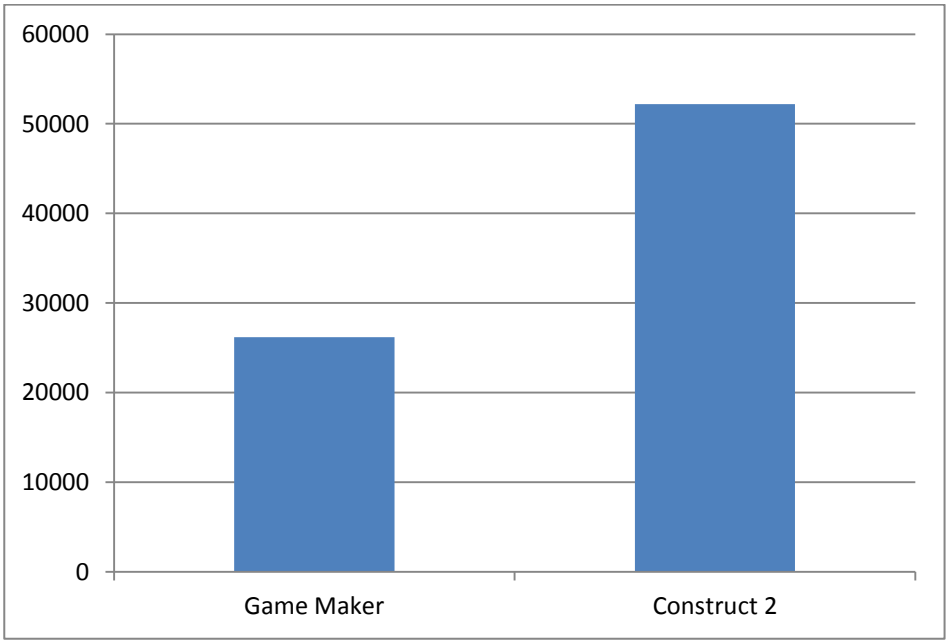

Рис 2.2 Гістограма кількості щоденних відвідувачів на офіційного ресурсу ігрових конструкторів

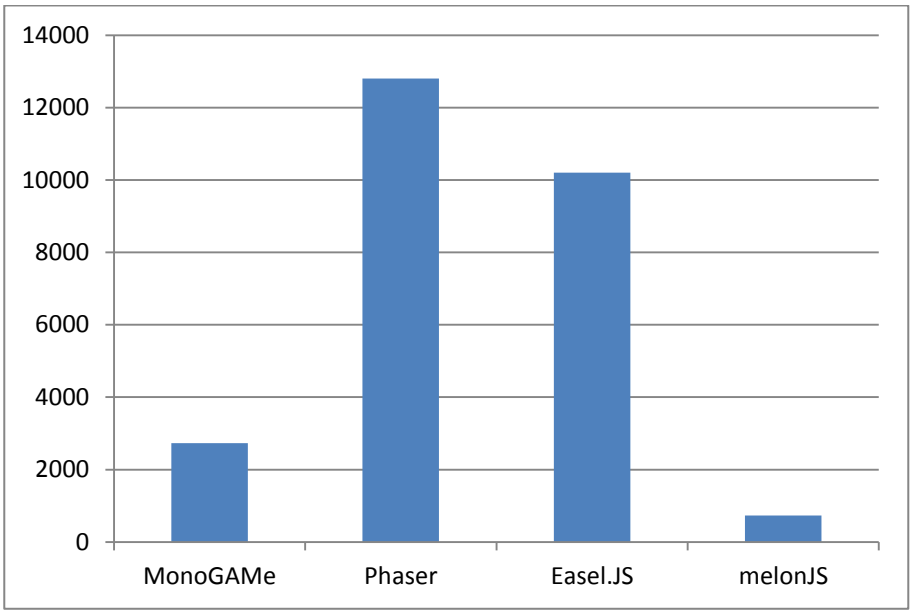

Рис 2.3 Гістограма кількості щоденних відвідувачів на офіційного ресурсу ігрових фреймворків

2.3 Висновки

Ми розглянули найбільш популярні сучасні засоби для розробки ігор, розбили їх на класи та проаналізували їх базові заявлені функціональні можливості. Для більш детального аналізу оберемо по одному найоптимальнішому представнику з кожного класу для розробки навчального проекту прототипу гри.

Серед ігрових рушіїв у колі незалежних розробників, останнім часом, найбільшою популярністю користуется Unity. Він повністью задовольняє нас за своїми базовими характеристиками до тогож має потужну підтримку товариства, що значно знижує поріг входження.

Серед конструкторів, досить привабливо виглядає Construct 2 , що номінально обіцяє широкий спектр функціоналу та кросплатформеність. До того ж знову таки, має чималу популярність та підтримку товариства користувачів , незважаючи на відносну молодість проекту. Єдиним мінусом є орієнтованість лише на 2-D розробку, що доведеться врахувати при формуванні концепції для прототипів.

Серед фреймворків для розробки ігор за функціональними можливостями можна виділити Phaser та melonJS , якщо не звертати увагу на те, що вони також лише для 2-D проектів. MelonJS вийшов у світ раніше та має більше реалізованих проектів, проте Phaser показав свою конкурентоздатність та стрімко набирає популярності, на разі він має майже в десятеро більшу кількість відвідувань офіційної сторінки в день[15]. Тому оберемо Phaser як більш прогресивний засіб з активнішим товариством.

3. РОЗРОБКА ПРОЕКТІВ ПРОТОТИПІВ ДЛЯ ДЕТАЛЬНОГО АНАЛІЗУ МОЖЛИВОСТЕЙ ХАРАКТЕРНИХ ПРЕДСТАВНИКІВ КЛАСІВ

3.1 Задачі прототипа

Окреслимо основні задачі прототипа для більш детального аналізу та порівняння обраних інструментальних засобів. Серед обмежень зумовлених вибором середовища розробки актуальною є використання 2-D графіки та фізики.

Перш за все гра-прототим має показати можливості роботи фізичного рушія та обробку колізії. При аналізі слід приділити увагу співвідношенню отриманого результату зі складністю її імплементації.

По-друге ми маємо продемонструвати роботу з графічними ресурсами, обробку анімації, ефектів тощо. Для подальшого аналізу, кожен прототип буде побудований на базі одних і тих же графічних матеріалів.

По-третє потрібно освітлити роботу з логікою гри, її взаємодію зі сценою, об’єктами. Тако слід приділити окрему увагу при аналізі на трудомісткість впровадження логіки в різні частини ігрового проекту.

На останок, слід проаналізувати можливості впровадження взаємодії гравця з грою, тобто підтримку периферійних пристроїв та їх обробку, роботу з елементами графічного інтерфейсу тощо.

Спираючись на означенні задачі та обмеження почнемо формувати опис прототипу. Спочатку визначимось із жанром майбутніх ігрових проектів. Для 2-D графіки класичними представниками є аркадні платформери, рольові ігри, та top-down шутери (камера висвітлює сцену згори). Рольлові ігри можна виключити за недостатністю можливостей продемонструвати роботу фізики та колізії, та надмірною вимогливостю до логіки та сценарію. А от шутери дають нам можливість гармонічно використати декілька приладів вводу, один для

руху персонажа та інший для переміщення прицiла. Вони не зав’язані на складній логіці, мають прямолінійний сценарій, або навіть ніякого та мають певні вимоги до обробки колізії, особливо куль, снарядів, тощо. Платформери ,як правило, мають спрощенне керування, також не обтяжуються глибокою логікою натомість, основну увагу приділяють саме фізиці та колізії, концентруючи увагу гравця саме на цій частині взаємодії обьєктів.

На мою думку найоптимальнішим буде варіант проекту – шутер з видом зверху з наступними особливостями:

- Гравцем що рухається по сцені та рухомим прицілом
- Гравець та вороги мають певний запас умовних одиниць здоров’я.
- Гравець своїми пострілами зменшує запас здоров’я ворогів
- Вороги , що підійшли надто близько зменшують запас здоров’я гравця.
- Об’єкти без запасу здоров’я помирають.
- Гравець повільно відновлюї свій запас здоров’я до максимуму
- Нескінченна кількість ворогів, що будут зявлятися з часом.
- На карті мають бути перешкоди . Їх можна зруйнувати
- За вбивство ворогів нараховуються бали
- Бали, запаз здоров’я гравця мають відображатися на єкрані

Перераховані вище вимоги будуть мінімально допустимими для створення прототипів, особливості над цього списку будуть впроваджуватись з точки зору доступності в тій чи іншій інструментальній середі.

3.2 Реалізація прототипу на Construct 2

3.2.1 Робота з графічними ресурсами

В Construct2 є вбудований редактор зображень в якому є можливість редагувати зображення, накладати фільтри. Результатом роботи над зображенням буде об`єкт спрайту. Цей об’єкт характеризується не лише

зображенням, він має цілий перелік властивостей та характеристик які допоможуть настроїти поведінку спрайту та способи його відображення.

