

**Міністерство освіти і науки України**  
**Дніпровський національний університет імені Олеся Гончара**  
**Факультет прикладної математики**  
**Кафедра комп'ютерних технологій**

**Дипломна робота  
за рівнем магістра**

на тему: «**Розробка моделей віртуальної реальності для створення середовища “Віртуальна екскурсія по факультету прикладної математики”»**

**Виконавець:** студент VI курсу, групи ПК-16м-1  
Єшошкін Данило Ігорович

**Керівник:** проф., докт. фіз.-мат. наук Гук Н.А.

Кількість балів \_\_\_\_\_

Національна шкала \_\_\_\_\_

Оцінка ECTS \_\_\_\_\_

\_\_\_\_\_ проф., докт. фіз.-мат. наук Гук Н.А.  
(підпис) (прізвище та ініціали)

\_\_\_\_\_ доц., канд. фіз.-мат. наук Зайцев В.Г.  
(підпис) (прізвище та ініціали)

\_\_\_\_\_ доц., канд. техн. наук Сердюк М. Є.  
(підпис) (прізвище та ініціали)

м. Дніпро, 2018 р.

## **РЕФЕРАТ**

**Дипломна робота: 88 с., 59 рис., джерел 49.**

**Об'єкт дослідження:** розробка веб-додатків на мовах HTML5, JavaScript, CSS з використанням технологій WebGL, ThreeJS, GLSL і алгоритмів візуалізації 3D об'єктів, тривимірного звуку, Web Audio Api, штучного інтелекту.

**Мета роботи:** розробка веб-додатку “ Віртуальна екскурсія по факультету прикладної математики ”».

**Одержані висновки та їх новизна:** реалізовано декілька моделей для веб-додатку FPM3D і 5 додаткових програм, які необхідні для його створення. Додаток встановлено на сервері компанії GitHub за адресою <http://fpm3d.github.io/>.

**Перелік ключових слів:** ВЕБ-ДОДАТОК, ВЕБ-РОЗРОБКА, THREEJS, WEBGL, JAVASCRIPT, CANVAS, GITHUB, GLSL, РЕНДЕР, СЦЕНА, КАМЕРА, МЕШ, ГЕОМЕТРИЯ, ТЕКСТУРА, МАТЕРІАЛ, ШЕЙДЕР, PCM, IKM, БУФЕР, ОБ'ЄКТ, СЛУХАЧ, SFX, SOUND FX, SOUND EFFECTS, ШТУЧНИЙ ІНТЕЛЕКТ, НЕЧІТКА ЛОГІКА, ВІДОБРАЖЕННЯ, КЛАСИФІКАЦІЯ, AI, SPRITE.

## ***AНОТАЦІЯ***

The graduation research of the 6 year student Egoshkin Danila (DNU, Applied Mathematics Department, the Computer Technology Department) working on design a rendering algorithms of 3D objects, design algorithms for 3d sound, design AI and collision detection system to create web application "Virtual Tour of Faculty of Applied Mathematics". This problem is solved by creating 5 auxiliary applications and models for web application which working on <http://fpm3d.github.io/> - GitHub server. The final application will immerse you into the 3D virtual tour of FPM.

Bibliography 49, pictures 59, pages 88.

<b>ЗМІСТ</b>	
<b>РЕФЕРАТ</b>	<b>2</b>
<b>АНОТАЦІЯ</b>	<b>3</b>
<b>ЗМІСТ</b>	<b>4</b>
<b>ВСТУП</b>	<b>6</b>
<b>ПОСТАНОВКА ЗАДАЧІ</b>	<b>7</b>
<b>ОГЛЯД ПРОБЛЕМИ</b>	<b>9</b>
<b>1. Моделі створення об'ємного звуку</b>	<b>19</b>
<b>1.1. РСМ та представлення звуку в цифровій техніці</b>	<b>19</b>
<b>1.2 Концепція аудіобуфер, об'єкт, слухач</b>	<b>22</b>
<b>1.3 Шляхи імітації об'ємного звуку</b>	<b>25</b>
<b>1.4 Основні моделі об'ємного звуку</b>	<b>27</b>
<b>1.4 Модель Гатлінга</b>	<b>30</b>
<b>2. Моделі поведінки камери</b>	<b>33</b>
<b>3. Модель штучного інтелекту NPC птиці</b>	<b>35</b>
<b>3.1. NPC</b>	<b>35</b>
<b>3.2. Системи штучного інтелекту</b>	<b>36</b>
<b>3.3. Вибір системи штучного інтелекту</b>	<b>39</b>
<b>4. Моделі двовимірних і тривимірних спрайтів</b>	<b>43</b>
<b>5. Реалізація 3D веб-додатку та моделей. Створення додаткового ПО.....</b>	<b>45</b>
<b>5.1. Реалізація моделі об'ємного звуку.....</b>	<b>45</b>
<b>5.1.1. Реалізація UML діаграм моделі об'ємного звуку .....</b>	<b>45</b>
<b>5.1.2. Реалізація ефекту Доплера .....</b>	<b>47</b>
<b>5.2. Реалізація моделі поведінки камери.....</b>	<b>49</b>
<b>5.3. Реалізація моделі штучного інтелекту NPC птиці .</b>	<b>59</b>

<i>5.3.1. Реалізація алгоритму штучного інтелекту NPC птиці .....</i>	59
<i>5.3.2. Реалізація зору NPC птиці .....</i>	64
<i>5.4 Реалізація моделі двовимірних і тривимірних спрайтів .....</i>	66
<i>6. Додаткове програмне забезпечення.....</i>	67
<i>7. Результати використання створених алгоритмів та ПО .....</i>	69
<i>7.1. Створення 3D веб-додатку .....</i>	69
<i>7.2. Завантаження 3D веб-додатку на сервер GitHub pages .....</i>	75
<b>ВИСНОВКИ</b>	89
<b>СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ</b>	91

## **ВСТУП**

В теперішній час, з постійно зростаючими обчислювальними можливостями апаратної частини комп'ютерів, складність створення якісних 2D і 3D інтерактивних додатків зростає за експоненціальним законом [1]. Однак сам процес створення інтерактивних додатків став відбуватися набагато швидше завдяки вільним ресурсам, програмному забезпечення, готовим конструкторам рівнів і сучасним обчислювальним машинам, але вимоги до додатків стали набагато вищі, а це збільшує складність процесу їх створення. З урахуванням сучасних вимог до швидкості, графічним і аудіо ефектам, анімації, якості обробки інтерактивних сцен, необхідність створення різних ефектів і інтегрування динамічних моделей поведінки системи, є невід'ємною частиною внутрішнього устрою інтерактивних додатків.

Так як 90% інформації людина отримує за допомогою органів зору, то саме 2D і 3D програми є найбільш інформаційними для людського сприйняття. Реалізація 2D і 3D інтерактивного додатку вимагає адекватного вибору веб-засобів та алгоритмів. 3D веб-додатки використовуються при розробці навчальних програм, так як наочно демонструють внутрішню будову людини при вивченні анатомії, візуалізують графіки функцій, полегшують побудову розрізів тіл складної форми. Крім того, їх можна використовувати для демонстрації проектів архітектурних споруд, дизайну автомобілів, електронних пристройів та ін..

Сьогодні, завдяки швидкому розвитку сучасної комп'ютерної техніки та веб-технологій ми можемо переглядати сцени з великою кількістю об'єктів, матеріалів, джерел світла і взаємодіяти з ними в режимі реального часу. Майже всі сучасні пристрої мають вбудовані графічні процесори і браузери, які здатні переглядати подібний 3D контент.

## ***ПОСТАНОВКА ЗАДАЧІ***

За допомогою сучасних веб-технологій створити 3D інтерактивний веб-додаток для проведення віртуальної екскурсії по факультету прикладної математики. Цей веб-додаток має запускатися на всіх видах пристройів з підтримкою WebGL технології, тобто на комп'ютерах, смартфонах, планшетах з різними операційними системами та браузерами, отже має бути кросплатформним. Також веб-додаток повинен бути не залежний від розподільної здатності екрану пристрою, тобто відображатися достатньо коректно на будь-якому екрані.

Для покращення сприйняття об'єктів, що демонструються, наближення елементів віртуальної екскурсії до реальних процесів розробити модель розповсюдження звуку, модель поведінки камери, модель руху птиці, та реалізувати обрані моделі у вигляді програмних додатків. **Прикладом подібних веб-додатків може бути: Epic Citadel - WebGL HTML 5 Benchmark – написаний за допомогою WebGL (рис.1), інтерактивний веб-додаток HelloRun – написаний за допомогою ThreeJS (рис.2), веб-додаток від компанії Google - CUBE SLAM – написаний за допомогою ThreeJS (рис.3).**

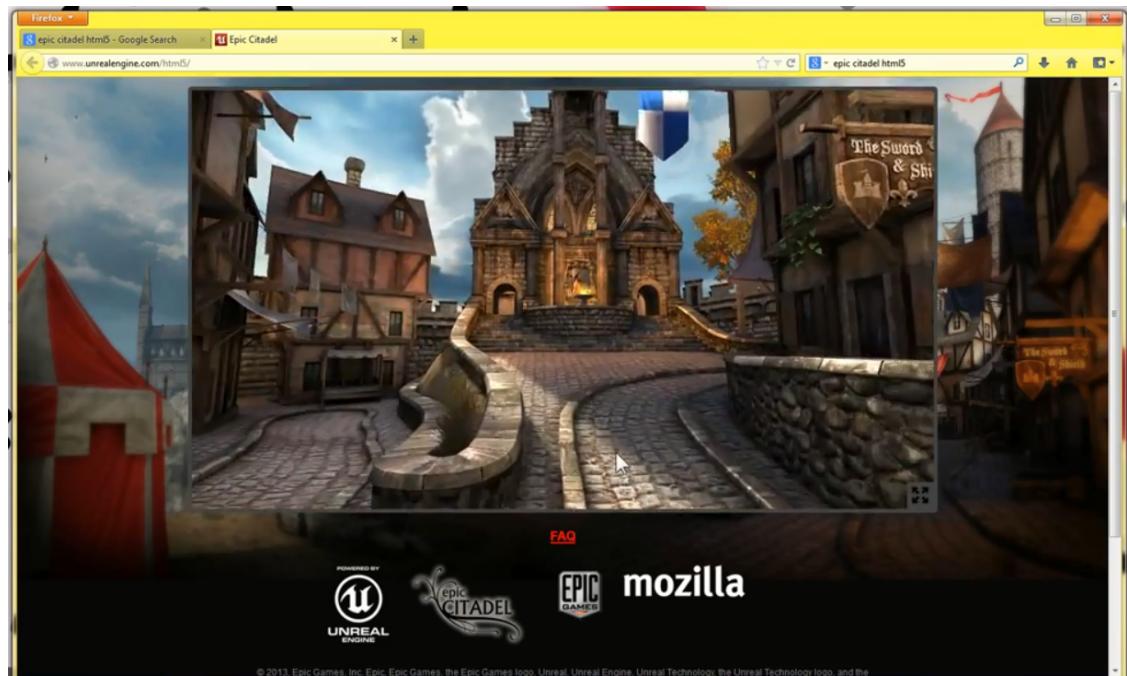


Рис. 1. Epic Citadel



Рис. 2. HelloRun



**Рис. 3. CUBE SLAM**

**Это из постановки надо убрать, потом ты это же дублируешь в огляде проблемы**

Веб-додаток має бути розміщений на безкоштовному веб-сервері з підтримкою JavaScript, з можливістю редагування коду веб-додатку у майбутньому та контролем версій проекту, що надає можливість сумісної розробки додатку між декількома розробниками, повернати окремі файли до колишнього вигляду, повернати до попереднього стану весь проект, визначати хто і коли вносив зміни у код модуля проекту.

## ***Розділ 1. Огляд проблеми створення віртуальної екскурсії***

**Сначала дать понятие, что такое виртуальная экскурсия, какие составные ее части, какие они бывают, какие цели преследуют, Потом написать, какие вообще технологии могут быть использованы для визуализации, потом сказать, что в основе лежит WebGL , а уже потом перейти к фреймворкам**

**Виртуальная реальность (VR, англ. *virtual reality*, VR, *искусственная реальность*) — созданный техническими средствами мир (объекты и субъекты), передаваемый человеку через его ощущения: зрение, слух, обоняние, осязание и другие. Виртуальная реальность имитирует как воздействие, так и реакции на воздействие. Для создания убедительного**

комплекса ощущений реальности компьютерный синтез свойств и реакций виртуальной реальности производится в [реальном времени](#).

Объекты виртуальной реальности обычно ведут себя близко к поведению аналогичных объектов материальной реальности. Пользователь может воздействовать на эти объекты в согласии с реальными [законами физики](#) (гравитация, свойства воды, столкновение с предметами, отражение и т. п.). Однако часто в развлекательных целях пользователям виртуальных миров позволяет больше, чем возможно в реальной жизни (например: летать, создавать любые предметы и т. п.)

**ВИРТУАЛЬНАЯ РЕАЛЬНОСТЬ** - модельная трехмерная (3D) окружающая среда, создаваемая компьютерными средствами и реалистично реагирующая на взаимодействие с пользователями.

Технической основой виртуальной реальности (ВР) служат технологии компьютерного моделирования и компьютерной имитации, которые в сочетании с ускоренной трехмерной визуализацией позволяют реалистично отображать на экране движение. В минимум аппаратных средств, требующихся для взаимодействия с ВР-моделью, входят монитор и указывающие устройства типа мыши или джойстика. В более изощренных системах применяются виртуальные шлемы с дисплеями (HMD), в частности шлемы со стереоскопическими очками, и устройства 3D-ввода, например, мышь с пространственно управляемым курсором или «цифровые перчатки», которые обеспечивают тактильную обратную связь с пользователем.

Основная особенность ВР-модели – это создаваемая для пользователя иллюзия его присутствия в смоделированной компьютером среде, которое называют дистанционным присутствием. Ощущение дистанционного присутствия в меньшей степени зависит от того, насколько естественно выглядят изображения среды, чем от того, как реалистично воспроизводятся движения и насколько убедительно ВР-модель реагирует при взаимодействии с пользователем. В некоторых из ВР-моделей пользователи воспринимают изменяющуюся перспективу и видят объекты с разных точек наблюдения, как если бы они перемещались внутри модели. Если пользователь располагает более чувствительными (погруженными) устройствами ввода, например, такими, как цифровые перчатки и виртуальные шлемы, то модель обеспечивается достаточным количеством данных, чтобы надлежащим образом реагировать на такие действия пользователя, как поворот головы или даже движение глаз.

Термин «виртуальная реальность» был введен в обращение в середине 1980-х годов Дж.Ланьеом – музыкантом, специалистом по компьютерной технике и предпринимателем, фирма которого « V PL Рисерч» разработала первую цифровую перчатку для управления ВР-взаимодействием, а также средства для построения ВР-моделей.

## Компьютерное моделирование и имитация

Визуализация трехмерной графики обеспечивает возможность просмотра большинства имитаций ВР; при этом взаимодействие пользователя с окружающей средой ВР базируется на компьютерном моделировании. В компьютерных моделях объекты наделяются определяющими их свойствами, которые задают их реакции на различные виды манипуляций. Компьютерные модели могут использоваться для исследования процессов без построения системы, в которой они реально происходят. Такие модели позволяют ускорить процессы (например, для определения эксплуатационного ресурса какого-либо нового изделия) или замедлить их (чтобы легче было наблюдать, например, движение пули или ракеты). Построение таких компьютерных моделей более сложно, а их эффективность зависит от точности используемых формул, описывающих зависимости всех переменных конкретного исследуемого процесса.

**Вот пример чего-то подобного, только текст надо адаптировать**

Для створення подібних проектів доцільно замість чистого WebGL використовувати фреймворки, які подібні 3D рушіям [24]. Деякі компанії, надають наступне рішення подібного роду, всі вони базуються на HTML5, JavaScript та WebGL: Unreal Engine 4 [7] на якому було реалізовано Epic Citadel - WebGL HTML 5 Benchmark (рис.4), ThreeJS [25] - безкоштовна легка кросбраузерна бібліотека написана на JavaScript, ця бібліотека використовується для створення та відображення анімованої комп'ютерної 3D графіки при розробці 3D веб-додатків (рис.5), Google Chrome Experiments – фреймворк від компанії Google, зараз знаходиться в розробці, проте на даний момент існують тестові версії застосування цієї технології (рис.6), Blend4Web [26] - фреймворк, призначений для створення і відображення інтерактивної тривимірної графіки в браузерах (рис.7).