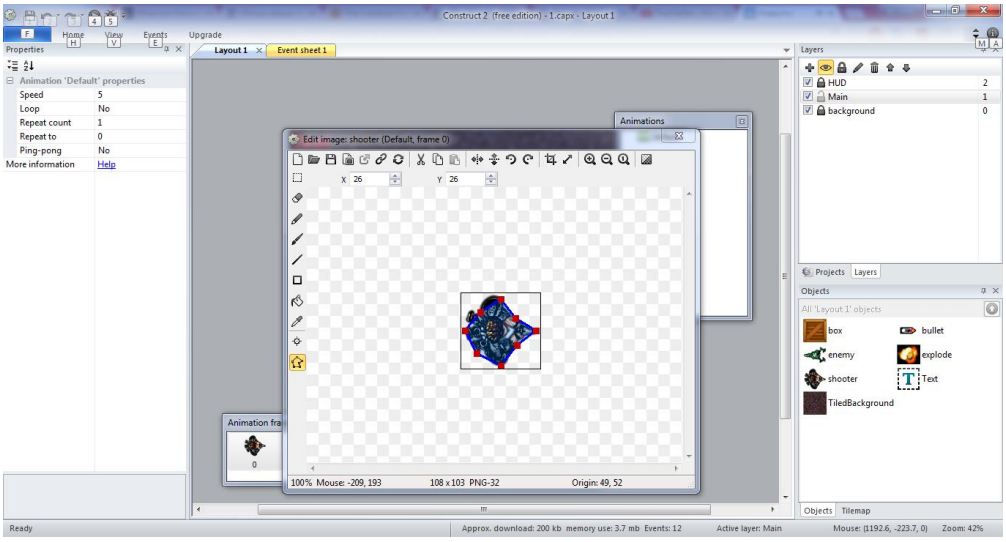

Рис. 3.1 Графічний редактор Construct 2

Надалі, ми взаємодіємо саме з об’єктами спрайтів, для контролю відображення на сцені, появі, перміщенню, тощо. Окрім графічних змін в редакторі таж є можливість змінювати полігон колізії ,на рис.3.1 він позначений синіми лініями з червоними гранями. Після закрузки зображення редактор автоматично намагається створити такий полігон, він знадобится нам пізніше для роботи з фізикою та колізією.Проте як вже , напевно стало зрозуміло два обьекта вважаються зіткнутими, якщо їх полігони зіткнулись. Також в редакторі є можливість додавати Image Points тобто якісь особливі точки на зображенні до яких поті можна буде прив’язатись для взаємодії, апріорі найпершою такою точкою є середина спрайту. Ця можливість нам сталаи у нагоді при обробці пострілів, адже ми змогли поставити таку точку на кінці пістолета та створювати кулі відштовхуючись від цієї точки а не від центру спрайту гравця.

3.2.2 Організація взаємодії з гравцем

Для того щоб дати змогу гравця переміщювати свого аватара, потрібно підключити обробку клавіатури та мишки. Оскільки це конструктор, яким

можуть користуватися навіть люди без навичок у програмуванні, для багатьох речей що притаманні іграм вже існують створенні об’єкти і нам потрібно лише обратиїхз контекстного меню Insert Object

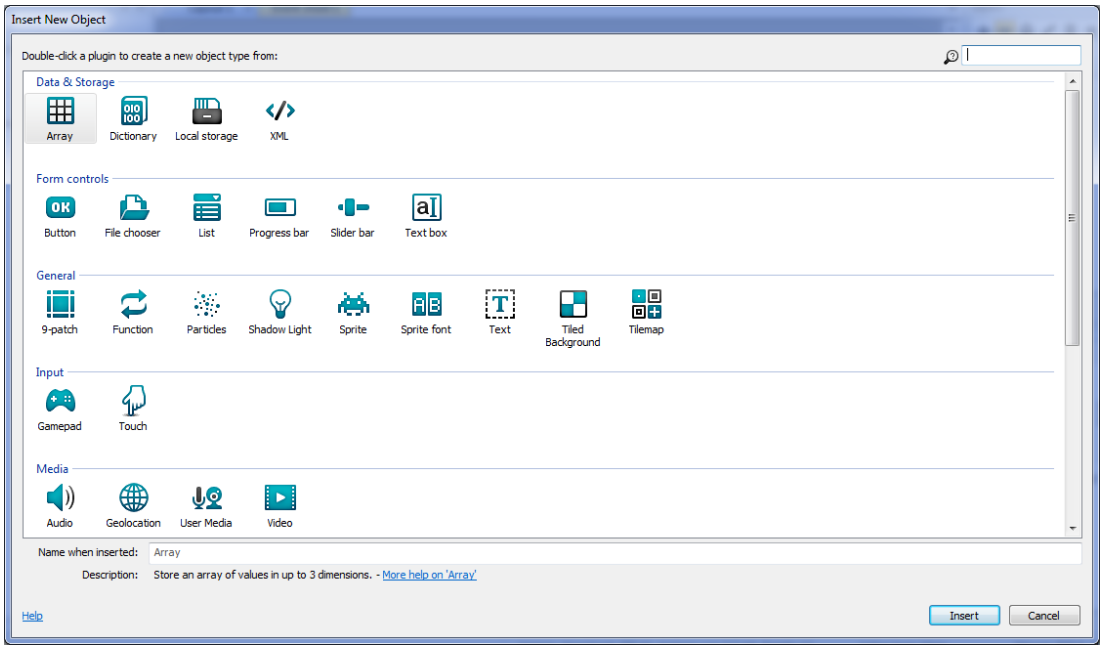

Рис 3.2 Меню вставки об’єктів Construct 2

Там присутні обьекти звуку, спрайтів, тексту, заднього плану, контролерій та інших периферійних пристроїв. По суті вся робота з тим чи іншум об’єктом полягає в додаванні його на сцену на настраюванні його властивостей. Використовуючи ці об’єкти мо додали до сцени задній фон, контролер типу миша та клавіатуру. Таким чином надалі при роботі з логікою ми зможемо звертатися до цих об’єктів та дізнаватися про маніпуляції гравця.

3.2.3 Робота з фізикою та колізією

Обробка фізики, колізії, та багатьох специфічних елементів поведінки об’єктів що притаманна більшості ігор і забрала б багато часу на реалізацію в логіці гри , реалізується за допомогою властивості Behaviors.

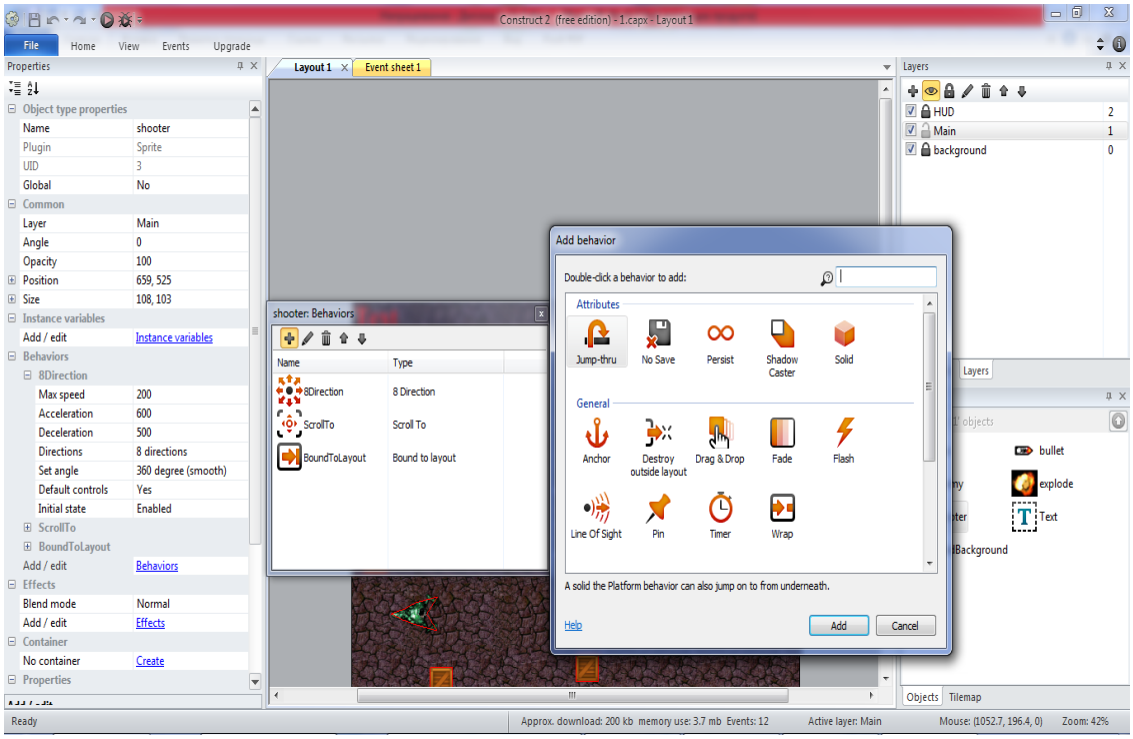

Рис 3.3 Діалогое вікно додавання поведінкт Construct 2

Тут присутні такі елементи як преміщення різного типу (8 напрямім, car-like та bullet-like),тінь, пошук шляху, та багато чого ще. На разі для гравця ми обрали преміщеня у 8 напрямах, заборону на вихід за межі сцени, прив’язка камери до обьекту. Надалі можна окремо настроїти властивості самої поведінки, так ми вказали для преміщення цільовий контролер – клавіатуру. Для обробки колізії потрібно що обидва об’єкта мали поведінку «Твердого тіла» а для роботи з фізикою – поведінку «Фізика». Під поняттям фізика наразі мається на увазі обробка сили тяжіння, деформації, масси, густоти, інерції тощо.

3.2.4 Робота з логікою гри

Найцікавіша частина – створення логіки гри без жодної строчки коду. Це релізовано завдяки окремії від сцени сутності під назвою Таблизя явищ. Як і передбачає назва тут знаходятся певні явища які ми очкуємо, перевірка певних критеріїв та реакція у разі успішної перевірки. Є можливість додавати декілька реакцій на одне явище. Ця таблиця також заповнюється завдяки послідовних

діалогових вікон. Ця таблиця явищ опитується раз за кожний «тік» почерзі з гори до низу. Кількість тіків в секунду можна також налаштувати у властивостях самого проекту, початкове значення 60.

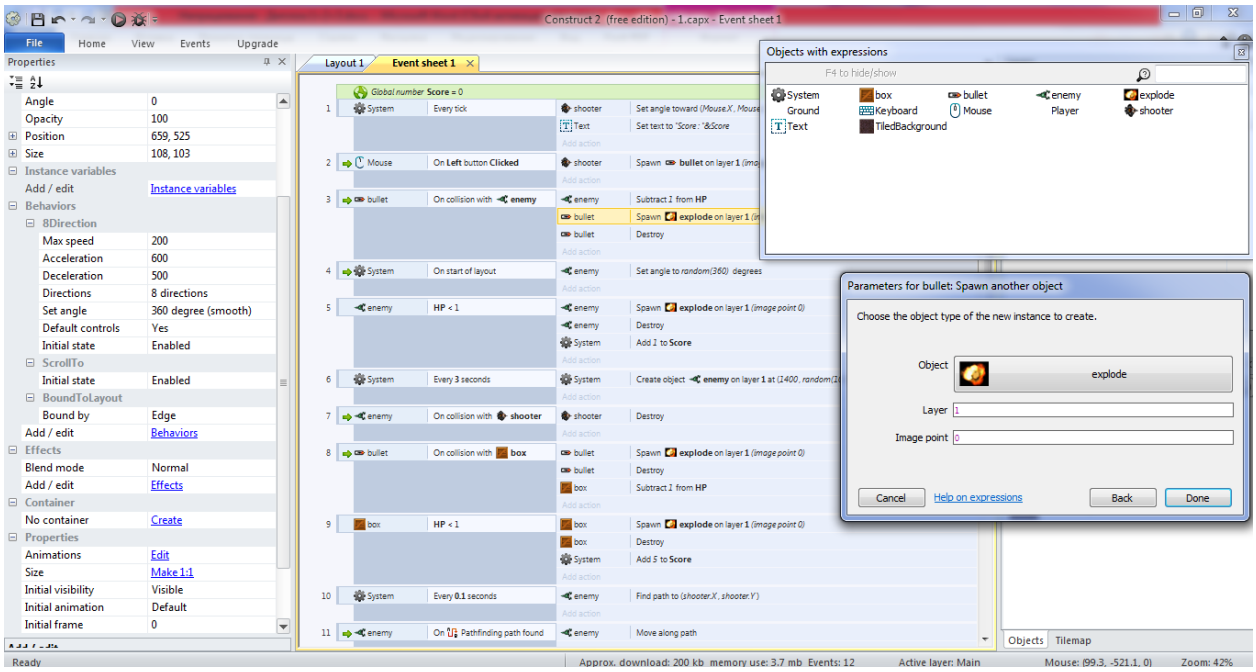

Рис 3.4 Таблиця явищ та діалогові вікна для їх редагування Construct 2

Перш за все створюючи новий запис в таблиці обирається об’єкт для опитування, далі стан який очікуєся від цього об’єкту, далі об’єкт над яким треба виконати дію реакції і саму дію. Наприклад : Мишка / Натискання лівої кнопки / гравець/ створити об’єкт кулі на Image Points №1. Таким чином і будуєтся вся логіка гри, взаємодія об’єктів тощо.

3.3 Реалізація прототипу на Unity

Unity studio – це окрема середа розробки з набором вбудованих редакторів та інструментів для відладки гри. Є можливість запустити гру а самій студії, слідкувати за змінними , емулявати дії гравця, тощо.

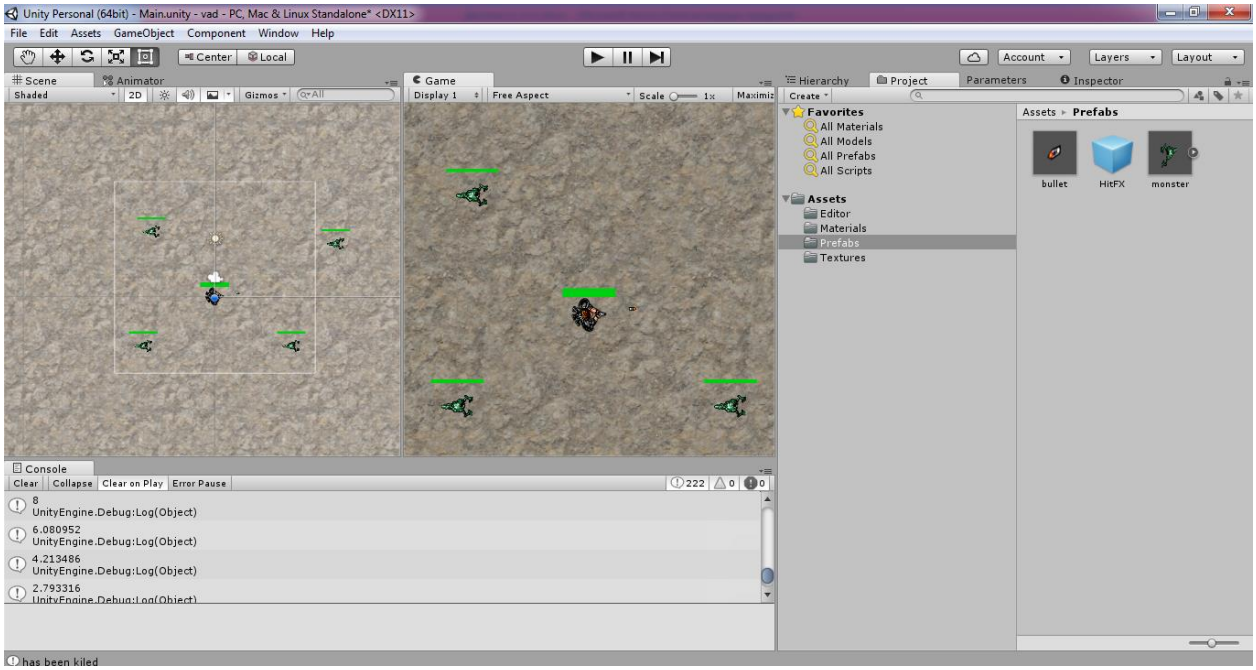

Рис.3.5 головний екран Unity studio, редактор сцени та предпоказ ігрового екрану

3.3.1 Робота з графічними ресурсами

Щоб додати графічні ресурси варто лише приєднати їх до проекту у спеціальному менеджері проекту. Студія автоматично розцінює графічні файди як текстури, проте якщо це набір спрайтій потрібно лише змінити властивість об’єкту на спрайт. Також присутній редактор спрайтів, де можна нарізати спрайти автоматично, чи вручну, задати центр для кожного спрайту, тощо.

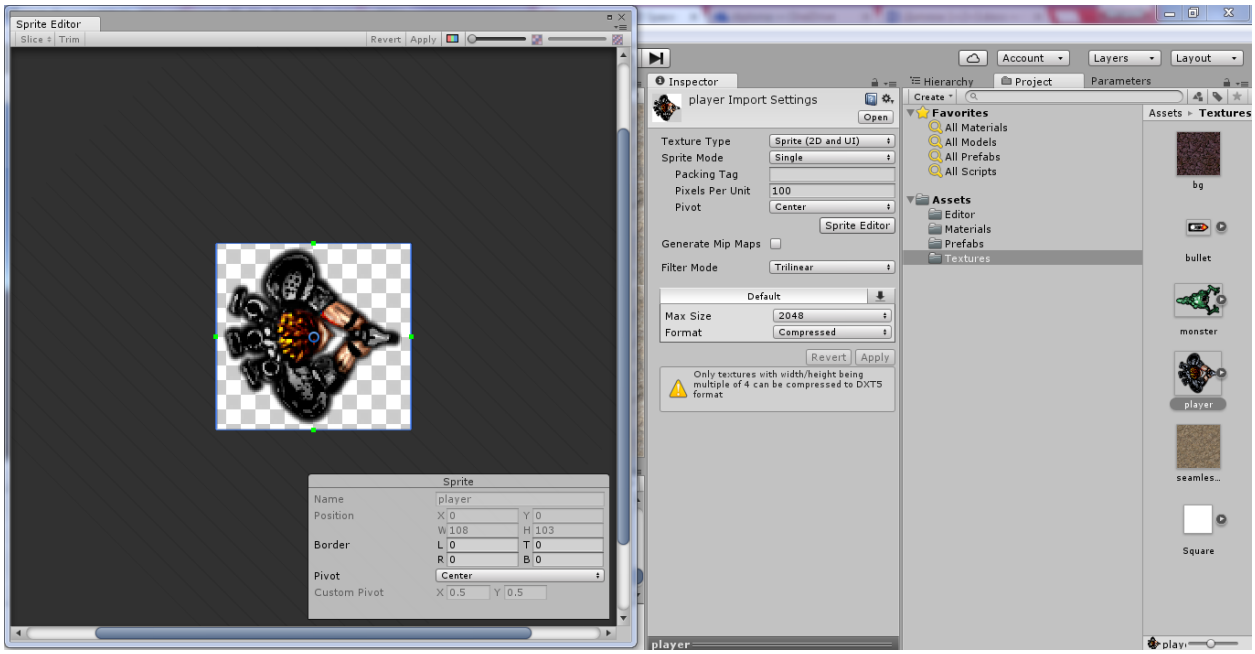

Рис 3.6 Робота з графічними ресурсами в Unity

3.3.2 Робота з фізикою та колізією

В Unity э декілька вбудованих рушіїв для 2D та 3D та декілька способів взаємодії з ними. Перш за все потрібно додати компонент Collider до об’єкту гри, тип Collider буде визначати тип рушія, для прототипу використаємо Physics 2D Collider. Цей колайдер буде використовуватись для визначення претинута доторкання об’єктів з іншими колайдерами, додаткові налаштування triggers дозволяє обмету проходити через інші об’єкти проте так само викликати подію накладіння, напроти Cinematic objects буде зупиняти об’єкти що доторкаються до нього , проте не будуть викликати подію накладання якщо будуть рухатись самі.