**Мне кажется, что это средства реализации, потому их лучше поставить в конец.**

А начать с того, что в таких экскурсиях нужен звук, движение и ..., потому основным является моделирование именно этих компонент таким образом, чтобы они приближали нас к действительности

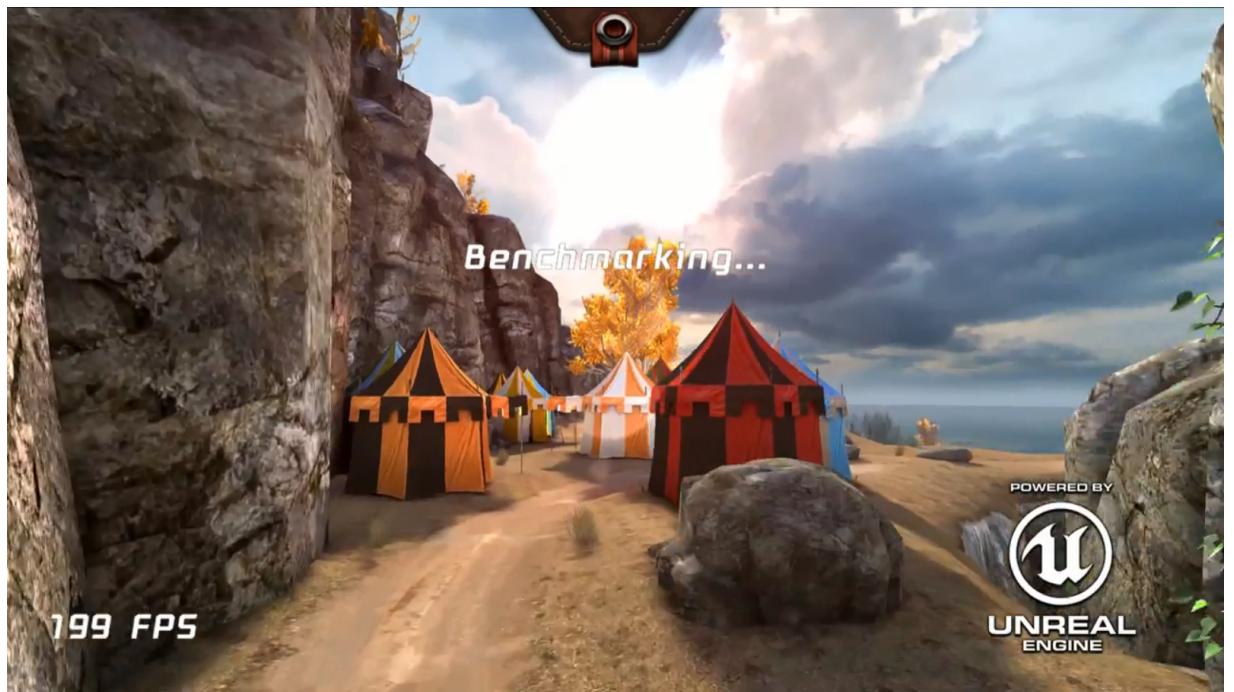


Рис. 4. Epic Citadel

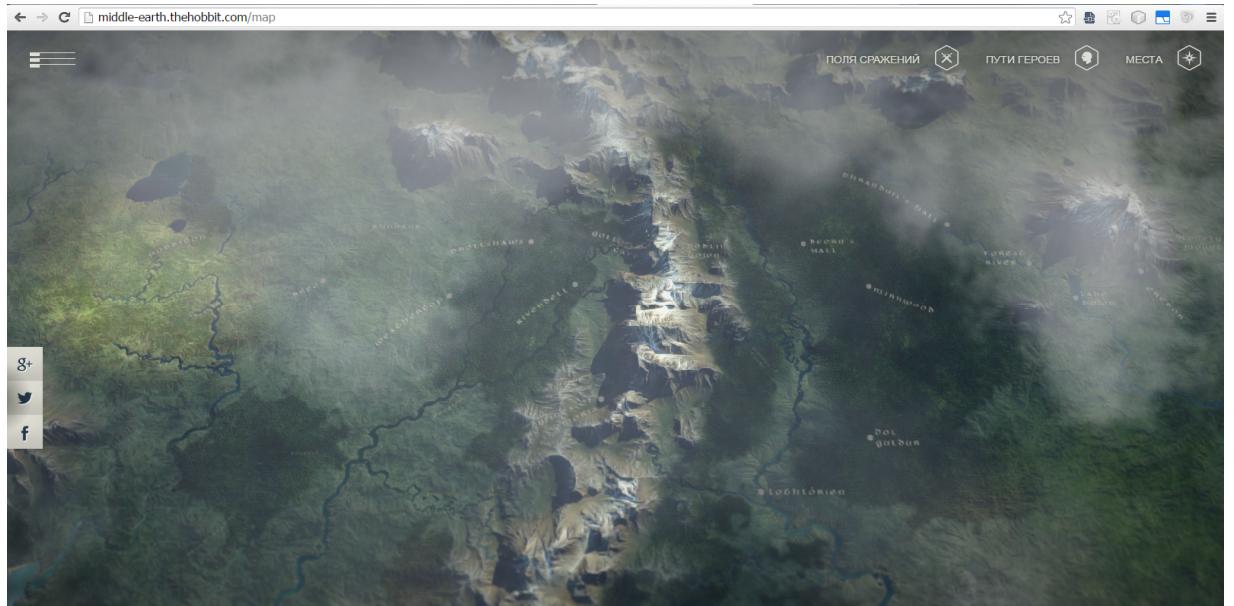
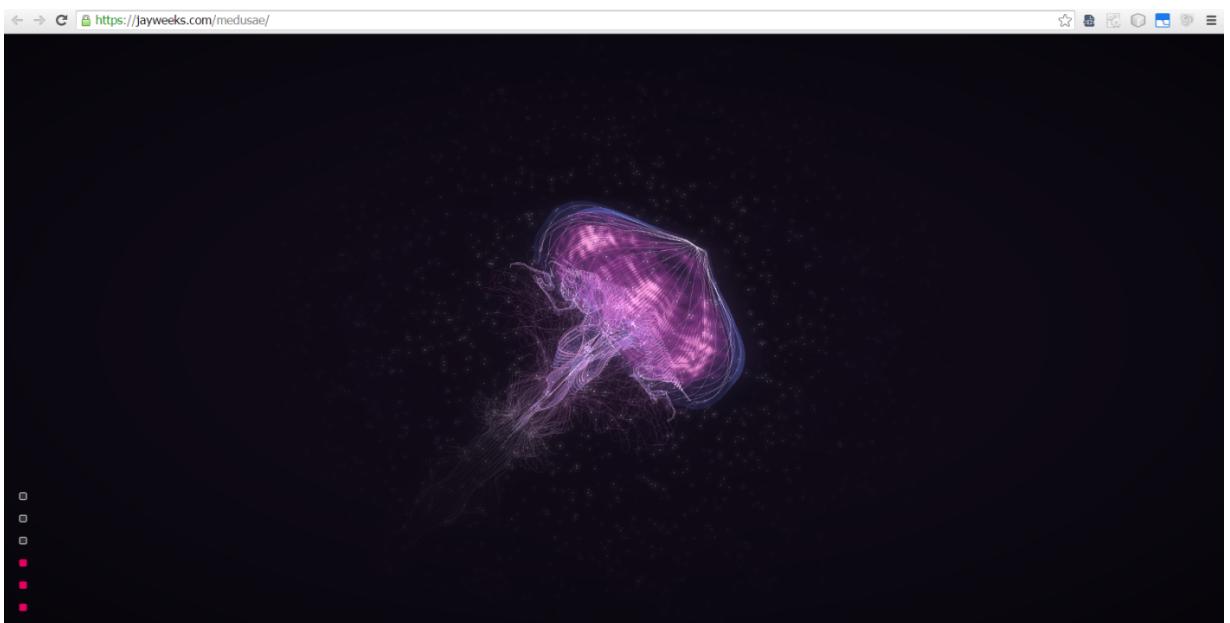


Рис. 5. Веб-додаток для 3D карты Середзем'я



**Рис. 6. 3D веб-додаток рух медузи**



**Рис. 7. Веб-додаток проектів створених на Blend4Web**

Для створення проекту подібного масштабу, буде потрібна велика кількість текстур, моделей, скриптів. Так само потрібно спеціалізоване програмне забезпечення для обробки і створення різних типів текстур (дифузна текстура, карта відблисків, карта нормалей, карта висот, карта змішування, карта оточення, карта світіння, карта прозорості, карта відображень), програмне забезпечення для побудови тривимірних моделей різного рівня деталізації. **Это все тоже детали реализации.**

А вот с этого надо начать, толькъ написать преамбулу, что моделирование именно этих компонент не обходимо привести таким образом, чтобы они приближали нас к действительности

Для створення якісних 2D і 3D інтерактивних додатків необхідні наступні компоненти [2,7,14]:

- Рушій рендеринга («визуалізатор») - проміжне програмне забезпечення, основним завданням якого, є візуалізація (рендеринг) двомірної або тривимірної комп'ютерної графіки. На даний момент розроблено безліч алгоритмів візуалізації. Існуюче програмне забезпечення може використовувати різні алгоритми для отримання кінцевого зображення. Трасування кожного променя світла в сцені непрактично і займає дуже багато часу. Внаслідок цього, було розроблено чотири групи методів, більш ефективних, ніж моделювання всіх променів світла:
  - Rasterization – Растеризація – візуалізація проводиться проектуванням об'єктів сцени на екран без розгляду ефекту перспективи щодо спостерігача;
  - Ray casting – Кидання променів – Сцена розглядається з певної точки - точки спостереження. З точки спостереження на об'єкти сцени прямують промені, за допомогою яких визначається колір пікселя на двовимірному екрані. При цьому промені припиняють своє поширення (на відміну від методу трасування та зворотного трасування), коли досягають будь-якого об'єкта сцени або її фону;
  - Ray tracing – Трасування променів – схожа на метод Ray casting – кидання променів. З точки спостереження на об'єкти сцени прямують промені, за допомогою яких визначається колір пікселя на двовимірному екрані. Але при цьому промінь не припиняє своє поширення, а розділяється на три промені-компоненти, кожен з яких вносить свій внесок в колір пікселя на двовимірному екрані: відбитий, тіньовий і заломлений. Кількість таких компонентів визначає глибину трасування і впливає на якість і фотorealістичність зображення.
  - Path tracing – Трасування шляху – є узагальненням традиційного методу Ray tracing – трасування променів, алгоритм який трасує

промені в напрямку від віртуальної камери крізь простір, промінь «відскакує» від предметів до тих пір, поки повністю не розсіється.

- Фізичний рушій - частина системи, яка виробляє комп'ютерне моделювання фізичних законів реального світу в віртуальному світі, з тим або іншим ступенем апроксимації. Так само застосовується як детектор колізій, процедурний генератор анімації. Фізичний движок необхідний для симулювання наступних процесів:
  - динаміка абсолютно твердого тіла;
  - динаміка тіла, що деформується;
  - динаміка рідин;
  - динаміка газів;
  - поведінку тканин;
  - поведінку мотузок (троси, канати та інші);
  - анімація потоків часток;
  - фізика ragdoll;
  - підтримка IK - inverse kinematics;
  - physically-bound;
  - dynamic motion synthesis;
- Звуковий движок - програмний компонент, що відповідає за відтворення звуку (шумове та музичне оформлення, голоси персонажів, генерування звукових коливання) в комп'ютерній грі або іншому додатку. Звуковий движок також часто відповідає за імітацію певних акустичних умов і створення спеціальних ефектів, тобто за SFX - SoundFX - Sound Effects, відтворення звуку відповідно до місця розташування, відлуння, фленжер, зворотнє відлуння, фейзер, ревербератор, сустейн та інші.
- Система скриптів - програмний компонент, який є інтерпретатором мови високого рівня, дозволяє автоматизувати часто повторювані процеси, спрощує структуру функцій та коду підпрограм, зменшує

ризики створення низькорівневих помилок, так як користувач, має доступ тільки до доступних інтерфейсів, також прискорює процес розробки і налагодження додатків.

- Анімація - програмний компонент. Даний компонент повинен виконувати наступні функції:
  - Підтримка та завантаження різних форматів анімації;
  - Відтворення і зупинка анімації;
  - Зациклення анімації;
  - Морфинг (morphing) - інструмент для створення більш плавною анімації між ключовими кадрами послідовності анімації. Дозволяє різними видами апроксимації наблизити точки (кадри) один до одного, тим самим відновити або створити плавну анімацію;
  - Мікшування, змішування і додавання чи так же Merging & Blending анімації - цей процес є необхідною функцією для забезпечення персонажа плавними анімаціями. Маємо окремі анімації, наприклад, цикл ходьби, цикл бігу, анімація в стані спокою або дії. У будь-який момент, повинна бути можливість переходу з анімації А - анімації спокою в анімацію Б - цикл ходьби і назад. Для досягнення плавних переходів анімації і без раптових ривків в русі. А так же для складання анімацій, це дозволяє створювати нові види анімації шляхом складання декількох існуючих. Тобто складання анімації А - махання рукою і анімації Б - цикл ходьби, дозволяє створити новий вид анімації;
  - Створення процедурної анімації - генерування анімації за допомогою внутрішнього функціоналу або з використанням фізичного движка. Технології: ragdoll, physically-bound, dynamic motion synthesis;
- Штучний інтелект – частина системи, яка здійснює контроль усіх динамічних елементів, що є частиною штучного інтелекту, тобто таких,

що володіють інтелектом і підпорядковується правилам логіки системи.

- Мережевий код – частина функціоналу ПЗ, яка забезпечує з'єднання, обмін інформацією по мережі між комп'ютерами користувачів та контроль її цілісності. Виробляє захист, шифровку інформації [15].
- Управління пам'яттю – програмний компонент, який відстежує всі об'єкти і покажчики на наявність витоку або переповнення пам'яті, а так само забезпечує збір вже не використованої інформації або інформації, покажчики на яку не належать жодному об'єкту для подальшого її очищення. Використовується не тільки для очищення пам'яті, а й для раціонального її збору.
- Багатопоточність – частина системи, яка дозволяє процесу складатися з декількох потоків, що виконуються «паралельно», тобто без запропонованого порядку в часі. При виконанні деяких завдань такий поділ може досягти більш ефективного використання ресурсів обчислювальної машини.

**У дипломній роботі бакалавра, було зроблено часткову реалізацію кількох цих компонентів, але для створення якісних 2D і 3D інтерактивних додатків – цього не достатньо.**

**Это не надо**

Для створення реалістичних ефектів, що дозволять перенести користувача в «наш світ» і створити атмосферу повного занурення, знадобляться наступні компоненти: звуковий движок, анімація, штучний інтелект. У попередній роботі саме ці компоненти не були розглянуті і розроблені. Отже для веб-додатку не вистачає: звукового движка - а саме, моделей об'ємного звуку; анімації - тобто, моделі поведінки камери; та штучного інтелекту для створення NPC.



## ***Розділ 2. Моделювання процесів та об'єктів віртуальної реальності***

### ***2.1. Моделювання об'ємного звуку***

Коротко описать, само понятие звука, с помощью чего звук моделируется, аналоговий и цифровой сигнал, что не обходимо преобразовывать.

Текст диплома должен быть написан, как сочинение на тему. Тот, кто будет читать с темой не знаком, потому все должно быть последовательно и четко описано

### ***2.1. PCM та представлення звуку в цифровій техніці***

Для представлення звуку в цифровій техніці використовується PCM - pulse code modulation або ж ICM - імпульсно-кодова модуляція - використовується для оцифровки аналогових сигналів. Практично всі види аналогових даних (відео, аудіо (голос, музика), телеметрія) допускають застосування ICM [6, 8, 12].

Для створення послідовність PCM з аналогового сигналу необхідно перетворення аналогового сигналу в цифровий для цього використовується аналого-цифровий перетворювач (АЦП). АЦП через рівні проміжки часу вимірює амплітуду аналогового сигналу - отримує миттєві значення або відліки сигналу, потім перетворює відліки в двійкові слова [2].

Таким чином, (далее ты описываешь, что получается, с чем ты работаешь)

Мгновенное измеренное значение (отсчёт) аналогового сигнала квантуется по уровням (округляется от ближайшего целого). Число уровней квантования, обычно, равно или кратно целой степени числа 2, например,  $2^3 = 8$ ,  $2^4 = 16$ ,  $2^5 = 32$  и т. д. Номер уровня кодируется двоичными словами длиной 3, 4, 5 и т. д. бит<sup>[2]</sup>.

Затем выходные слова АЦП в параллельном коде подвергаются кодированию при помощи передачи на регистр сдвига, тактируемый вспомогательным генератором сдвига. На выходе регистра сдвига формируются пачки<sup>[поясните]</sup> кодированных импульсов в последовательном коде. Затем пачки импульсов передаются в канал связи<sup>[2]</sup>.

Частота отсчётов сигнала (или скорость оцифровки, частота дискретизации) для исключения потерь информации в соответствии с теоремой Котельникова должна быть не меньше удвоенной максимальной частоты в спектре аналогового сигнала.

Существуют специализированные интегральные микросхемы, совмещающие АЦП, регистр сдвига, тактовые генераторы и другие устройства.