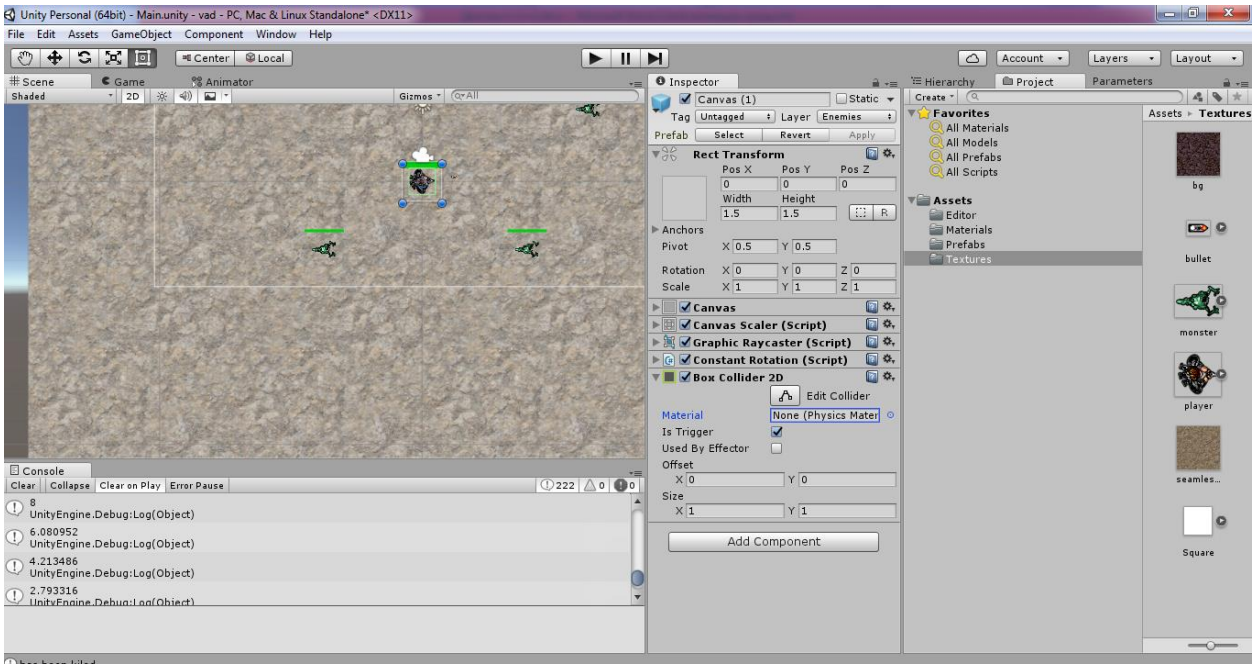

Рис 3.7 Налаштування Collider в Unity

Ще одною важливою деталю налаштування колізії є рівні(Layers) можна назначити певним об’єктам певні рівні. Обьєети між різними рівнями не взаємодіють, до того ж можна налаштувати колізію між об’єктами на одному рівні. Таким чином для нашого прототипу буде доцільно вимкнути взаємодію куль між собою, кулі та гравця та ворогів один з одним.

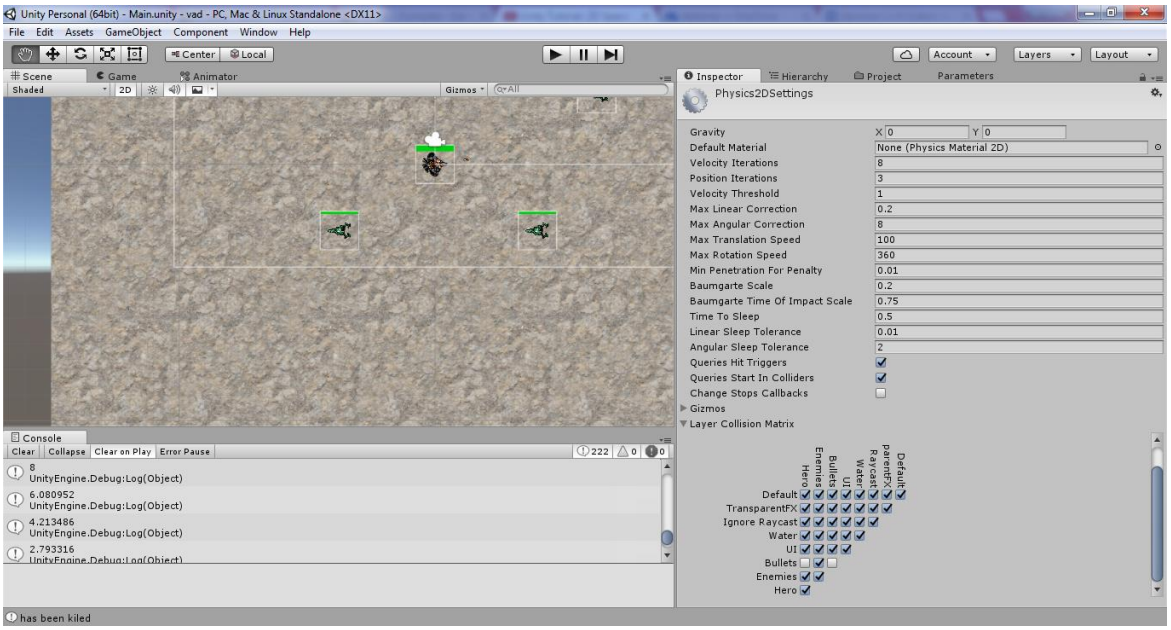

Рис 3.8 Layer collision matrix в Unity

3.3.3 Робота з логікою гри

Робота з логікою гри проходить за допомогою написання скриптів та додаванню ції скриптів до сцени чи обраного об’єкту. Головна концепція скриптів це використання певних явищ чи методів з періодичним викликом. Таким чином цей скрипт можна приєднати до декількох об’єктів та декілька скриптів до одного. Скажімо скрипт ,що під час кожного виклику методу Update перевіряє чи не знизився показник здоров’я цілі до 0, і якщо так то знищює його. Такий скрипт можна додати як до об’єкту гравця так і до об’єктів ворогів і навіть перешкод що моуть руйнуватись.

Скрипти пишуться на мові C# в MS Visual Studio з якою інтегрується проект створенний в Unity studio. Таким чином стає доступним весь арсенал відладки Visual Studio при роботі з грою.

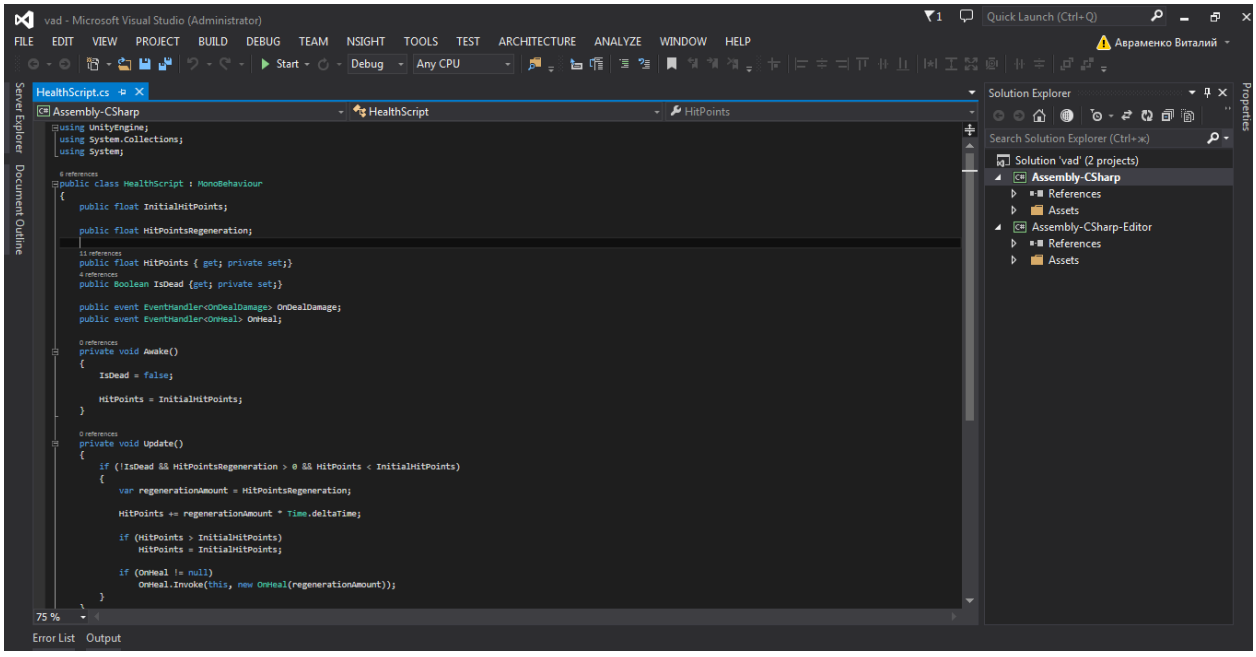

Рис 3.9 Скрипт для Unity в Visual Studio

3.3.4 Організація взаємодії з гравцем

В Unity існує два методи, призначення яких обновлювати данні, Fixedupdate та Update. Різниця в тому, що Fixedupdate викликається зі сталою

періодичністю в приблизно 2 милисекунди і використовується для обробки фізики, зміни швидкості, тощо. Тоді як Update викликається кожного разу перед оновленям фрейму, іноді пріод цього виклику більше 2 мілесекунд іноді менше. Для обробки сингалів від гравця використовується саме методі Update, адже він викличется одразу ж після отримання сигналу. В цьому методі ми маємо доступ до об’єкту input, з його допомогою дізнаємося про стан клавіатури та очікувати настикання якоїсь певної клавіші. Або натомість використати більш гнучке рішення звернушись до властивості input під назвою Axes – своєрідний рівень абстракції до якого можна прив’язати логіку гри. Таким чином незалежно від того, буде використовуватися клавіатура, джойстик чи навіть тачскрін як пристрій вводу, гра буде коректно працювати. Настроїти об’єкти Axes можна в спеціальному менеджері, задати ім’я, та сигнали за замовчуваням.

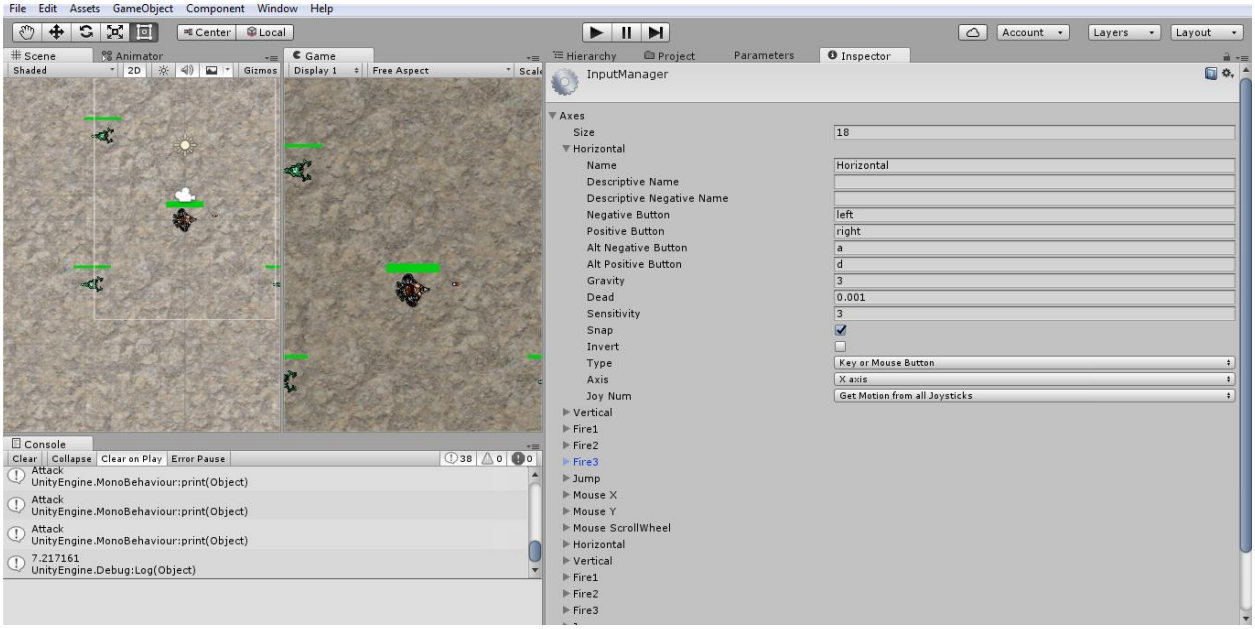

Рис. 3.10 Менеджер налаштування об’єктів Axes в Unity

3.4 Реалізація прототипу на Phaser

Phaser.js не має власного редактору чи середовища розробки, фреймворк поставляжтся у вигляді даваскиптового файлу. Прочатковою точкою проекту

виступає єдиний html файл в якому підключается скрипт фреймворку та файли скрипту з кодом гри. Для запуску гри треба лише запустити html файл в браузері.

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Phaser Prototype</title>
    <meta charset="utf-8">
    <script src="js/phaser.min.js"></script>
    <script src="js/game.js"></script>

</head>
</body>
</html>
```

Рис 3.11 Приклад html файлу для запуску гри в Phaser.js

В файлі скрипту гри треба оголосити змінну класу Phaser.Game та предати туди екземпляр класу Phaser.State в якму містится три методи preload, create, update.

3.4.1 Робота з графічними ресурсами

Для того щоб працювати з графічними елементами, перш за все варто їх завантажити в методі preload та назначити їм зручний аліас за допомогою якого ми будемо надалі до них звертатись.

Надалі в методі create мо можемо створити ігровий об’єкт використовуючи аліас спрайту, задати йому початкове положення, розтянути, перевернути зображення, тощо. Також є можливість задати anchor - це точка відносно якої буде позиціонуватися об’єкт на сцені, в термінах Phaser вона називається world. Початкові значення цієї точки (0,0) – це верхній лівий кут, значення anchor задається у відносних одиницях для прикладу точка (0.5,0.5) це центр спрайту. Також є можливіст завантажити зображення зі спрайтовою анімацією та вказати розмір фреймів і фреймвор автоматично розділить його на мисив зображень.

```
var game = new Phaser.Game(800, 600, Phaser.AUTO, 'phaser-example', { preload: preload, create: create, update: update, render: render });

function preload () {

    game.load.image('player', 'assets/games/tanks/player.png');
    game.load.image('enemy', 'assets/games/tanks/monster.png');
    game.load.image('logo', 'assets/games/tanks/logo.png');
    game.load.image('bullet', 'assets/games/tanks/bullet.png');
    game.load.image('earth', 'assets/games/tanks/bg.png');
    game.load.spritesheet('kaboom', 'assets/games/tanks/explosion.png', 64, 64, 23);

}
```

Рис 3.12 Приклад реалізації методу preload в Phaser

3.4.2 Робота з фізикою та колізією

Для підключення фізики до об’єктів гри в методі create варто задати екземпляр об’єкту фізики до об’єкту гри та до властивості body ігрового об’єкту. У Phaser існує 3 варіанта фізики Arcade,Ninja,P2 для різних типів ігор, в нашому випадку використовуємо варінт Arcade для ігор з видом згори.

Надалі налаштування поведінки кожного об’єкту реалізужтся завдяки взаємодіє з властивістю body цих об’єктів, тобто ми задаємо колізію, прискорення, швидкість , гравітацію тощо.

```
function create () {

    //  Resize our game world to be a 2000 x 2000 square
    game.world.setBounds(-1000, -1000, 2000, 2000);

    //  Our tiled scrolling background
    land = game.add.tileSprite(0, 0, 800, 600, 'earth');
    land.fixedToCamera = true;

    //  The base of our tank
    tank = game.add.sprite(0, 0, 'player');
    tank.anchor.setTo(0.5, 0.5);


    //  This will force it to decelerate and limit its speed
    game.physics.enable(tank, Phaser.Physics.ARCADE);
    tank.body.drag.set(0.2);
    tank.body.maxVelocity.setTo(400, 400);
    tank.body.collideWorldBounds = true;


    //  The enemies bullet group
    enemyBullets = game.add.group();
    enemyBullets.enableBody = true;
    enemyBullets.physicsBodyType = Phaser.Physics.ARCADE;
    enemyBullets.createMultiple(100, 'bullet');

    enemyBullets.setAll('anchor.x', 0.5);
    enemyBullets.setAll('anchor.y', 0.5);
    enemyBullets.setAll('outOfBoundsKill', true);
    enemyBullets.setAll('checkWorldBounds', true);
```

Рис 3.13 Приклад методу Create та роботи з фізикою у Phaser

3.4.3 Робота з логікою гри та взаємодія з гравцем

Вся логіка гри зібрана в методі update що викликається декілька разів за секунду. Обробка реакції гравця також реалізується в цьому методі. Використовуючи методи об’єкту фізики ми можемо легко первірити чи зіштовхнулись, наклались 2 обьєкта, задати їм швидкість тощо. Також ми мажмо обробити реакцію на натиснення клавіш пермішення , пострілу та реакцію камери на переміщення гравця. Для обробки складних дій , аби зберігати читабельність коду, доцільно виносити їх в окремі функції та викликати в методі update або навіть створювати цілі класси для обробки складної логіки, виносити їх в окремі фали і підключити в html документі на початку.