На рис. наведено Пример 4-битной (16-уровневой) ИКМ. Показано квантование аналогового сигнала и пачки импульсов, кодирующих отсчёты. Передача в канале производится старшими битами вперёд

То есть сначала надо все потребно описать, как и что моделируется и отображается, а уже потом это, что тобой была найдена ошибка. Тоже можно подобне к чему это приводило и что дало исправление этой ошибки

В ході вивчення матеріалу про імпульсно-кодову модуляцію - в даному ресурсі [6] - була знайдена помилка в графіку, а саме при переході через нуль різко змінювалася послідовність біт, дивитися рис. 1.1.1..

Послідовність РСМ по осі t - з помилкою:

{1000,1001,1011,1101,1110,1111,1111,1111,1110,1101,1100,1010,**0111,1101,**  
0101,0011,0010}.

Послідовність РСМ по осі t - помилка виправлена:

{1000,1001,1011,1101,1110,1111,1111,1111,1110,1101,1100,1010,**1001,0111,**  
0101,0011,0010}.

Дана помилка була виправлена і графік перебудований, дивитися рис. 1.1.2., інформація про це була відправлена автору, а також розміщена в профілі того, хто виправив помилку [13].

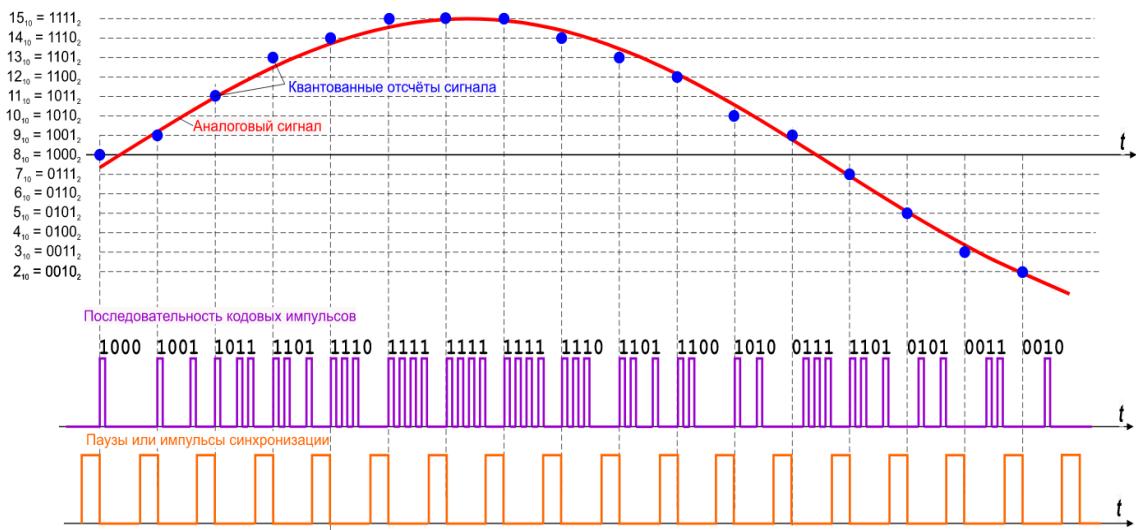


Рис. 1.1.1. Схема PCM - з помилкою

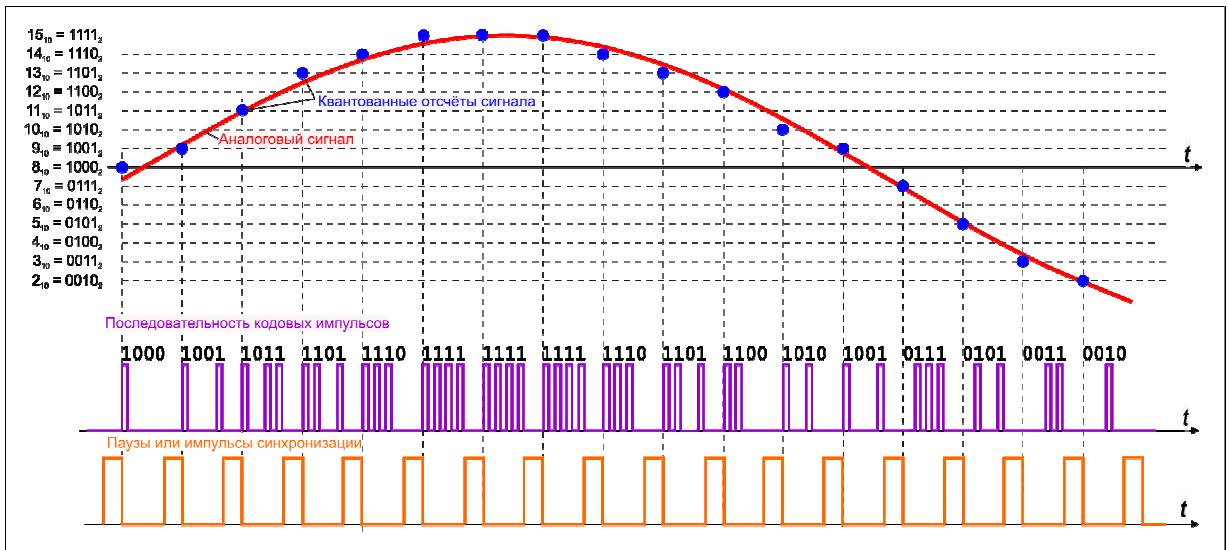


Рис. 1.1.2. Схема PCM - помилка виправлена

Далі розглянемо формати аудіофайлів. Аудіофайл (файл, який містить звукозапис) - комп'ютерний файл, що складається з інформації про амплітуду і частоту звуку, збереженої для подальшого відтворення на комп'ютері або програвачі [12]. Тобто, все аудіофайли є послідовність PCM - але, при цьому діляться за методом стиснення:

- аудіоформати без стиснення, такі як WAV, AIFF і інші;

- аудіоформати зі стисненням без втрат (APE, FLAC і інші);
- аудіоформати зі стисненням з втратами (MP3, Ogg і інші);

## ***1.2 Концепція аудіобуфер, об'єкт, слухач***

Основними елементами звукового движка, а саме концепцією 3D звуку, є: концепція аудіобуфер, об'єкт, слухач [3]. Саме ці об'єкти є ключовими елементами для створення системи відтворення звуку в тривимірному просторі.

Елементи звукового движка:

- Buffer – Буфери (аудіобуфери) – містять аудіодані в форматі PCM в 8-ми, 16-ти, 20-ти, 24-х або 32-х бітному варіанті, проте, зазвичай, число рівнів квантування, дорівнює або кратне 2 в степені n, де n - ціле число. Аудіодані також можуть містити декілька каналів аудіо – багатоканальні аудіодані: моно – 1 канал, стерео – 2 канали, а у системах багатоканального звуку 5.1 → 5+1 каналів або 7.1 → 7+1 каналів). Максимальна кількість каналів на сьогодні 255 у форматі Ogg Vorbis [19]. Буфери - використовуються для динамічного завантаження інформації з метою економії місця в оперативній пам'яті, так наприклад звуковий файл в 4 хвилини в стандарті Audio CD буде займати в пам'яті близько 40 мб. А також для динамічного перемикання між потоками даних в процесі відтворення.
- Source – Об'єкт (джерело звуку) – включає в себе покажчик на буфер, швидкість відтворення PCM послідовності з буфера, позицію у тривимірному просторі, напрямок і інтенсивність звуку. Використовується для програвання буферів і позиціонування звуку в тривимірному просторі. Для створення динамічних ефектів, швидких змін буферів і економії оперативної пам'яті, буфери додаються в стек – черга всередині об'єкта.

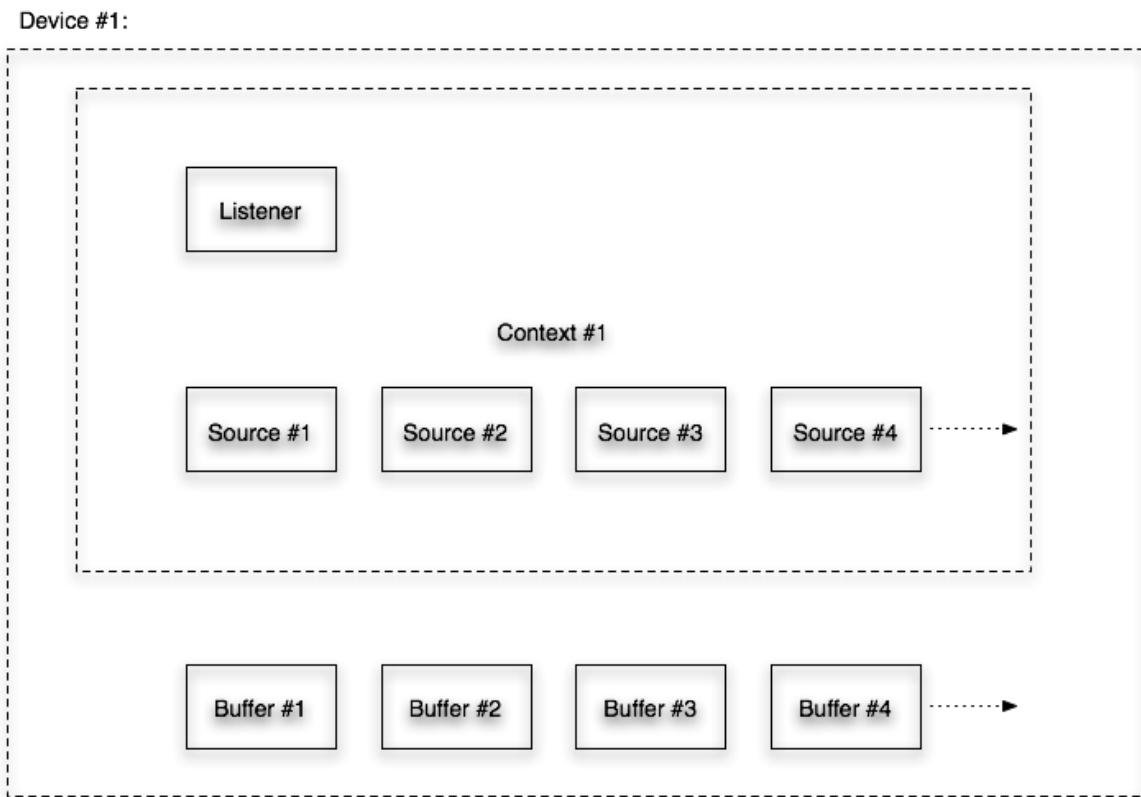
- Listener – Слухач - містить інформацію про швидкість, позицію, напрямок слухача у тривимірному просторі. Також може зберігати інформацію про загальне посилення звуку в цілому.
- Функція рендеринга звуку – проводить розрахунок звукових даних щодо стану об'єктів (джерел звуків), поточних буферів, позиції і налаштувань слухача за допомогою мікшування. Ця функція робить необхідні обчислення, такі як визначення відстані, ефекту Доплера, і так далі.
- Context – Контекст – це область створена у процесі розробки сцени у якій знаходяться слухач так джерела звуку, контекст допомагає відокремити різні сцени та різні звуки у них. Таким чином, це допомагає згрупувати, впорядкувати та розділити джерела звуку відносно сцени, та зменшити кількість необхідної оперативній пам'яті. Зміною контексту може керувати штучний інтелект програми, або сам користувач.
- Device – Пристрій виводу звуку – це відтворювач звуку на системі чи пристрой, тобто це може буди: звукова карта, драйвер, бібліотека, система, інший периферійний пристрій, та інші. Більшість пристрій виводу звуку використовуються як DAC – Digital-to-Analog Converter або ЦАП – Цифро-аналоговий перетворювач. Його основною метою є приймання PCM послідовності та відтворення її за допомогою акустичні системи чи іншого аналогового пристрою.

При цьому усі елементи звукового движка знаходяться у певній залежності один від одного, а саме:

- 1) При ініціалізації системи відтворення звуку, принаймні один пристрій повинен бути відкритий, в іншому випадку система не має пристрою для відтворення звуку, а отже і відтворити звук на цій системі не можливо.

- 2) У цьому пристрой буде створено принаймні один контекст.
- 3) У цьому контексті передбачається принаймні один об'єкт слухача, але не більше 4 (проте, це залежить від реалізації), і може бути створено безліч об'єктів – джерел звуку, чи жодного.
- 4) Кожне джерело звуку може мати один або більше об'єктів типу буфер, які приєднуються до джерела.
- 5) Буферні об'єкти не є частиною певного контексту - вони діляться між усіма контекстами на одному пристрій.

Таким чином повну концепцію 3D звуку можна представити як схему взаємодії об'єктів звукового движка, дивитися рис. 1.2.1..



**Рис. 1.2.1. Основні елементи звукового движка**

### ***1.3 Шляхи імітації об'ємного звуку***

На даний момент існує кілька шляхів для створення та імітації об'ємного звуку:

- Багатоканальна стереофонія (в тому числі системи Surround Sound) – найпростіший метод для створення об'ємного звуку. Для його реалізації необхідне використання мікрофонних систем для просторової звукозапису та подальше зведення об'ємного звуку за допомогою мікшерів або спеціалізованого програмного забезпечення для систем гучномовців, що оточують слухача при відтворенні звуку з різних сторін.
- Бінауральна стереофонія – перетворення звуку з урахуванням психоакустичних методів локалізації звуку для моделювання двовимірного звукового поля. У цьому випадку кількість гучномовців та тип акустичної системи користувача не мають значення, про те у класичному випадку, необхідно лише два гучномовці.
- Системи синтезу звукового поля навколо слухача – ця технологія, заснована на принципі Гюйгенса, являє собою спробу відновити записане звукове поле в просторі приміщення слухача, в формі так званої «голофонії»:
  - Wave field synthesis (WFS) – відтворює віртуальне акустичне поле за рахунок акустичного фронту, створеного системою розподілених на поверхні гучномовців. Комерційні системи WFS, представлені в даний час на ринку компаніями Sonic emotion і Iosono, вимагають велику кількість гучномовців і значні обчислювальні потужності.
  - Амбіофоніческі системи – Ambisonics – також засновані на принципі Гюйгенса, дозволяють отримати точне відтворення

звуку в центральній точці, але менш точний при видаленні від центру.

- П'ятиканальне поле - просторове звучання досягається зміною рівнів з стереофонічного джерела звуку з використанням цифрової обробки сигналу, аналізуючи стерео запис на предмет положення окремих звуків в панорамі, потім зміщуючи їх, відповідно, в п'ятиканальне поле. Однак, кращих результатів можна досягти, якщо таке перетворення робиться з квадрофонічного запису. Основним інструментом у даному методі є спектральний аналіз звуку.

Слід врахувати, що багатоканальна стереофонія та системи синтезу звукового поля навколо слухача - вимагають наявності спеціального обладнання для звукозапису, а так само обробки і зведення звукових доріжок. Окрім цього для відтворення подібних композицій знадобиться наявність професійних гучномовців, ЦАП (Цифро-аналоговий перетворювач) та АЦП (аналого-цифровий перетворювач). Також виходячи з вимог дипломної роботи, ведеться розробка веб-додатку, тому необхідно максимально зменшити навантаження та використання процесорного часу і ресурсів оперативної пам'яті. Також необхідно врахувати той факт, що більшість приладів мають тільки два динаміка, а тому для створення та імітації об'ємного звуку слід вибрати бінауральну стереофонію.

Далі розглянемо моделі, що використовуються для імітації бінауральної стереофонії.

## **1.4 Основні моделі об'ємного звуку**

Данні моделі вивчаюся у фізиці у розділі акустика. Опис моделей та сприйняття цих моделей людиною описується у напрямку психоакустики.

Психодинаміка - наукова дисципліна, що вивчає психологічні і фізіологічні особливості сприйняття звуку людиною.

В аспекті суто звуковому, основними завданнями психоакустики є наступні:

- зрозуміти, як система слухового сприйняття людини розшифровує той чи інший звуковий образ;
- встановити основні відповідності між фізичними стимулами і слуховими відчуттями;
- виявити, які саме параметри звукового сигналу є найбільш значущими для передачі семантичної (смислової) і естетичної (емоційної) інформації.

Розглянемо основні моделі для створення об'ємного звуку:

- Ефект Доплера - зміна частоти і, відповідно, довжини хвилі випромінювання, сприймається спостерігачем (приймачем), внаслідок руху джерела випромінювання і / або руху спостерігача (приймача) [4, 5]. Даний ефект можна спостерігати, коли проїжджає поїзд, видаючи звуковий сигнал. У разі, коли поїзд буде наблизатися, висота частоти звуку буде вище, потім, коли поїзд порівняється з вами, частота звуку різко знизиться і далі, при видаленні, звук сигналу поїзда буде звучати на більш низькій частоті і поступово затухати.
- Head-related transfer function (HRTF) - функція характеризує як вухо отримує звук від джерела звуку в просторі. Оскільки звук, який сприймається слухачем, залежить від розміру і форми голови, вух,

вушного каналу, щільності голови, розміру і форми носових і ротових порожнин, все це трансформує звук і впливають на те, як він сприймається, підвищуючи деякі частоти і послаблюючи інші. Формально це реалізація бінаурального ефекту.

- Ефект реверберації – це процес багаторазового відображення звукового сигналу від різних поверхонь з поступовим зменшенням його інтенсивності. У реверберації існує ряд параметрів – час раннього відображення, час пізнього відображення, швидкість загасання, процентне співвідношення «сухого» сигналу з обробленим. Ці параметри вказують на розмір приміщення і місце джерела звуку в ньому відносно слухача. Інколи використовуються конволюційні (згорткові) ревербератори, до яких застосовуються імпульси реальних приміщень. Де сам імпульс являє собою шумовий «зліпок» приміщення (звуковий файл), який модулює вихідний звуковий файл, тим самим імітуючи простір.
- Бінауральний ефект - ефект, що виникає при сприйнятті звуку двома вухами. Він дозволяє визначити напрямок на джерело звуку, що робить звукове сприйняття об'ємним. Ефект полягає в тому, що якщо людина обернена обличчям до джерела звуку, то звукова хвиля доходить до обох його вух одночасно - в одній фазі. Коли людина повертає голову, скажімо, вліво від напрямку на джерело звуку, то його правого вуха звукова хвиля досягає раніше, ніж лівого, і фази звукових коливань в вухах виявляються зсунутими один щодо одного. З цього зсуву фаз мозок і визначає напрямок на джерело звуку [9,10,11].

У даній роботі будемо використовувати концепції 3D звуку описані вище і ефект Доплера - для створення ефекту об'ємного 3D звуку. Так як він менш вимогливий до обчислювальних ресурсів, що дозволить веб-додатку працювати швидше і бути менш вимогливим до апаратної частини пристройів,

це в свою чергу розширити підтримку веб-додатку у різних ОС, браузерах та пристроях.

Відомі роботи в області розробки моделей об'ємного звуку належать: Garin Hiebert, Keith Charley, Peter Harrison, Jean-Marc Jot, Daniel Peacock, Jean-Michel Trivi, Carlo Vogelsang, Р. Бессон, Г.С. Ландсберг і ін. [3, 10, 11]. Великий внесок у розвиток ідей об'ємного 3D звуку вносять корпорації Creative Technology, «Dolby Laboratories, Inc», THX - Tomlinson Holman's crossover, Lucasfilm, LucasArts, The Walt Disney Company, Un4seen Developments, Sega Corporation, Croteam, Firelight Technologies, Valve Corporation, Blender Foundation, Razer, 4A Games, Bugbear Entertainment, id Software, Ubisoft, EA Digital Illusions Creative Entertainment, Epic Games, Sony Interactive Entertainment, Sony Corporation, GSC Game World, Bethesda Softworks LLC, The OGRE Team, Bohemia Interactive Studio, CD Projekt, Team17 Software, UNIGINE Corporation, Unity Technologies та інші.

## ***1.4 Модель Гатлінга***

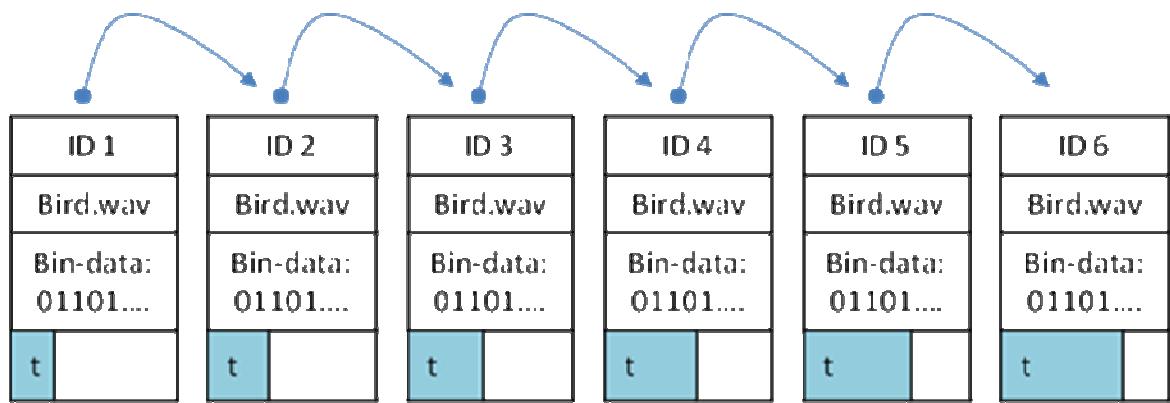
У разі відтворення великої кількості звукових файлів, необхідно врахувати можливість економії і контролю кількості виділеної оперативної пам'яті. Розглянемо дану проблему на наступному прикладі:

У випадку з величезною кількістю звуків які відтворюються одночасно і їх динамічному збільшенні користувачем. Припустимо, один звук з відлунням триває 5 секунд, а інтервал між звуками може бути 0.5 секунд. Якщо просто запускати файл спочатку, то попередній звук різко обрветься. Якщо клонувати елемент звуку кожного разу перед запуском, це може привести до витоку пам'яті, так як минулий звук ще програється і зберігається в пам'яті. Тому треба розробити модель, що зможе обмежити кількість одночасно відтворених звукових потоків – буферів.

Для цього скористаємося схемою Гатлінга. Ідея базується на створенні для кожного джерела звуку окремої структурі всередині якої зберігатися черга одночасно відтворюваних буферів. Слід врахувати те, що черга обмежена певною кількістю елементів в ній. У схемі Гатлінга, ця кількість дорівнює 6-ти. Людина з ідеальним, професійним слухом може ідентифікувати більш ніж 50 звуків, що грають одночасно, наприклад диригент оркестру може ідентифікувати кожен зі 100 інструменті та його партію. Проте, в даному випадку розглядається одне джерело звуку та його буфери, а не декілька джерел звуку. Тому 6-ти одночасно відтворюваних буферів, буде достатньо, проте для більшої реалістичності, цей показчик можна збільшити. Тому у реалізації моделі слід зробити цей показчик змінним, але за замовчуванням рівним 6-ти.

У разі якщо користувач відтворить одне джерело звуку послідовно 6 разів, буде відбуватися відтворення 6-ти буферів одночасно, з подальшим їх видаленням з черги, відносно початку відтворення і тривалості. Робота

моделі Гатлінг для даного випадку представлена на схемі, дивитися рис. 1.4.1.



**Рис. 1.4.1. Схема роботи моделі Гатлінга**

В даній схемі:

- ID – ідентифікатор буфер – унікальний номер відтвореного буфера. Ідентифікатор який йому присвоїла система або модель.
- Bird.wav – назва файлу на який вказує буфер.
- Bin-data – двійковий (бінарний) файл в широкому сенсі: файл, що містить послідовність довільних байтів (Саме ці данні зчитав буфер з файла Bird.wav).
- t – тривалість відтворення файла.

Як бачимо на схемі, дивитися рис. 1.4.1., буфер з ID 6 майже закінчився і скоро буде видалений з черги. А буфер з ID 1 – тільки нещодавно розпочався.

У разі якщо користувач відтворить одне джерело звуку послідовно 7 разів, а перший буфер ще не встиг закінчитися, то перший відтворений буфер буде видалений з черги, а усі буфери зсунуться на один вперед, тому 7 буфер стане 6 у черзі.

Робота моделі Гатлінг для даного випадку представлена на схемі, дивитися рис. 1.4.2.

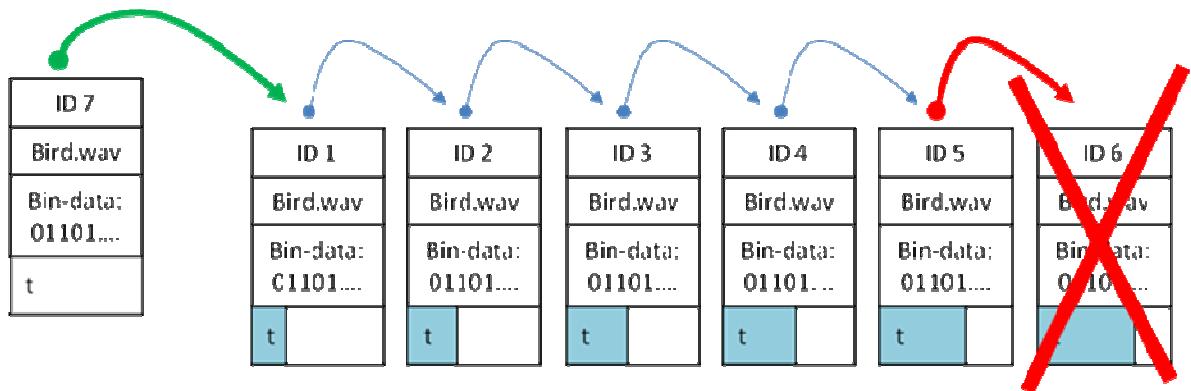


Рис. 1.4.1. Схема роботи моделі Гатлінга

Як бачимо на схемі, дивитися рис. 1.4.2., файл з ID 6 буде видалений з черги. А буфер з ID 7 – стане 6 у чорзі.

Дану схему досить просто реалізувати програмними засобами за допомогою стека, черги, списку або навіть масиву.

## ***2. Моделі поведінки камери***

Камера – точка огляду сцени з певними характеристиками. Камери в бувають двох типів - спрямована (Target) і вільна (Free) [10].

Для створення реалістичної поведінки камери при переміщенні користувача в тривимірному просторі, необхідно побудувати модель, що описує рух камери при здійсненні кроків користувачем.

Даний ефект можна зробити наступним чином:

- 1) Ключові кадри - цей вид анімації використовує послідовність заздалегідь розставлених ключових кадрів або позиції точок, створених вручну аніматором. Генерування проміжних кадрів або точок проводиться спеціальною програмою або функціями за допомогою морфинга, мікшування, змішування, складання [16, 17].
- 2) Запис руху - даний вид анімації записується за допомогою спеціального обладнання з реальних об'єктів, що рухаються, в тому числі людей і тварин, потім дані представляються у вигляді хмари рухомих точок, які використовуються для імітації руху об'єктів в комп'ютерній графіці.

Поширений приклад такої техніки - Motion capture, дослівний переклад з англійської - захоплення руху. Даний вид анімації є досить дорогим, так як потрібні реальні предмети або живі актори для запису анімації, обладнання для захоплення рухів, дуже потужне програмне забезпечення і величезні ресурси обчислювальної машини для швидкої обробки та збереження великих і швидко мінливих потоків інформації [17].

- 3) Процедурна анімація - процедурна анімація повністю або частково розраховується комп'ютером. Даний вид анімації вимагає мінімальну кількість оперативної пам'яті, так як всі дані

перераховуються динамічно і немає сенсу зберігати всю послідовність анімації в оперативній пам'яті, сама ж анімація представляється у вигляді функції або набору функцій, що її створюють. Але даний вид анімації вимогливий до частоти процесора комп'ютера, але тільки в разі дуже складних анімацій, в разі простих анімацій він так само виграє у всіх інших методів. Основною складністю даного методу, є саме конструювання функції необхідного виду, що само по собі є досить складним завданням, а так само максимальне спрощення цієї функції для зменшення обчислювальних ресурсів [14].

Виходячи з вимог дипломної роботи, ведеться розробка веб-додатку, тому необхідно максимально зменшити навантаження та використання процесорного часу і ресурсів оперативної пам'яті. Також процес, що описує рух камери при здійсненні кроків користувачем є фізичним процесом і його можна заздалегідь обрахувати, також він не містить хаотичної інформації, окрім тієї коли користувач керує камерою. Тобто таку анімацій можливо представити у вигляді функції.

Виходячи з цих вимог будемо користуватися третім варіантом, а саме процедурною або генерованою анімацією.

### **3. Модель штучного інтелекту NPC птиці**

#### **3.1. NPC**

NPC – Non-Player Character – з англ. «не керований гравцем, персонаж» – персонаж, що перебуває під контролем програми, а не користувача, проте може з ним взаємодіяти. У комп'ютерних програмах поведінку таких персонажів визначають програмно за допомогою штучного інтелекту.

NPC можуть бути дружніми, нейтральними та ворожими. NPC є важливим засобом створення атмосфери, мотивують користувача робити ті чи інші дії, є основним джерелом інформації про світ та сюжет [22].

Для вирішення цих завдань застосовуються різні види штучного інтелекту: алгоритми, нечітка логіка, ваговій граф, нейронні мережі, генетичні алгоритми [6] [8].

Такий підхід дозволяє обходитися без побудови складної і дорогої математичної моделі. Адже розробка точних математичних моделей в ряді випадків є досить складною, а іноді і зовсім неможливою, в зв'язку з неповнотою інформації про об'єкт і його параметри, зазвичай, це такі об'єкти, які досить складно формалізувати. Ситуація ще більше ускладнюється, якщо властивості об'єкта або процесу змінюються динамічно. Окрім цього знання для штучного інтелекту достатньо швидко змінюються і розвиваються, що в свою чергу може привести до повної переробки математичної моделі. В цьому випадку штучний інтелект проходить перенавчання і переналаштовується під нові знання.

У даній роботі розглядається створення штучного інтелекту NPC птиці. Так як опис усіх ситуацій положення об'єктів-перешкод є досить складним завданням. Для створення реалістичної поведінки птиці буде використовуватися одна з систем штучного інтелекту.

### *3.2. Системи штучного інтелекту*

Поняття штучного інтелекту багатогранно. Але кілька найбільш важливих аспектів все ж можна виділити:

- 1) По-перше, це питання про те, що таке штучний інтелект, адже визначення поняття обумовлює предмет, мету, методи, успішність дослідження.
- 2) Інтелект має на увазі обробку інформації, тому важливою є проблема подання знань в системах штучного інтелекту.
- 3) Існували і існують різні підходи до вирішення питань, пов'язаних зі створенням інтелектуальних систем, і їх розгляд проливає світло на багато аспектів проблеми.
- 4) Велике значення має забезпечення взаємодії систем штучного інтелекту з людиною на природній мові, так як при цьому значно полегшується ведення діалогу з ними.

Незважаючи на те, що, на думку деяких вчених, штучний інтелект принципово неможливий, розробки в області створення систем штучного інтелекту є в даний час одними з пріоритетних напрямків в науці.

Штучній інтелект (artificial intelligence - AI) – наукова дисципліна, яка займається проблемами імітації інтелекту. В рамках цієї дисципліни будується теорії і моделі, покликані пояснити роботу інтелекту і впровадити в технічних системах принципи і механізми інтелектуальної діяльності. Штучній інтелект – це один з напрямків інформатики, метою якого є розробка програмно-апаратних засобів, що дозволяють користувачеві ставити і вирішувати завдання, що традиційно вважаються інтелектуальними, спілкуючись з ЕОМ на обмеженій підмножині природної мови. Іншими словами штучній інтелект - це технічна (інформаційна та програмно-апаратна) реалізація імітації діяльності експерта або процесу.

Характеризуючи особливості систем штучного інтелекту, фахівці вказують на[20, 28]:

- 1) Наявність в них власної внутрішньої моделі зовнішнього світу; ця модель забезпечує індивідуальність, відносну самостійність системи в оцінці ситуації, можливість семантичної і прагматичної інтерпретації запитів до системи;
- 2) здатність поповнення наявних знань;
- 3) здатність до дедуктивного висновку, тобто до генерації інформації, яка в явному вигляді не міститься в системі; ця риса дозволяє системі конструювати інформаційну структуру з новою семантикою і практичною спрямованістю;
- 4) вміння оперувати в ситуаціях, пов'язаних з різними аспектами нечіткості, включаючи «розуміння» природної мови;
- 5) здатність до діалогової взаємодії з людиною;
- 6) здатність до адаптації.

На питання, чи всі перераховані умови є обов'язковими, необхідні для визнання системи інтелектуальної, вчені відповідають по-різному. У реальних дослідженнях, як правило, визнається абсолютно необхідним наявність внутрішньої моделі зовнішнього світу, і при цьому вважається достатнім виконання хоча б одного з перерахованих вище умов.

Популярні системи штучного інтелекту:

- Алгоритм — набір інструкцій, які описують порядок дій виконавця, щоб досягти результату розв'язання задачі за скінченну кількість дій;
- Нечітка логіка — Fuzzy Logic — розділ математики, який є узагальненням аристотелевої логіки і теорії множин. Уперше введений Лотфі Заде в 1965 році [23] як розділ, що вивчає об'єкти з функцією належності елемента до множини, яка приймає значення у інтервалі [0, 1], а не тільки 0 або 1. На основі цього поняття вводяться логічні операції над нечіткими множинами, і формулюється поняття лінгвістичної змінної, якою виступають нечіткі множини.