```
function update () {
    game.physics.arcade.overlap(enemyBullets, shooter, bulletHitPlayer, null, this);
    enemiesAlive = 0;
    for (var i = 0; i < enemies.length; i++)
    {
        if (enemies[i].alive)
        {
            enemiesAlive++;
            game.physics.arcade.collide(shooter, enemies[i].shooter);
            game.physics.arcade.overlap(bullets, enemies[i].shooter, bulletHitEnemy, null, this);
            enemies[i].update();
        }
    }
    if (cursors.left.isDown)
    {
        shooter.angle -= 4;
    }
    else if (cursors.right.isDown)
    {
        shooter.angle += 4;
    }
    if (cursors.up.isDown)
    {
        currentSpeed = 300;
    }
    else
    {
        if (currentSpeed > 0)
        {
            currentSpeed -= 4;
        }
    }
    if (currentSpeed > 0)
    {
        game.physics.arcade.velocityFromRotation(shooter.rotation, currentSpeed, shooter.body.velocity);
    }
    land.tilePosition.x = -game.camera.x;
    land.tilePosition.y = -game.camera.y;
    shooter.rotation = game.physics.arcade.angleToPointer(shooter);

    if (game.input.activePointer.isDown)
    {
        fire();
    }
}
```

Рис 3.14 Приклад методу update з логікою гри та обробкою сигналів від гравця в Phaser

3.5 Висновок до розділу

Ми розробили три гри прототипа, використовуючи інструментальні засоби різних класів. Як наслідок можемо проаналізувати кладність реалізації на кожному з них, можливості до масштабування,ускладнення.

На Phaser розробка пройшла найшвидше, інтуїтивний та зручний інтерфейс, багато вбудованого функціоналу що реалізує найбільш вживанні можливості, прозорість під час розробки. Але складається враження, що з ростом складності гри підходи чо використовуються перетворяться на громіздкі та заплутанні конструкції. До того ж якщо доведеться розробляти щось екстра ординарне, спецефічне, що виходить за рамки загальноприйнятих ігор, то вбудованого функціоналу просто не вистачить. На мою думку цей конструктор ідеально підходить для невеликих простих ігор, що можуть створювати навіть люди без навичок програмування, до того ж це може стати прекрасним інструментом для швидкого створення макетів гри ігровими дизайнерами.

На фреймворку Phaser було написанно найбільше коду, власне всі елементи гри , так чі інакше викликались вручну. Сподобалась велика гнучкість , можливість реалізувати різноманітний функціонал. Фреймворк надає свободу до реалізації, проте стимулює структурованість коду, та полегшує взаємодію з графічним та фізичним рушіюми, менеджером ресурсів, кросплатформеністю залишаючи за користувачем лише реалізацію сутності гри. Великим мінусом стала відсутність можливості відладження коду. Отже використовуючи цей інструментальний засіб слід переважно для створення невеликих та середніх кросплатформених ігор, не залежно від жанру та специфіки. Розробка ж великих проектів може виявитися над обтяжливою.

Unity – ігровий рушій, зібравший в собі фізичний, графічний рушій, великий набір інструментів та широкий функціонал. Гарний баланс міл кількістю написаного коду та використання вбудованого інструментарію для налаштування елементів гри. Зручна візуалізація продукту та чудові можливості відладки як коду так і поведінки гри в цілому. Великі можливості

для повторного використання коду та вбудована оптимізація графіки, фізики. Unity виявився найпотужнішим засобом для розробки гри серед розглянутих, використаний для аналізу прототип виглядав надто легким для використання представленного функціоналу. Найкраще Unity покаже себе в розробці великих та середніх проектів командою чи поодиноким розробником.

4. ФУНКЦІОНАЛЬНО ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, призначеного для демонстрації можливостей створення ігор за допомогою ігрових рушіїв. Прототипи ігор були розробленні на за допомогою Unity та Construct2

Програмні продукти призначено для використання на персональних комп’ютерах під управлінням будь-якої операційної системи.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі

оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

для кожної функції визначаються повні річні витрати й кількість робочих часів.

для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки системи аналізу нелінійних нестаціонарних процесів. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу гетероскедастичних процесів в економіці та фінансах.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;
- забезпечувати високу швидкість обробки даних та відклик користувачеві у реальному часі;
- забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;
- передбачати мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір ігрового рушія;

F_3 – вибір жанру гри.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) C#;

б) JavaScript;

Функція F_2:

а) Unity

б) XNA

в) Construct2

Функція F_3:

а) Шутер;

б) Гонки;

в) Платформер;

4.3 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

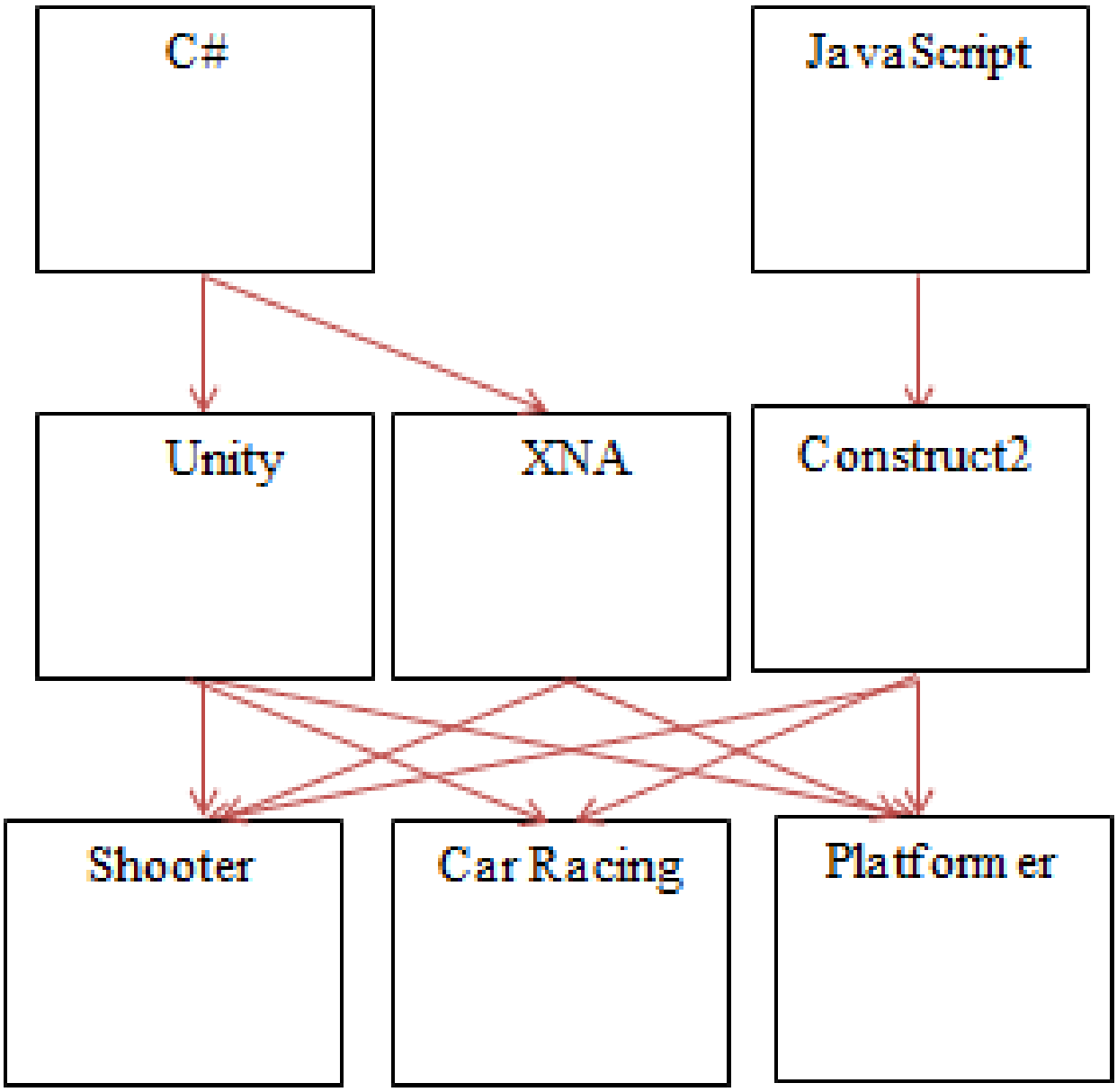

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Зручніше під час написання коду	Прив’язка до платформи .Net
	Б	Низький поріг входу для новачків, кросплатформеність	Важче відладити
F2	А	Потужний графічний та фізичний рушій, зручна відладка, конвертує код скриптів в код на мові С	Більш ресурсоемка реалізація, потребую більше написання алгоритмів
	Б	Багата база знань та гарна підтримка спільноти користувачів.	Потребує найбільше самописного коду, мало готових рішень для полегшення розробки
	В	Вбудована підтримка багатьох платформ (в браузері), багато готових рішень / напрацювань/ алгоритмів готових до використання, швидкість написання	Меньше контролю над графікою, обмеженість в складності алгоритмів
F3	*А*	Динамічна, видовищна, жанр користується особливою популярністю	Більш складні алгоритми обробки подій, більше вимог до графіки
	Б	Легка логіка, невелика кількість необхідного коду	Важко продемонструвати можливості рушія
	В	Видовищна, гарно ілюструє можливості фізичного рушія	Складна реалізація, низька популярність

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки в межах дипломної роботи ставиться демонстрація функціональних можливостей рушіїв , тому різноплановий підхід до написання гри та використання декількох мов буде плюсом. Обираєво обидва варіанти А та Б .

Функція *F2:*