- Ваговій граф - або зважений граф в теорії ігор - дерево рішень або питально-відповідна система QA. Це граф, що складається з усіх можливих відповідей системи. В таких системах кількість можливих ситуацій заздалегідь визначена.
- Нейронні мережі - це математична модель, а також її програмна та апаратна реалізація, побудовані за принципом організації та функціонування біологічних нейронних мереж — мереж нервових клітин живого організму.
- Генетичний алгоритм – Genetic algorithm – це евристичний алгоритм пошуку, який використовується для вирішення задач оптимізації та моделювання шляхом випадкового підбору, комбінування і варіації параметрів з використанням механізмів, аналогічних природному відбору в природі, що нагадують біологічну еволюцію. Є різновидом еволюційних обчислень, за допомогою яких вирішуються оптимізаційні задачі з використанням методів природної еволюції, таких як успадкування, мутації, відбір і кросинговер.

### *3.3. Вибір системи штучного інтелекту*

Для вибору алгоритму розглянемо Переваги та Недоліки популярних систем штучного інтелекту:

- Алгоритм
  - Переваги:
    - найпростіший метод реалізації штучного інтелекту;
    - підходить для проектування простих, невеликих і нескладних систем;
    - має високу швидкість вирішення задач;
  - Недоліки:
    - досить складно вивести повний алгоритм;
    - чим складніший процес тим більша ймовірність допущення помилки в алгоритмі;
    - складно застосувати у всіх спектрах завдань в порівнянні з нейронними мережами і генетичними алгоритмами;
    - складно, а інколи і не можливо провести внесення нових можливостей в алгоритм та зробити його модернізацію, так як може знадобитися повна переробка алгоритму.
- Нечітка логіка
  - Переваги:
    - підходить для проектування простих та складних систем;
    - має високу швидкість вирішення задач;
    - є можливість навчання та перенавчання системи за допомогою: набору правил, параметрів системи, функцій приналежності;

- Дозволяє візуально продемонструвати хід прийняття того чи іншого рішення за допомогою графіків і функцій приналежності [27].
- Недоліки:
  - складно застосувати у всіх спектрах завдань в порівнянні з нейронними мережами і генетичними алгоритмами;
  - має ефект перенавчання;
  - необмежене зростання правил, які можуть значно сповільнити роботу системи;
- Ваговій граф
  - Переваги:
    - відносна простота реалізації
    - має високу швидкість вирішення задач;
    - легко аналізувати прийняті рішення переглянувши дерево виводу;
  - Недоліки:
    - необмежене зростання елементів графу, які можуть значно сповільнити роботу системи;
    - майже неможливо застосувати у всіх спектрах завдань в порівнянні з нейронними мережами і генетичними алгоритмами;
- Нейронні мережі
  - Переваги:
    - Адаптація до змін;
    - Стійкість до шумів у вхідних даних;
    - Великий спектр застосування для вирішення задач абсолютно різного роду
  - Недоліки:
    - відносно низька швидкістю виконання завдання;
    - не має виведення повного алгоритму;

- споживанням більшої кількості нейронів, ніж необхідно для вирішення завдання [28];
- труднощі аналізу результатів роботи нейронної мережі і пояснень, чому вона прийняла те чи інше рішення;
- Генетичний алгоритм
  - Переваги:
    - Адаптація до змін;
    - Стійкість до шумів у вхідних даних;
    - Великий спектр застосування для вирішення задач абсолютно різного роду
  - Недоліки:
    - відносно низька швидкістю виконання завдання;
    - не має виведення повного алгоритму;
    - ефективно сформулювати завдання, визначити критерій відбору хромосом (задати код) і інші параметри ГА може тільки фахівець;
    - труднощі аналізу результатів роботи генетичного алгоритму і пояснень, чому він прийняла те чи інше рішення;

Для вибору системи штучного інтелекту розглянемо штучний інтелект птиці як систему з необмеженою кількістю відповідей. Де відповідю системи є контроль швидкості руху і поворот птиці в сторону відсутності перешкод штучнім інтелектом.

Існує безліч інтелектуальних систем здатних вирішувати завдання такого роду. Для вирішення завдань з необмеженою кількістю відповідей в основному використовуються нейронні мережі та генетичні алгоритми - так як дані системи здатні навчатися набагато швидше, здатні змінювати свою структуру і універсальні для всіх класів задач. Але при цьому мають явні недоліки, а саме швидкість виконання завдання, вивід повного алгоритму,

споживання більшої кількості нейронів, ніж необхідно для вирішення завдання [28].

Алгоритми нечіткої логіки в основному застосовуються в системах, де всі можливі висновки системи заздалегідь визначені. Дані алгоритми мають високу швидкість вирішення завдань. Дозволяють візуально продемонструвати хід прийняття того чи іншого рішення за допомогою графіків і функцій належності [27]. Але при цьому дана експертна система має недолік: цей алгоритм складно застосувати у всіх спектрах завдань в порівнянні з нейронними мережами і генетичними алгоритмами. Це пов'язано з невизначеністю відповідей і необмеженістю області можливих рішень.

В алгоритмах нечіткої логіки для розширення спектра завдань використовуються різні методи навчання і налаштування системи, які [32] [33] [34] [35]:

- доповнюють і виправляють вихідний набір правил, що описує стану системи. Вихідний набір правил формується людиною і внаслідок чого може мати не повний набір правил, суперечливі або ідентичні правила;

- проводять налаштування параметрів системи, так як ці параметри спочатку задаються експертом суб'єктивно, що може призводити до неправильної роботі системи.

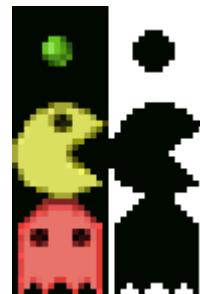
- підбирають вид функцій принадлежності - аналізуючи результати роботи системи в різних ситуаціях.

Дані методи дозволяють значно поліпшити ефективність роботи системи, але мають ряд недоліків: ефект перенавчання; необмежене зростання правил;

#### **4. Моделі двовимірних і тривимірних спрайтів**

У комп'ютерній графіці Спрайт – Sprite – це двомірний растровий малюнок, зазвичай з альфа каналом, інтегрований у більшу сцену (рис. 4.1).

З 1970 років під спрайтами розуміли невеликі малюнки, які виводилися на екран із застосуванням системного прискорення (MSX 1, NES).



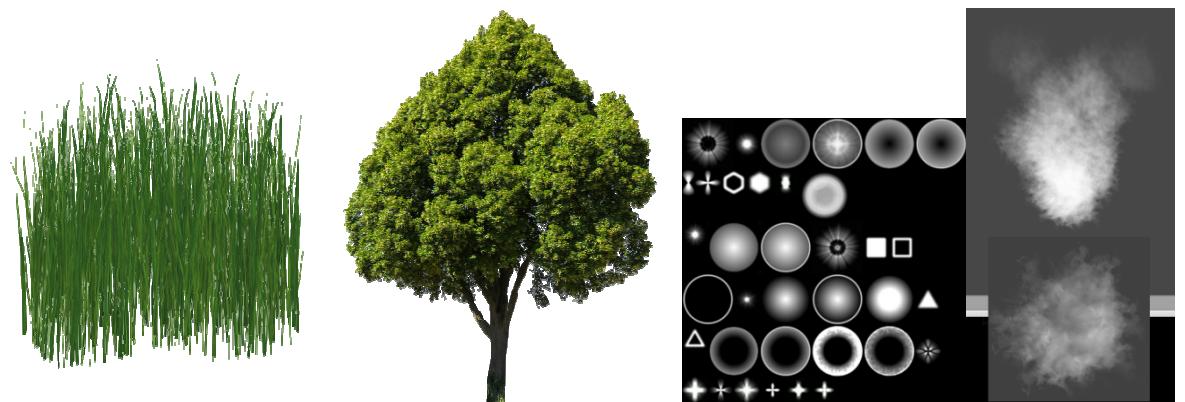
**Рис. 4.1. Спрайт Пакман та Привід – текстура та альфа канал**

Сьогодні спрайти використовуються для відтворення рослинності, віддалених об'єктів або для потоку частинок [36]. Іноді їх називають "рекламними щитами" – "billboards" або "Z-sprites".

У тривимірному світі двовимірний спрайтовий об'єкт постійно тримається до спостерігача нормаллю до поверхні, що призводить до того, що він візуально «повертається» весь час до спостерігача «обличчям».

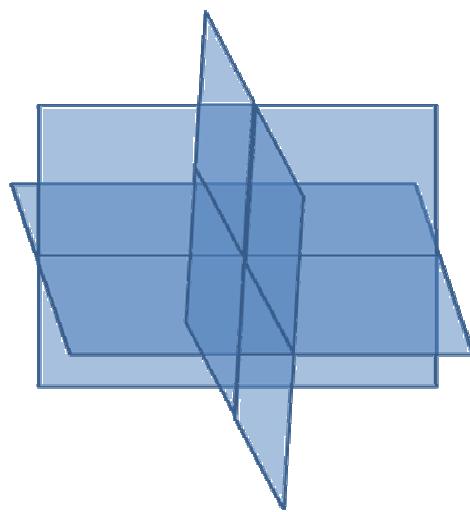
Спрайти можуть застосовуватися для оптимізації графічного ядра, коли оформлення тривимірної деталізованої моделі надто дорогое і може привести до сильного падіння швидкості рендерінгу (рис. 4.2):

- Об'єкти, присутні в сцені з великою кількістю і які, будучи тривимірними, мали б велику кількість граней і дуже сильно вплинули на продуктивність - наприклад, масовка в задньому плані, дим, трава та інші.
- Віддалені об'єкти, які зблизька малюються полігональними моделями, за ступенем віддаленості перетворюються в спрайти різної деталізації (LOD).



**Рис. 4.2. Спрайт трава, дерево, частки, дим з часток**

Під тривимірним спрайтом мається на увазі три двовимірні спрайти, що розглядаються, як три ортогональні поверхні (рис. 4.2). Центри яких знаходяться в одній точці.



**Рис. 4.3. Тривимірних Спрайт**

## ***5. Реалізація 3D веб-додатку та моделей. Створення додаткового ПО.***

### ***5.1. Реалізація моделі об'ємного звуку***

#### ***5.1.1. Реалізація UML діаграм моделі об'ємного звуку***

За допомогою UML діаграм і програмного забезпечення «White Star UML» була реалізована схема моделі об'ємного звуку Рис. 5.1.1.1..

В даній UML діаграмі моделі об'ємного звуку:

- FILE\_CODECS – це декодери звукових файлів, таких як:.wmv, .mp3, .aac, .ac3, .flac, .dts, .ogg, .wma та інші.
- PCM\_GENERATORS – генератори PCM даних.
- PCM – об'єкт PCM даних.
- Buffer – Буфери (аудіобуфери) – містять аудіодані в форматі PCM, як покажчик.
- Source – джерела звуку.
- Listener – об'єкт Слухача.
- EAX та SoundFX – Sound Effects – різні види ефектів, такі як: відтворення звуку відповідно до місця розташування використовуючи моделі об'ємного звуку (ефект Доплера), відлуння, фленжер, зворотне відлуння, фейзер, ревербератор, сустейн та інші.
- PCM\_out – вихідний сигнал PCM даних.

Реалізацію роботи моделі можна побачити на сайті <https://knightdanila.github.io/WebAudio/GatlingGun/Audio%20test.htm> або <https://knightdanila.github.io> [39,40].

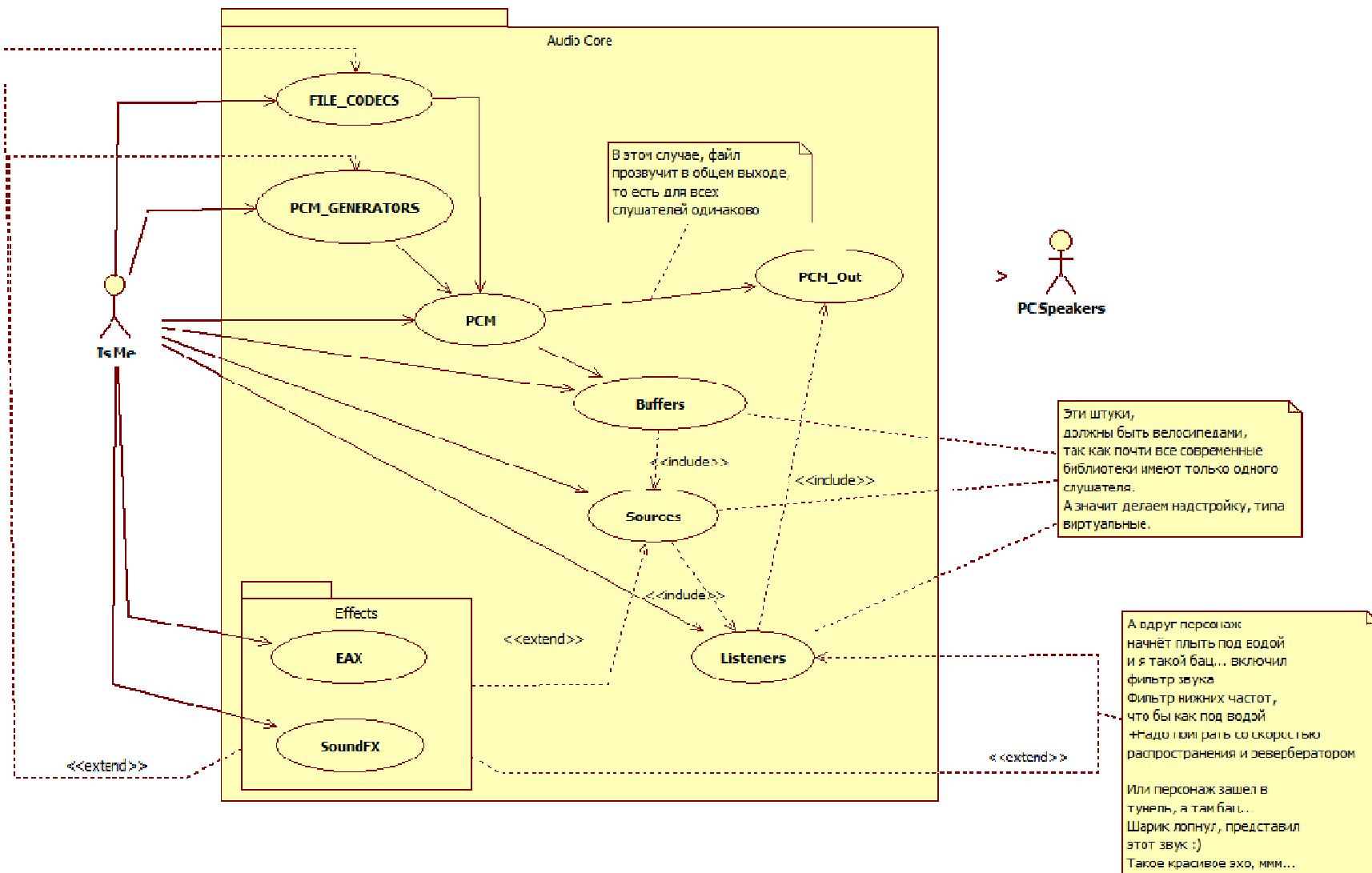


Рис. 5.1.1.1. UML Схема моделі об'ємного звуку

### 5.1.2. Реалізація ефекту Доплера

Ефект Доплера залежить від швидкості джерела та слухача відносно середовища та швидкості поширення звуку в цьому середовищі. Сума частотного зсуву (зміна тону) пропорційна швидкості слухача та джерела по їхній лінії зору.

Реалізований Ефект Доплера, описується формулою нижче та схемою Рис. 5.1.2.1.:

SS: SPEED\_OF\_SOUND = швидкість звуку (значення за замовчуванням 343.3)

DF: DOPPLER\_FACTOR = коефіцієнт Доплера (за замовчуванням 1.0)

vls: Скалярна швидкість слухача

vss: Скалярна швидкість джерела

f: Частота вибірки

f ': Ефект Доплера - зсувна частота

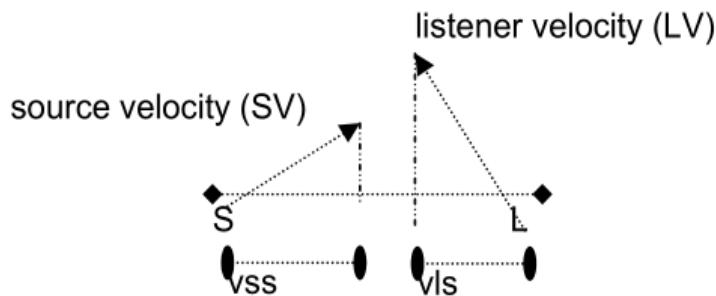
SL: Source to listener vector = вектор джерело слухач

SV: Source Velocity vector = Вектори швидкості джерела

LV: Listener Velocity vector = Вектори швидкості слухача

Mag: magnitude or length = довжина вектора

DotProduct = Скалярний добуток



**Рис. 5.1.2.1. Схема ефекту Доплера**

Математичне представлення  $v_{ls}$  та  $v_{ss}$ :

$$\text{Mag}(\text{vector}) = \sqrt{\text{vector.x} * \text{vector.x} + \text{vector.y} * \text{vector.y} + \text{vector.z} * \text{vector.z}}$$

$$\text{DotProduct}(v1, v2) = (v1.x * v2.x + v1.y * v2.y + v1.z * v2.z)$$

$$v_{ls} = \text{DotProduct}(SL, LV) / \text{Mag}(SL)$$

$$v_{ss} = \text{DotProduct}(SL, SV) / \text{Mag}(SL)$$

Обчислення ефекту Доплера:

$$v_{ss} = \min(v_{ss}, SS/DF)$$

$$v_{ls} = \min(v_{ls}, SS/DF)$$

$$f' = f * (SS - DF*v_{ls}) / (SS - DF*v_{ss})$$

Считаю, что это нужно набрать в редакторе формул

## **5.2. Реалізація моделі поведінки камери**

Для створення реалістичної поведінки камери при переміщенні користувача в тривимірному просторі, необхідно зрозуміти суть процесу. Це потрібно для того, щоб створи математичну модель процесу.

Необхідно побудувати модель, що описує рух камери при здійсненні кроків користувачем. У параграфі 93 книги з фізики [18] можемо знайти наступну інформацію, що при переміщенні тіла в горизонтальній площині сила тяжіння не здійснює роботи. Вся робота, яку доводиться витрачати при такому переміщенні - це робота на подолання тертя і опору середовища. Коли велосипедист їде по горизонтальному шляху, він не здійснює роботи проти сили тяжіння; тільки піднімаючись вгору, він здійснює роботу проти цієї сили. Дещо інша справа з пішоходом. При ходьбі по горизонтальному шляху центр ваги тіла людини не залишається на одній і тій же висоті, а при кожному кроці піднімається і потім знову опускається. Коли центр ваги піднімається вгору, людина здійснює роботу. Тому при ходьбі навіть по горизонтальному шляху здійснюється робота не тільки проти сили опору середовища, а й проти сили тяжіння. Вважаючи, що при кожному кроці центр ваги піднімається на 5 см, а маса людини дорівнює 70 кг, знайдемо, що при кожному кроці відбувається досить значна робота 35 Дж на підняття центру ваги. Негативна ж робота при опусканні центру ваги не використовується. Правильна хода зменшує витрату роботи при ходьбі і тому менше стомлює.

Так як камери в комп'ютерній графіці бувають двох типів - Target (Спрямована) і Free (Вільна) [10]. А при кожному кроці людина зміщує свій центр ваги, при цьому положення голови трохи «погойдується», а положення цілі на яку спрямований погляд людини незмінне, можна припустити, що слід використовувати спрямовану камеру (Target).

Тоді даного ефекту можна досягти використовуючи модель коливання маятника, коливання якого будуть передаватися камері як її положення, а мета камери, цілі на яку спрямований погляд, буде залишатися незмінною. Для цього скористаємося простим диференціальним рівнянням коливання маятника (нескінченного коливання):

$$\text{diff}(x(t), t) = -y(t),$$

$$\text{diff}(y(t), t) = x(t)$$

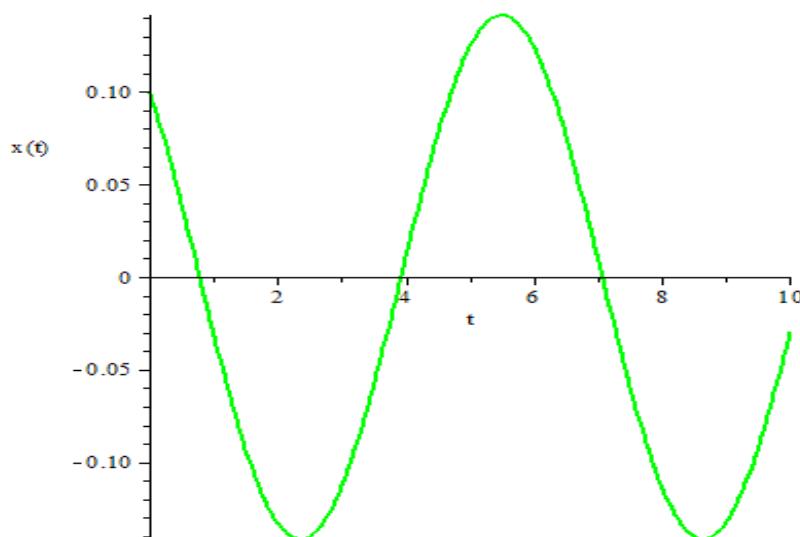
Загальне рішення цього рівняння буде мати вигляд:

$$x(t) = C1 * \sin(t) + C2 * \cos(t),$$

$$y(t) = -C1 * \cos(t) + C2 * \sin(t)$$

**Необхідно набрати в редакторе формул**

При початкових даних  $x(0)=0.1$ ,  $y(0)=0.1$  отримано наступні графіки Рис. 5.2.1. та Рис. 5.2.2.



**Рис. 5.2.1. 2D - графік  $x(t)$  та  $t$**

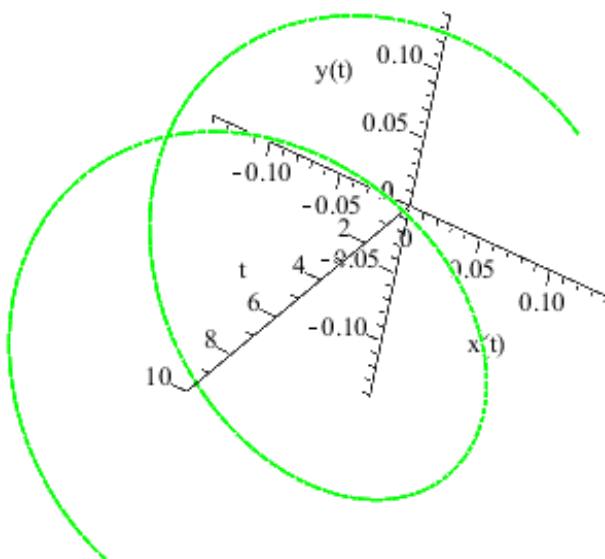


Рис. 5.2.2. 3D - графік  $x(t)$ ,  $y(t)$  та  $t$

Виходячи з цього загального розв'язку , побудуємо новий вид функцій, зробимо функцію у залежною від функції  $x$  і максимально спростимо самі функції:

$$x(t) = a * \sin(t);$$

$$y(x) = \text{abs}(\sin((1/2) * \pi * x))$$

де  $a$  - коефіцієнт прискорення в даній моделі знаходиться на інтервалі  $[0..1]$ ,  $t$  - час від початку першого кроку з циклу кроків, при кожній зупинці, цей показник починається знову з 0.

Отримано наступні графіки Рис. 5.2.3. та Рис. 5.2.4.

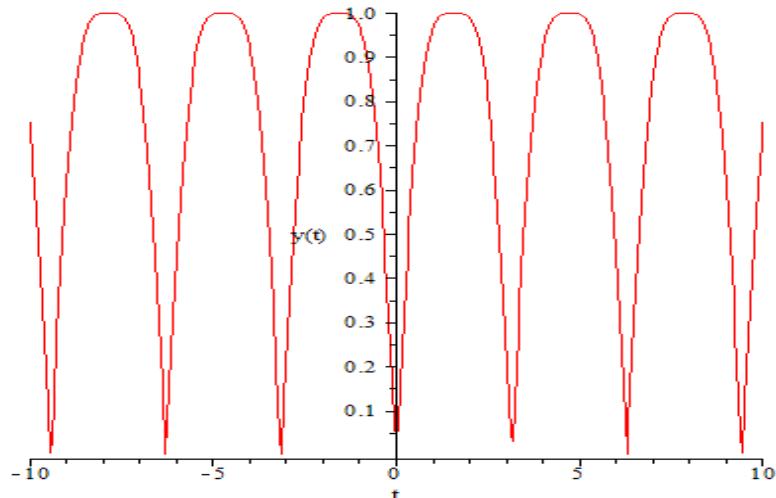


Рис. 5.2.3. 2D - графік  $y(t)$  та  $t$

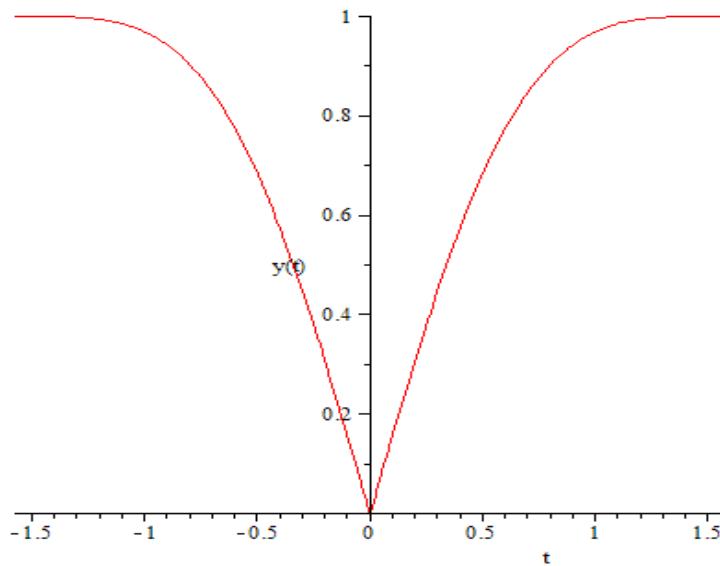


Рис. 5.2.4. 2D - графік  $y(t)$  та  $t$ , де  $t = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$

Тепер для створення більш плавного переходу, замінимо модуль на зведення функції в квадрат.

$$x(t) = a * \sin(t);$$

$$y(x) = \text{pow}(\sin((1/2) * \text{PI} * x), 2);$$

Тепер додамо ще кілька змінні для створення початкової швидкості і прискорення, та замінимо змінну часу  $t$  на  $x0$  – що виступає в якості часу від початку першого кроку з циклу кроків.

$$x0 = x0 + xs * \text{walkRun}; \quad \text{у } x \text{ индекс 0 должен быть}$$

**внизу**

$$x(x0) = a * \sin(x0) * SW;$$

$$y(x) = \text{pow}(\sin((1/2) * \text{PI} * x), 2) * SH;$$

У даній моделі (всі значення змінних і їх інтервали отримані експериментальним шляхом, залежно від масштабів, можуть відрізнятися):

- **x0** – визначає початкове положення для  $x$ , при старті дорівнює 0, так само виступає в якості часу від початку першого кроку з циклу кроків,

при кожній зупинці, цей показник починається знову з 0. Ця змінна прив'язана до кнопки – дії «йти вперед – W or UP\_ARROW».

- xs – «x speed» – швидкість ходьби, дорівнює 0.1.
- walkRun – визначає стан руху, ходьба або біг, при ходьбі дорівнює 1, при бігу 1.7. Ця змінна прив'язана до кнопки – дії «прискорення - SHIFT».
- a – коефіцієнт прискорення в даній моделі знаходиться на інтервалі [0..1], в залежності від початку руху, якщо персонаж не рухається дорівнює 0, якщо персонаж почав рух поступово збільшується залежно від здібностей персонажа до прискорення, через деякий час ставати рівним 1 .
- SW – «screen width» – коефіцієнт масштабування анімації відносно ширини екрану пристрою.
- SH – «screen height» – коефіцієнт масштабування анімації відносно висоти екрану пристрою.

Отримано наступні графіки Рис. 5.2.5. та Рис. 5.2.6.

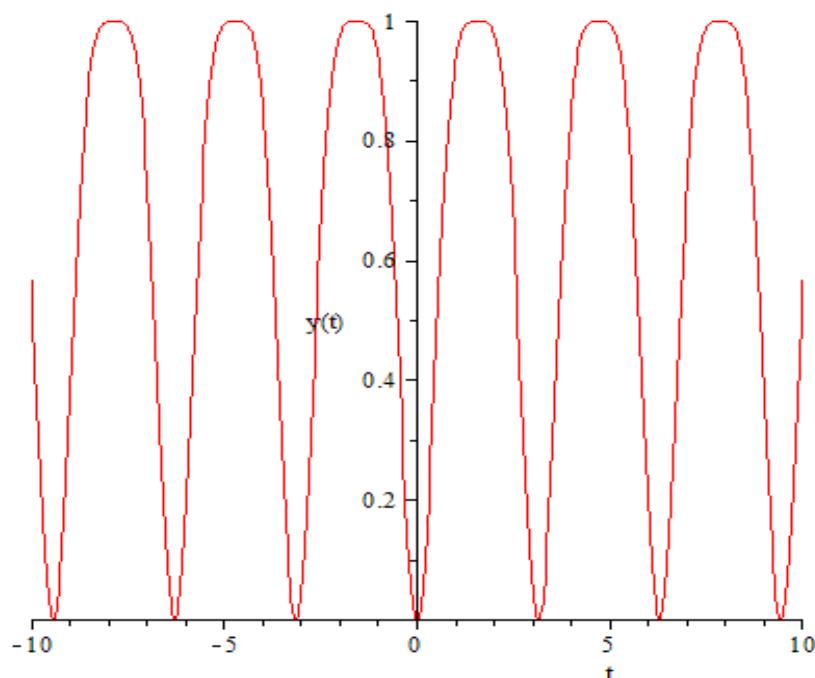


Рис. 5.2.5. 2D - графік  $y(t)$  та  $t$

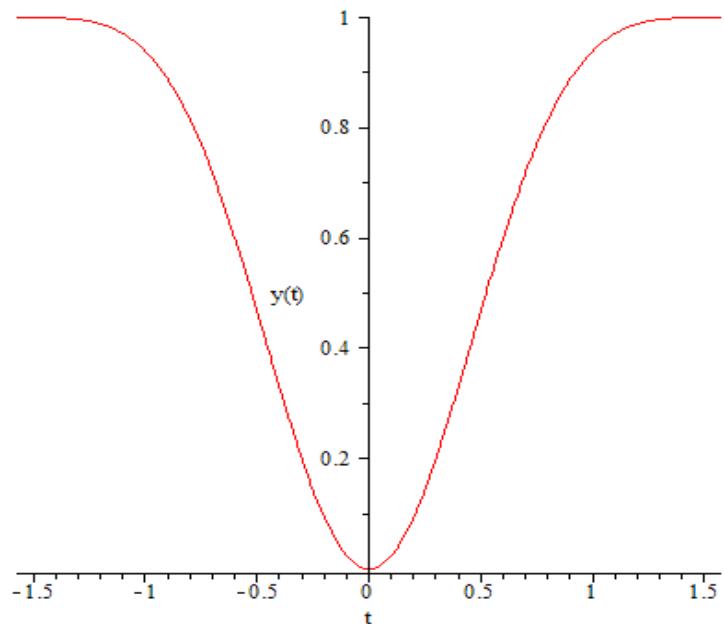


Рис. 5.2.6. 2D - графік  $y(t)$  та  $t$ , де  $t = \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$

Основний код, що виконує дану функцію на мові JavaScript представлено нижче:

```
<script>

// Ініціалізація змінних
var a = 0, as = 0.05, x = 0, x0 = 0, xs = 0.1, y = 0, walkRun = 1;
var LEFT = 37, UP = 38, RIGHT = 39, DOWN = 40,
P_KEY = 80, = 17, SHIFT = 16, ENTER = 13;
var dirs = { [LEFT]: 0, [UP]: 0, [RIGHT]: 0, [DOWN]: 0, [SHIFT] : 0 };
var SPEED = 10;

// Клавіші
$(document).keydown(function (e) {
    dirs[e.keyCode] = 1;
    if(dirs[ENTER]){ changePanorama360();}
    if(dirs[P_KEY]){ seeHideRedPoint();}
});
$(document).keyup(function (e) {dirs[e.keyCode] = 0; });

// Функція вмикач – для червоної точки, що відображає траєкторію
var seeRedPoint = true;
function seeHideRedPoint(){
    if(seeRedPoint){
        seeRedPoint = false;
        document.getElementById("iPoint").style.display = "none";
    } else {
        seeRedPoint = true;
        document.getElementById("iPoint").style.display = "block";
    }
}

// Головна функція для траєкторії
function move() {
    // Перевірка прискорення – вмикає біг
    if (dirs[SHIFT] || dirs[CTRL]) {
        if (walkRun < 1.7) {walkRun = walkRun + as;
            if(walkRun > 1.7){ walkRun = 1.7; }
        }
    } else {
        // Повернення до нормального шагу – до спокійної ходьби
        if (walkRun > 1) { walkRun = walkRun - as;
            if(walkRun < 1){ walkRun = 1; }
        }
    }
}
```

```

        }
    }

    // Перевірка прискорення – вмикає спокійну ходьбу
    if (dirs[UP]) {
        if (firstStep) {x0 = 0; firstStep = false; }
        if (a < 1) {
            a += as;
            if (a > 1) {a = 1; }
        } else {a = 1;}
        // Повернення до стану очікування
    } else {
        firstStep = true;
        if (a > 0) {
            a -= as;
            if (a < 0) { a = 0; }
        } else { a = 0; }
    }
    // Запуск головна функція для траєкторії
    move();
    // Функція розрахунку траєкторії
    x0 += xs * walkRun;
    x = a * Math.sin(x0) * 100;
    y = (-1) * Math.pow(Math.sin(Math.PI / 2 * x / 100), 2) * 100;

    // Функція малювання – для червоної точки, що відображає траєкторію
    if(seeRedPoint){
        $("#iPoint").css('transform', `translate(${x}px, ${y}px)`);
    }
}

</script>
```

Покрокова реалізація роботи моделі:

- 1) Поведінка камери лише по осі x Рис. 5.2.7.
- 2) Поведінка камери по осі x та у Рис. 5.2.8.
- 3) Поведінка камери по осі x та у. Сцена та камера Рис. 5.2.9.
- 4) Поведінка камери по осі x та у. Сцена, камера та рух Рис. 5.2.10.