Написання прототипів за допомогою рішення Б займе надто багато часу та зусиль тому, відкинемо його.

Обираєво варіанти А та В .

Функція *F3:*

Згідно намічених задач, ми маємо продемонструвати можливості рушіїв з допомогою демо програм, тому варіант Б відкидаємо як недостатньо показовий, а варінт В як надто складний у реалізації для демо прототипу.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а
2. F1б – F2в – F3а

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.4 Обґрунтування системи параметрів ПП

4.4.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри: На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються

основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – швидкодія мови програмування;
- *X2* – об’єм пам’яті для збереження даних;
- *X3* – час обробки даних;
- *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

X4: Відображає час, який витрачається на дії.

4.4.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Швидкодія мови програмування	X1	Оп/мс	19000	11000	2000
Об’єм пам’яті для збереження даних	X2	Мб	32	16	8
Потенційний об’єм програмного коду	X3	кількість строк коду	2000	1500	1000
Час обробки запитів користувача	X4	Мс	200	100	50

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

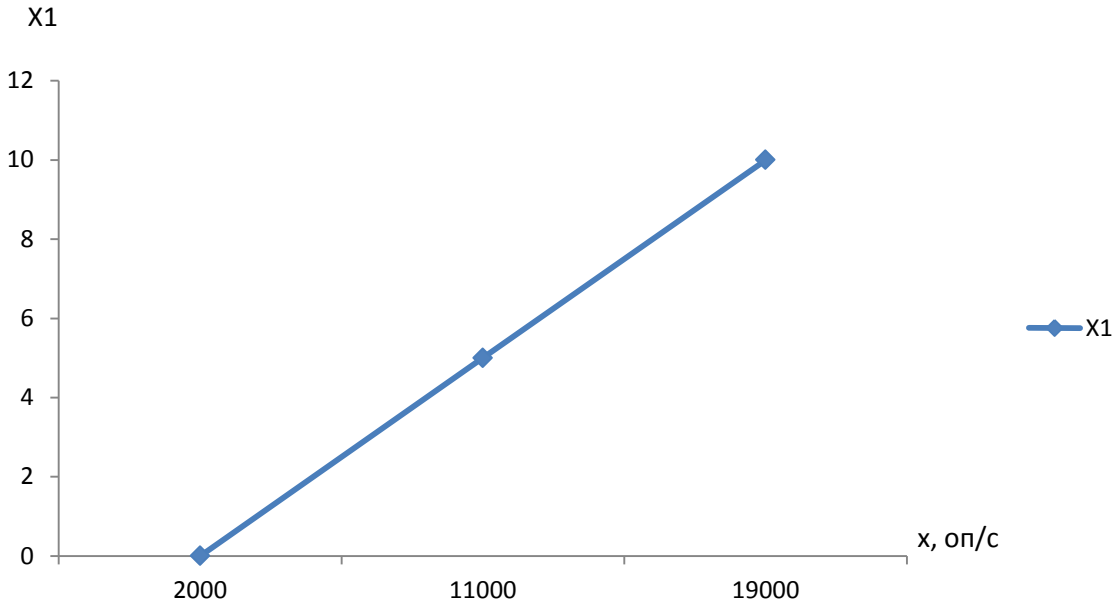

Рисунок 4.2 – X1, швидкодія мови програмування

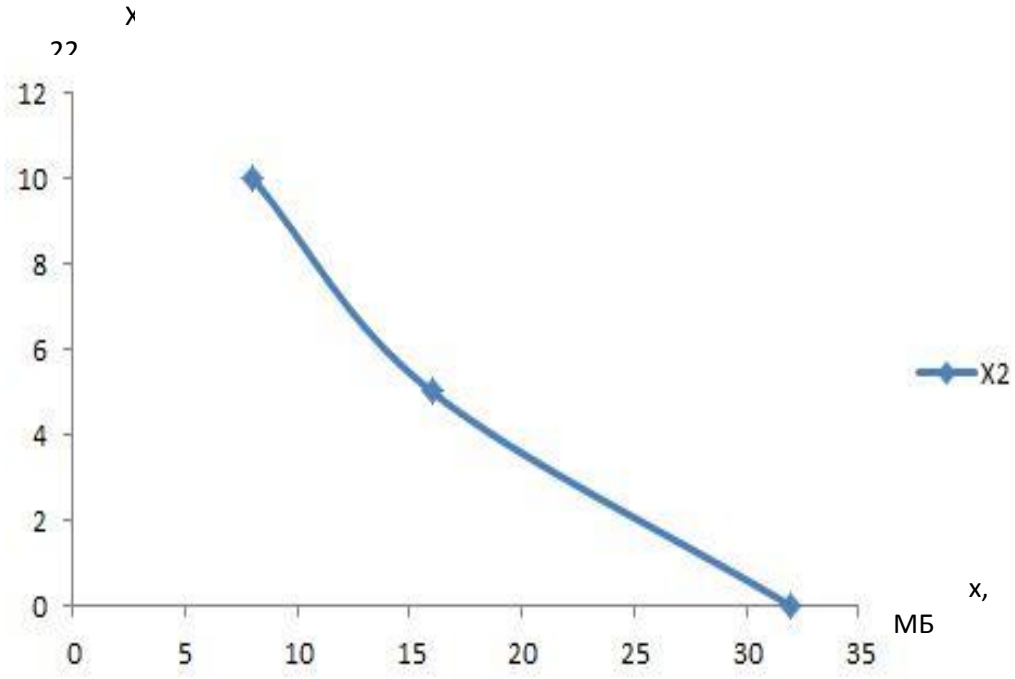

Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

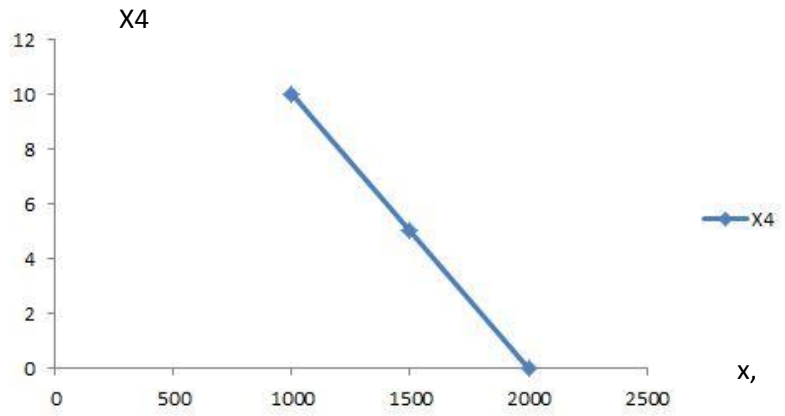

Рисунок 4.4– X3, потенційний об’єм програмного коду

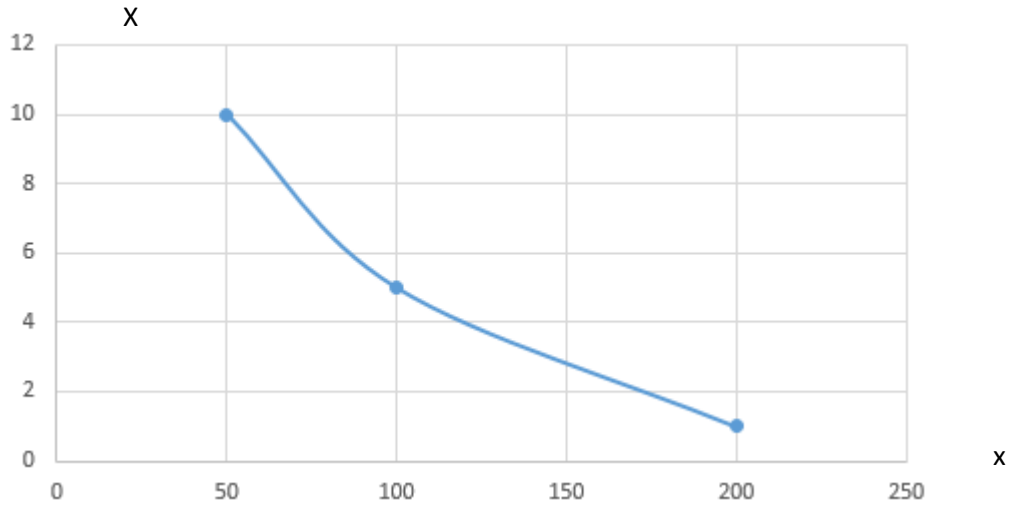

Рисунок 4.5 – X4,час обробки запитів користувача

4.4.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який має найбільш зручний інтерфейс та зрозумілу взаємодію з користувачем

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	3	2	2	1	1	2	2	13	-4,5	20,25
X2	Об’єм пам’яті для збереження даних	Мб	1	1	1	2	2	1	1	9	-8,5	72,25
X3	Час обробки запитів користувача	Мс	2	3	3	3	3	3	4	21	3,5	12,25
X4	Потенційний об’єм програмного коду	кількість строк коду	4	4	4	4	4	4	3	27	9,5	20,25
	Разом		10	10	10	10	10	10	10	70	0	195

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 195.$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 195}{7^2(4^3 - 4)} = 0{,}8 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю4.4.

Таблиця4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	>	>	<	<	<	0,5
X1 і X3	<	>	>	>	>	>	>	>	1,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	<	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметриx_i	Параметриx_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1,0	0,5	1,5	1,5	4,5	0,281	16,25	0,275	59,125	0,274
X2	1,5	1,0	1,5	1,5	5,5	0,344	21,25	0,36	77,875	0,361
X3	0,5	0,5	1,0	1,5	3,5	0,219	12,25	0,208	44,875	0,207
X4	0,5	0,5	0,5	1,0	2,5	0,156	9,25	0,157	34,125	0,158
Всього:					16	1	59	1	216	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів *X2*(об’єм пам’яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*потенційний об’єм програмного коду) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 2000 або варіанту б) 1500

Абсолютне значення параметра *X4 (*час обробки запитів користувача) обрано не найкращим (не мінімальним), тобто це значення відповідає або варіанту а) 50 мс або варіанту б) 100 мс

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j},$$

де n – кількість параметрів;$K_{\text{в}i}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри, що беруть участь у реалізації функцій	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	Б	X1	100	5	0,274	1,37
F2	А	X2	500	6,7	0,361	2,42
F3	А	X3	1500	2,5	0,158	0,395
	Б		800	6,4	0,158	1,01
	А	X4	50	7,5	0,207	1,55
	Б		100	5,5	0,207	1,14

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

$K_{K1} = 1{,}37 + 2{,}42 + 1{,}55 + 0{,}395 = 5{,}725$

$K_{K2} = 1{,}37 + 2{,}42 + 1{,}14 + 1{,}01 = 5{,}95$

Як видно з розрахунків, кращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.1 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (5.1)$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт;$K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування;$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4$ людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44$ людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$T_I = (122.4 + 19.44 + 4.8 + 19.44) \cdot 8 = 1328{,}64$ людино-годин;

$T_{II} = (122.4 + 19.44 + 6.91 + 19.44) \cdot 8 = 1345.52$ людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці бере участь один програміст з окладом 6000 грн., один фінансовий аналітик з окладом 9000грн. Визначимо зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_ч = \frac{6000 + 9000}{2 \cdot 21 \cdot 8} = 44{,}64 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_ч \cdot T_i \cdot К_Д,$$

де $С_ч$– величина погодинної оплати праці програміста;T_i – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 44{,}64 \cdot 1328.64 \cdot 1.2 = 71172{,}59$ грн.

II. $С_{ЗП} = 44{,}64 \cdot 1345.52 \cdot 1.2 = 72076{,}82$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 71172,59 \cdot 0.22 = 24429$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.3677 = 67281,38 \cdot 0.22 = 24739$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 6000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 6000 \cdot 0,2 = 16800$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 16800 \cdot (1 + 0.2) = 20280$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 20280 \cdot 0,22 = 4461,6$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 8000 = 2300$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 8000 \cdot 0.05 = 460$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 0,9733 \cdot 2,0218 = 523.83$ грн.,

де N_C – середньо-споживча потужність приладу; $K_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 8000 \cdot 0,67 = 5360$ грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС} = 17280 + 6353,86 + 2300 + 460 + 523.83 + 5360 = 33883,55$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г} = С_{ЕКС} / Т_{ЕФ} = 33883,55/1706,4 = 20,68$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т$$

I. $С_М = 20,68 \cdot 1328,64 = 27476$ грн.;

II. $С_М = 20,68 \cdot 1345.52 = 27825$ грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н = 71172,59 \cdot 0,67 = 47685,63$ грн.;