Реалізацію роботи моделі можна побачити на сайтах [41, 42, 43, 44]:

[knightdanila.github.io/Walking/test\\_1.html](https://knightdanila.github.io/Walking/test_1.html);

[knightdanila.github.io/Walking/test\\_2.html](https://knightdanila.github.io/Walking/test_2.html);

[knightdanila.github.io/Walking/test\\_3.html](https://knightdanila.github.io/Walking/test_3.html);

[knightdanila.github.io/Walking/test\\_4.html](https://knightdanila.github.io/Walking/test_4.html);

або на сайті <https://knightdanila.github.io> [39].

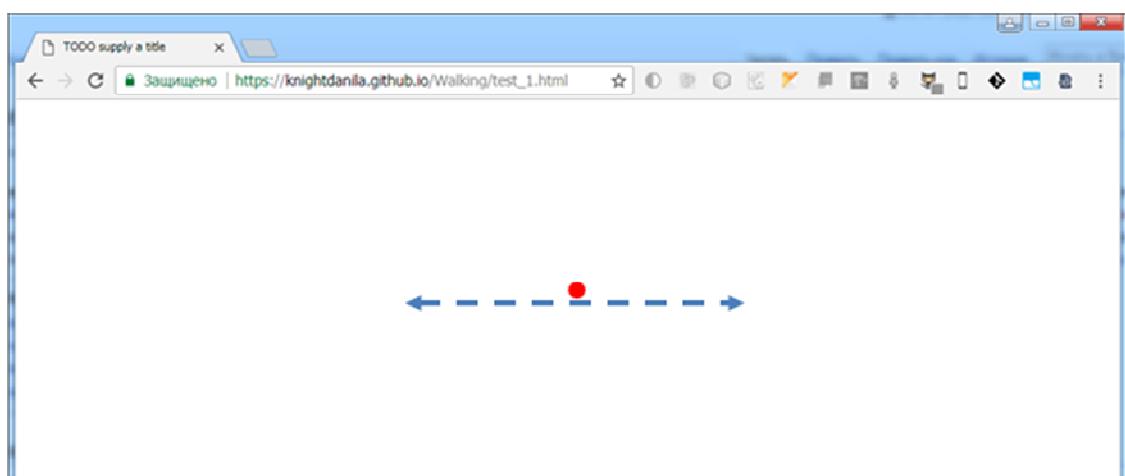


Рис. 5.2.7. test 1 – поведінка камери по осі x

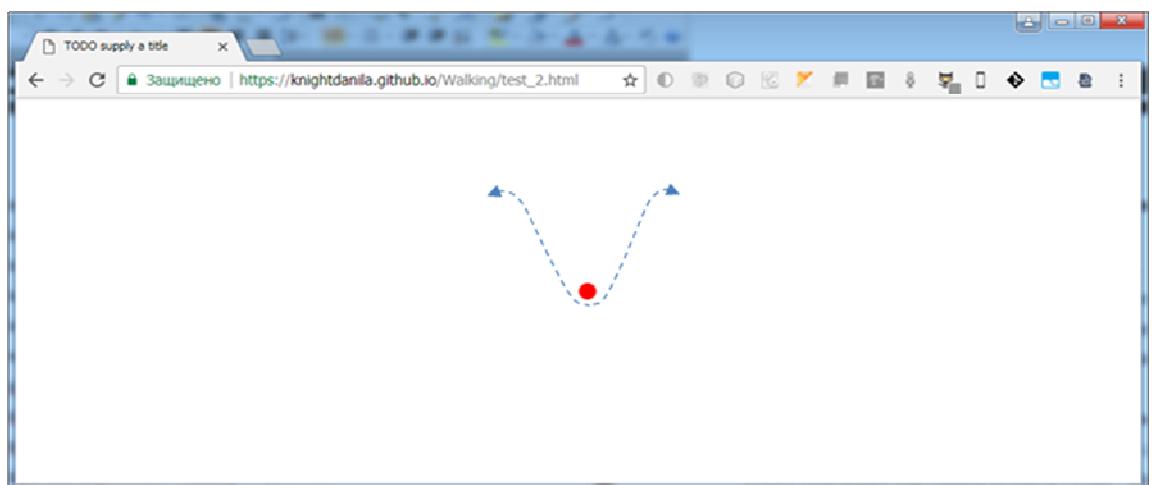


Рис. 5.2.8. test 2 – Поведінка камери по осі x та у

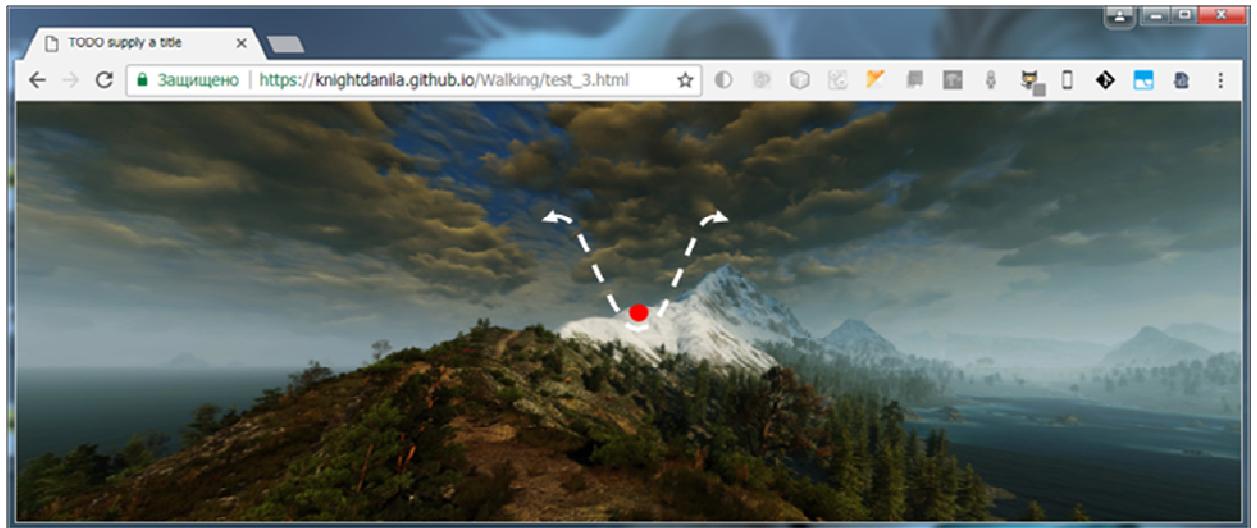


Рис. 5.2.9. test 3 – Поведінка камери по осі х та у. Сцена та камера

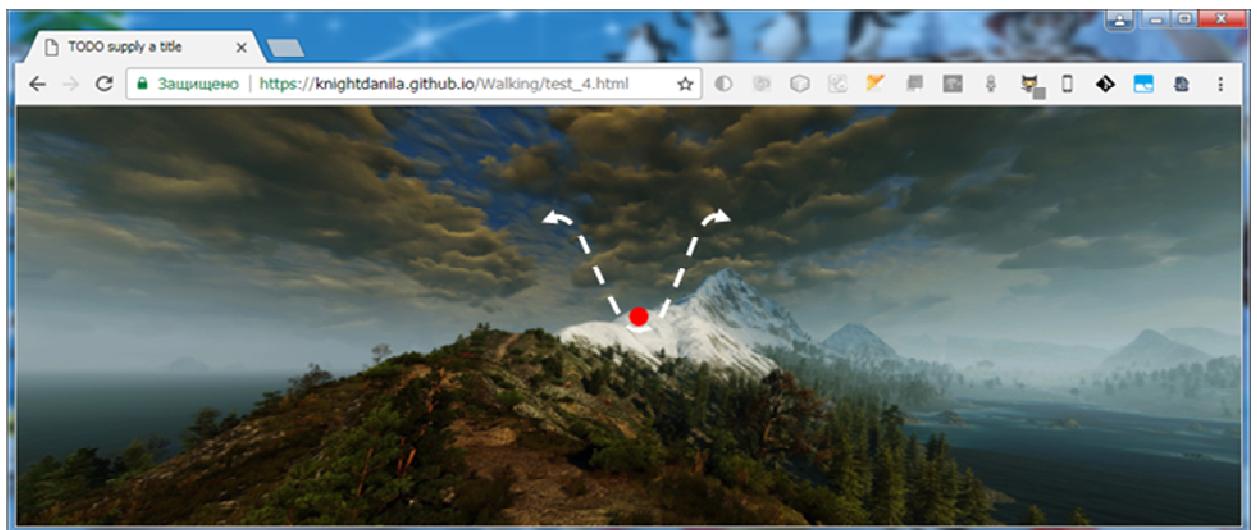


Рис. 5.2.10. test 4 – Поведінка камери по осі х та у. Сцена, камера та рух

### **5.3. Реалізація моделі штучного інтелекту NPC птиці**

#### **5.3.1. Реалізація алгоритму штучного інтелекту NPC птиці**

В роботі ведеться розробка алгоритму на базі нечіткої логіки здатного вирішувати задачі більш широкого класу. Даний алгоритм повинен підходити для моделей, в яких заздалегідь невідомо, скільки можливих відповідей, може повернути система.

Алгоритм повинен мати можливість розширення кількості відповідей, які може повернати система, при цьому правила системи повинні залишатися незмінними. Подібні вимоги дозволяють уникати додаткових перевірок логічних правил на наявність протиріч, після додавання нових відповідей в систему.

Реалізації даного алгоритму:

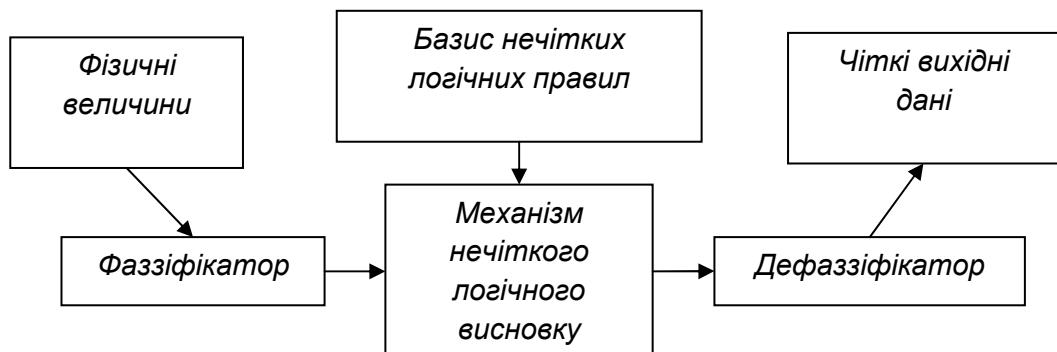
- 1) Проведення класифікації, відображення довільного користувача в множину бази даних персонажів коміксів «CMD - Combat Marvel DC» [45].
- 2) Управління польотом птиці - на базі шаблонів - можливих поводжень, для обминання перешкод «Flying Bird» [47].

Задача полягає у виділенні основних характеристик об'єкта, за якими і буде проходити класифікація. Для цих характеристик створюються нечіткі логічні правила, якими буде користуватися механізм нечіткого логічного висновку.

В основному система нечіткої логіки складається з наступних об'єктів (Рис. 5.3.1.1.) [2] [3] [7] [10]:

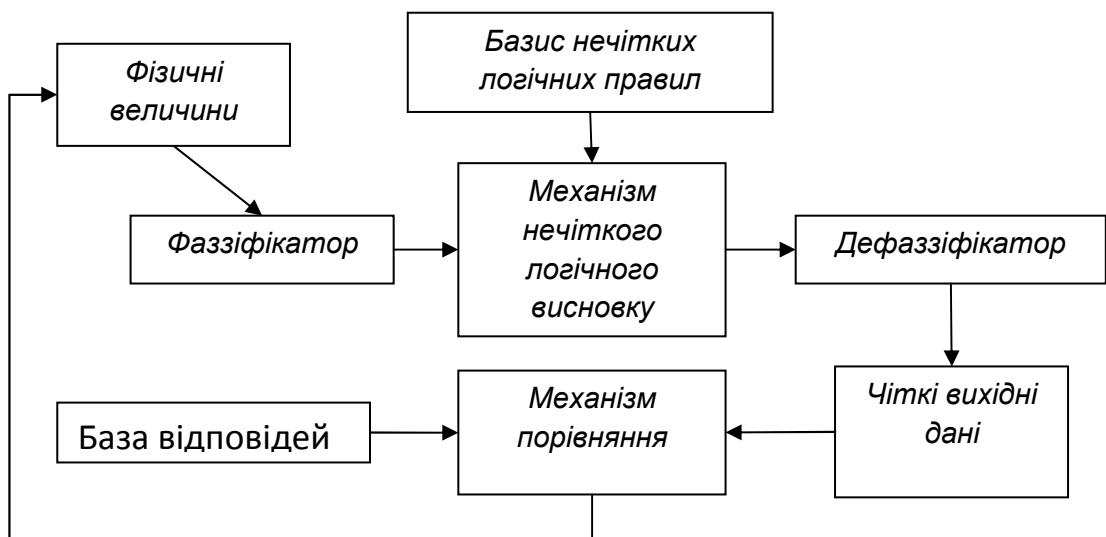
- 1) Фаззіфікатор - механізм, який відображає чітке фізичне значення в нечітку множину.

- 2) Механізм нечіткого логічного висновку - приймає «Нечіткі вхідні дані» - фактично, дані створені фаззіфікатором і на основі базису нечітких логічних правил, повертає «Нечіткі вихідні дані» - дані, які передаються на вхід дефаззіфікатору.
- 3) Дефаззіфікатор - механізм, який відображає нечітку множину в чітке фізичне значення.



**Рис. 5.3.1.1. Стандартний алгоритм нечіткої логіки**

В даній задачі відбувається додавання ще одного об'єкта, а саме механізму порівняння, тобто кожен об'єкт, який введений в систему проходить через стандартний алгоритм нечіткої логіки, після чого повертає деяке фізичне значення, яке описує цей об'єкт [13]. Потім всі об'єкти бази відповідей системи, також проходять через алгоритм нечіткої логіки. Після чого відбувається пошук найбільш схожого об'єкта (Рис. 5.3.1.2.).



**Рис. 5.3.1.2. Модифікований алгоритм нечіткої логіки**

**В этом разделе у меня не отобразились формулы**

Запропонований алгоритм реалізується послідовністю дій. В якості вихідної інформації є: базис нечітких логічних правил; фаззіфікатор; механізм нечіткого логічного висновку; дефаззіфікатор; вхідний об'єкт  $\mathbf{Y}$ ; база

$$\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$$

відповідей, наприклад з 3 об'єктів:

$$\mathbf{R} = \mathbf{R}_1, \dots, \mathbf{R}_n$$

Базис нечітких логічних правил . Необхідною вимогою до бази знань є повнота використовуваних правил:

$$\text{Supp } \mathbf{R}_i$$

де – носій нечіткої множини  $\mathbf{R}_i$ .

Це означає, що для кожного поточного стану процесу  $\mathbf{Y}$  існує хоча б одне керуюче правило, посилка якого має ненульову ступінь приналежності для  $\mathbf{Y}$ .

Вхідний об'єкт  $\mathbf{Y}$  – має наступні параметри (характеристики):

;

**X<sub>1</sub>, X<sub>2</sub>, X<sub>3</sub>**

База відповідей містить, наприклад, 3 об'єкти:

кожен з яких має наступні параметри

$$\mathbf{X}_i = \mathbf{x}_1, \dots, \mathbf{x}_m;$$

;

Компоненти  $\mathbf{y}^j$ ,  $i = 1, \dots, n$ ,  $j = 1, \dots, m$  є окремими характеристиками об'єкта.

Кількість характеристик  $m$ ,  $j = 1, \dots, m$  визначає розмірність простору характеристик.

Крок 1. Проведення об'єкта  $\mathbf{Y}$  через стандартний алгоритм нечіткої логіки.

- 1) На етапі фаззіфікатора - отримуємо принадлежність параметрів об'єкта  $\mathbf{Y}$  до термів системи.
- 2) Механізм нечіткого логічного висновку та базис нечітких логічних правил - повертають нечіткі вихідні дані.
- 3) На етапі дефаззіфікатора - отримуємо чітке фізичне значення, бажано отримати саме одне фізичне значення, таким чином, це буде унікальне значення, яке описує весь об'єкт.