II. $С_Н = 72076,82 \cdot 0,67 = 48291,47$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

I. $С_{ПП} = 71172,59 + 4461,6 + 27476 + 47685,63 = 150795,82$ грн.;

II. $С_{ПП} = 72076,82 + 4461,6 + 27825 + 48291,47 = 152654,89$ грн.;

4.6 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 5,325 / 150795,82 = 0,35 \cdot 10^{-4}$;

$К_{ТЕР2} = 6,35 / 152654,89 = 0,42 \cdot 10^{-4}$;

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}= 0,42\cdot 10^{-4}$.

4.7 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості КТЕР = 0,42· 10-4.

Цей варіант реалізації програмного продукту має такі параметри:

Мова програмування JavaScript;

Construct2 як ігровий рушій;

Жанр гри – Шутер(Shoter).

Отже обраний варіант створення демонстраційного прототипу найкраще підійде для ілюстрації можливостей сучасних ігрових рушіїв.

ВИСНОВКИ

В ході даної дипломної роботи, в першому розділі було досліджено розвиток індустрії комп’ютерноих ігор. Були освітленні основні принципи та процеси при розробці мультимедійних ігор. Розглядались дві головні течії в індустрії ігор – комерційна та інді розробка. Також була сформулювана актуальність роботи та визначенні задачі.

В другому розділі було розглянуто найбільш популярні сучасні засоби для розробки ігор. Проведено розбиття їх на класи та проаналізовано їх базові заявлені функціональні можливості. Для більш детального аналізу було обранно оберемо по одному найоптимальнішому представнику з кожного класу для розробки навчального проекту прототипу гри.

Серед ігрових рушіїв у колі незалежних розробників, останнім часом, найбільшою популярністю користуется Unity. Він повністю задовольняє нас за своїми базовими характеристиками до тогож має потужну підтримку товариства, що значно знижує поріг входження.

Серед конструкторів, був обраний Construct 2 , що номінально обіцяє широкий спектр функціоналу та кросплатформеність. До того ж також, має чималу популярність та підтримку товариства користувачів , незважаючи на відносну молодість проекту.

Серед фреймворків для розробки ігор за функціональними можливостями можна виділити Phaser та melonJS. MelonJS вийшов у світ раніше та має більше реалізованих проектів, проте Phaser показав свою конкурентоздатність та стрімко набирає популярності, на разі він має майже в десятеро більшу кількість відвідувань офіційної сторінки в день. Тому оберемо Phaser як більш прогресивний засіб з активнішим товариством.

У третьому розділі ми розробили головні концепції гри прототипа та реалізували їх тричі , використовуючи інструментальні засоби різних класів. .на їх базі Проаналізували складність реалізації на кожному з них, можливості до масштабування, сферу використання.

На Phaser розробка пройшла найшвидше, інтуїтивний та зручний інтерфейс, багато вбудованого функціоналу що реалізує найбільш вживанні можливості. Проте з ростом складності гри підходи чо використовуються перетворюються на громіздкі та заплутанні конструкції. Є ризик , що при виході за рабки звичайних жанрів ігор вбудованого функціоналу просто не вистачить. Цей конструктор ідеально підходить для невеликих простих ігор, що можуть створювати навіть люди без навичок програмування, до того ж це може стати прекрасним інструментом для швидкого створення макетів гри ігровими дизайнерами.

На фреймворку Phaser було написанно найбільше коду, власне всі елементи гри , так чі інакше викликались вручну. З боку фреймворку була представленна велика гнучкість , можливість реалізувати різноманітний функціонал. Фреймворк стимулює структурованість коду, та полегшує взаємодію з графічним та фізичним рушіюми, менеджером ресурсів, кросплатформеністю залишаючи за користувачем лише реалізацію сутності гри. Великим мінусом стала відсутність можливості відладження коду. Отже використовуючи цей інструментальний засіб слід переважно для створення невеликих та середніх кросплатформених ігор, не залежно від жанру та специфіки. Розробка ж великих проектів може виявитися над обтяжливою.

Unity – ігровий рушій, зібравший в собі фізичний, графічний рушій, великий набір інструментів та широкий функціонал. Гарний баланс міл кількістю написаного коду та використання вбудованого інструментарію для налаштування елементів гри. Зручна візуалізація продукту та чудові можливості відладки як коду так і поведінки гри в цілому. Великі можливості для повторного використання коду та вбудована оптимізація графіки, фізики. Unity виявився найпотужнішим засобом для розробки гри серед розглянутих, використаний для аналізу прототип виглядав надто легким для використання представленного функціоналу. Найкраще Unity покаже себе в розробці великих та середніх проектів командою чи поодиноким розробником.

ПЕРЕЛІК ПОСИЛАНЬ:

1. Офіційний сайт Construct2.- www.scirra.com. – Дата доступу : 27.04.2016.
2. Офіційний портал розробників XNA - https://msdn.microsoft.com/ru-RU/games-development-msdn – Дата доступу : 27.04.2016.
3. Офіційний сайт MONO game - http://www.monogame.net – Дата доступу : 27.04.2016.
4. Офіційний сайт Phaser - Режим доступу http://phaser.io/ – Дата доступу : 27.04.2016.
5. Офіційний githab репозиторій Pixijs - Режим доступу https://github.com/pixijs/pixi.js – Дата доступу : 27.04.2016.
6. Офіційний сайт Blender - Режим доступу https://www.blender.org/features/ – Дата доступу : 27.04.2016.
7. Офіційний сайт easeljs - Режим доступу http://createjs.com/easeljs – Дата доступу : 27.04.2016.
8. Портал розробників Html5 ігор - Режим доступу https://html5gameengine.com/ – Дата доступу : 27.04.2016.
9. Офіційний сайт Melonjs - Режим доступу http://melonjs.org/ – Дата доступу : 27.04.2016.
10. Офіційний сайт PandaJs - Режим доступу http://www.pandajs.net/ – Дата доступу : 27.04.2016.
11. Офіційний сайт GameMaker - Режим доступу http://www.yoyogames.com/gamemaker – Дата доступу : 27.04.2016.
12. Офіційний сайт Unity - Режим доступу http://unity3d.com/ru/ – Дата доступу : 27.04.2016.
13. Портал ігрових новин - Режим доступу http://www.3dnews.ru/games/622071 – Дата доступу : 27.04.2016.
14. Портал ігрових новин - Режим доступу http://www.sk-gaming.com/content/4811 – Дата доступу : 27.04.2016.

15. Аналіз популярності веб сайтів - Режим доступу http://compare.easycounter.com/ – Дата доступу : 27.04.2016.
16. Популярний портал розробників ігор- Режим доступу http://www.gamedev.net/ – Дата доступу : 27.04.2016