**X<sub>1</sub>, X<sub>2</sub>, X<sub>3</sub>**

Крок 2. Проведення об'єктів бази відповідей через стандартний алгоритм нечіткої логіки.

За аналогією з кроком 1, кожен об'єкт бази відповідей  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$

– буде описаний деяким унікальним значенням.

Крок 3. Пошук відстані між об'єктом  $\mathbf{y}$  і об'єктами бази відповідей  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ .

Відстань визначається на основі обраної метрики в просторі характеристик. Міри близькості елементів, можуть бути обчислені як: евклідова відстань, відстань по Хемменгу, відстань Чебишева або ін.

Крок 4. Висновок.

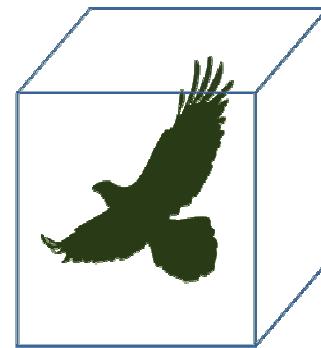
$\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$

Висновком системи, є вибір відповіді серед  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ , тобто система повертає об'єкт, який знаходиться найближче до об'єкта  $\mathbf{y}$ . Фактично ми відобразили об'єкт  $\mathbf{y}$  в безліч об'єктів бази відповідей  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ .

Для розробки NPC птиці буде використовуватися алгоритм описаний вище.

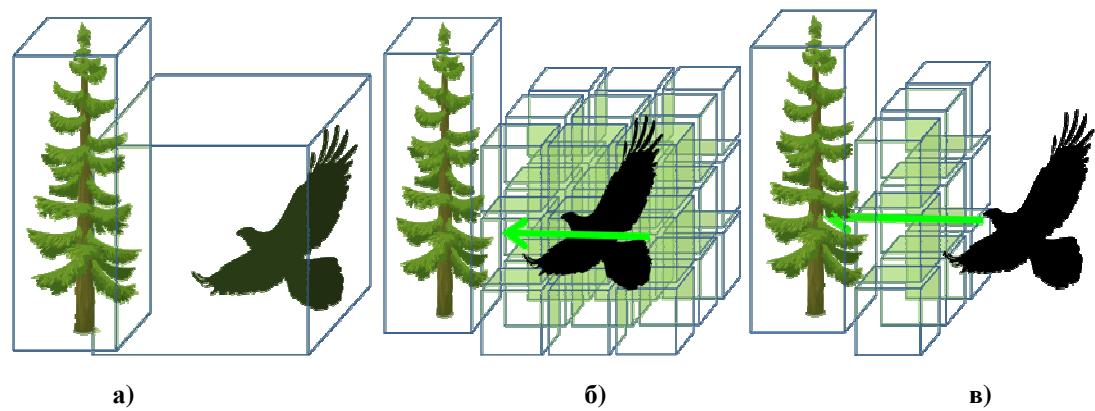
### 5.3.2. Реалізація зору NPC птиці

Навколо птиці є мінімальна обмежувальна коробка (minimum bounding box), як і навколо інших об'єктів. Про те у випадку птиці, це її поле зору (Рис. 5.3.2.1.)



**Рис. 5.3.2.1. Minimum bounding box**

Якщо, відбувається зіткнення обмежувальної коробки птиці з іншим об'єктом (Рис. 5.3.2.2. а), птиця вмикає другий шар зору, з куба, що складається 3x3x3 коробок (Рис. 5.3.2.2. б). Після цього знаходить сторону (Рис. 5.3.2.2. в).



**Рис. 5.3.2.2. Етапи виконання алгоритму**

**а)** - зіткнення,      **б)** - другий шар зору,      **в)** – пошук сторони

Після цього знайдена сторона, що складається з 3x3 коробок перетворюється на матрицю рівнів перетину.

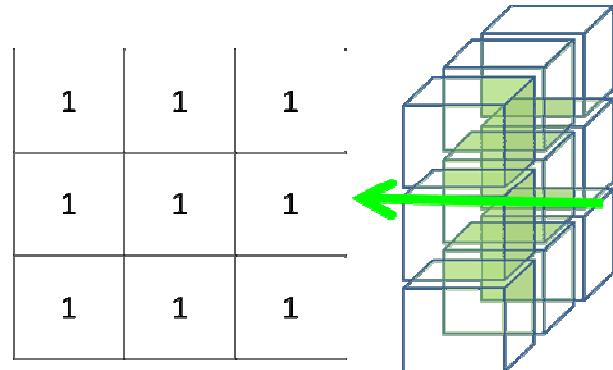


Рис. 5.3.2.3. матрицю рівнів перетину

Матриця рівнів перетину – зберігає в собі відстань внутрішніх перетинів, чи рівнів.

Після цього матриця розглядається як 4 підматриці (Рис. 5.3.2.2. а).


Рис. 5.3.2.4. 4 підматриці

Далі ця матриця передається у алгоритм описаний вище, де зберігаються різні матриці, що відповідають різним ситуаціям. Алгоритм знаходить подібну матрицю, та використовує її правила руху.

## 5.4 Реалізація моделі двовимірних і тривимірних спрайтів

Для реалізації двовимірних спрайтів у тривимірному світі необхідно повертати спрайт до спостерігача «обличчям» (Рис. 5.4.1.).

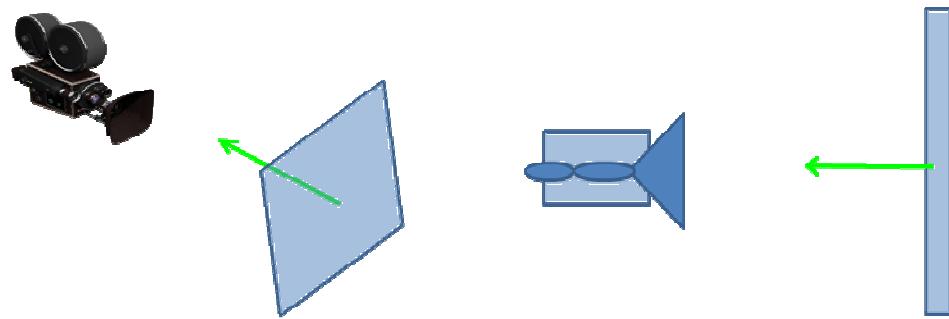


Рис. 5.4.1. Спрайт та камера.

Використаємо для цього функцію atan2 – яка повертає кут між осью x і вектором (x, y), як правило, в радіанах і підписаний таким чином, що для позитивного y отримується кут між 0 і  $\pi$ , а для негативних y результат знаходиться між  $-\pi$  і 0.

$$x_1 = -10; y_1 = 0; \quad x_2 = 10; y_2 = 0;$$

$\text{atan2}(x_2-x_1, y_2-y_1) = +1.570796 \text{ radians}$  (позитивне обертання за годинниковою стрілкою)

$$x_1 = 10; y_1 = -10; \quad x_2 = -10; y_2 = 10;$$

$\text{atan2}(x_2-x_1, y_2-y_1) = -0.785398 \text{ radians}$  (негативне обертання, проти годинникової стрілки)

Де  $x_1, y_1$  – координати камери,  $x_2, y_2$  – координати спрайту.

Реалізацію роботи моделі можна побачити на сайтах  
[knightdanila.github.io/Sprite3DViewer](https://knightdanila.github.io/Sprite3DViewer) ,  
[knightdanila.github.io/Walking/Sprite\\_test\\_1.html](https://knightdanila.github.io/Walking/Sprite_test_1.html) або  
<https://knightdanila.github.io> [39, 48, 49].

## ***6. Додаткове програмне забезпечення***

Для створення проекту використовуються наступні програми забезпечення:

- 1) Git
- 2) GIMP
- 3) Blender
- 4) AwesomeBump
- 5) White Star UML

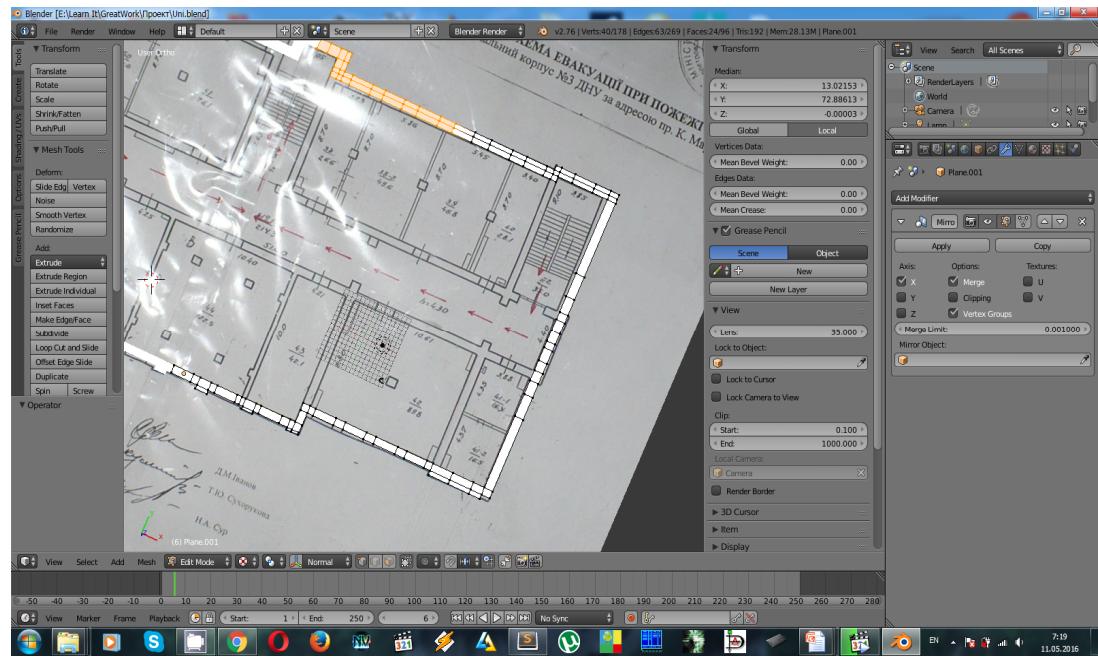
Всі перераховані вище програми є безкоштовним програмним забезпеченням. Мають відкриті вихідні коди. А так же є кросплатформним ПО.

Git — розподілена система керування версіями файлів та спільної розробки. Використовується для завантаження веб-додатку на сервер і його подальшої розробки [46].

GIMP (The GNU Image Manipulation Program) — растроїй графічний редактор, із деякою підтримкою векторної графіки [38].

Blender — пакет для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, вимальовування, після-обробки відео, а також створення відеоігор [37].

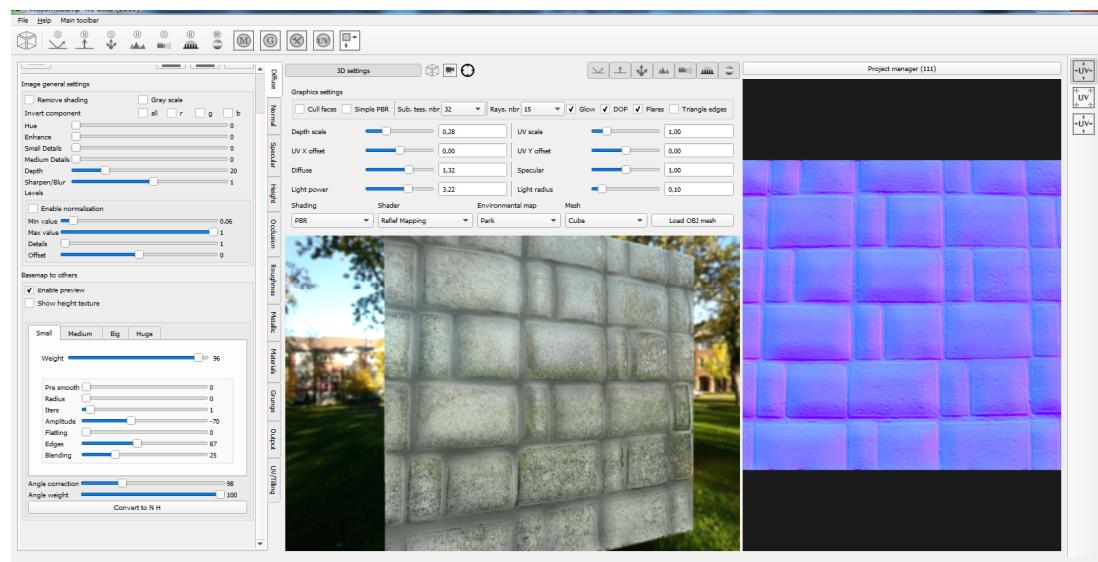
На (рис 3.4.1.) зображено використання програми Blender для створення 3D моделі університету.



**Рис. 3.4.1. Веб-Додаток Canvas, JavaScript, WebGL, GLSL**

AwesomeBump - це безкоштовна програма, призначений для створення нормальної, висотної, дзеркальної та для інших видів текстури з одного зображення. Оскільки обробка зображень виконується в 99% на GPU програма працює дуже швидко, і всі параметри можуть бути змінені в режимі реального часу.

На рис. 3.4.2. зображено використання AwesomeBump для створення текстури нормалі з фотографії зробленої на території університету.



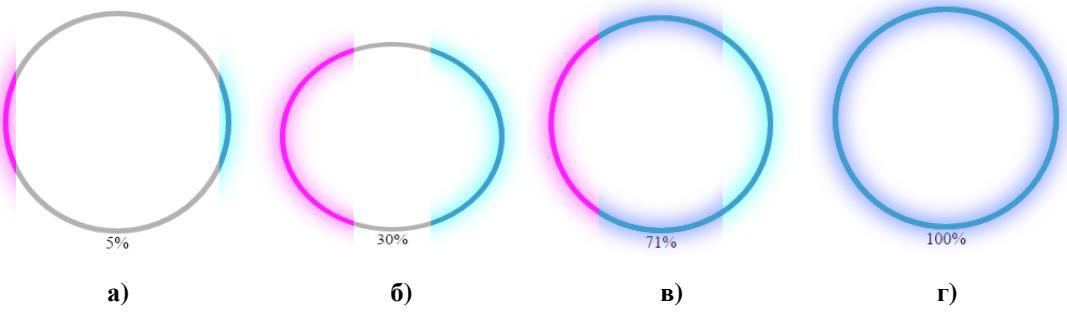
**Рис. 3.4.2. Веб-Додаток Canvas, JavaScript, WebGL, GLSL**

## 7. Результати використання створених алгоритмів та ПО

### 7.1. Створення 3D веб-додатку

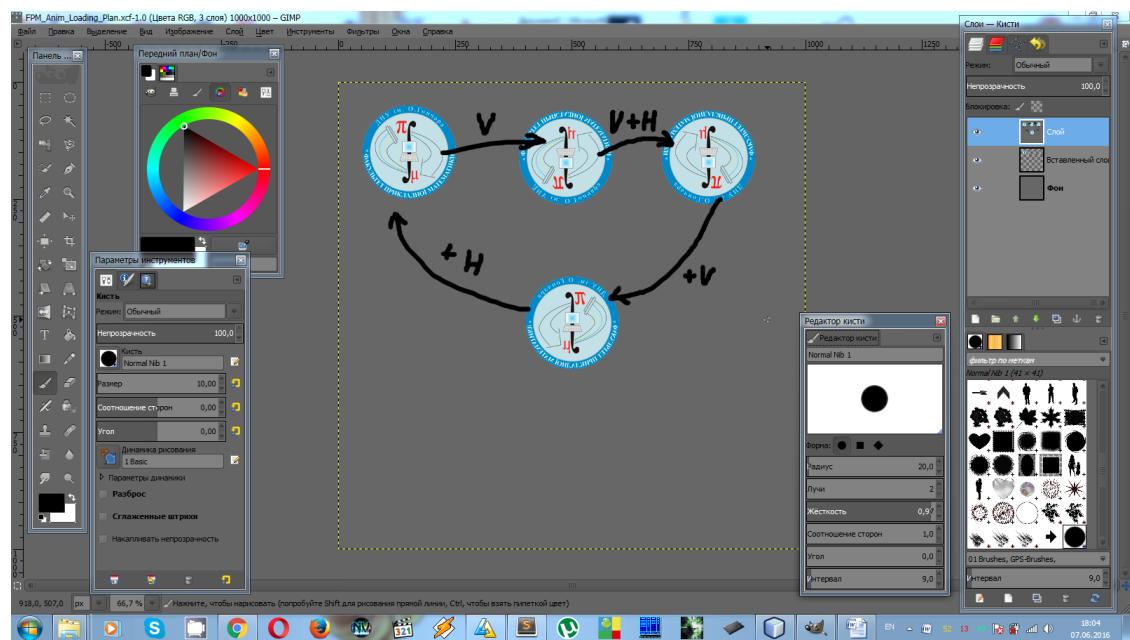
Все почалося зі створення дизайну завантаження для веб-додатку, адже треба наочно показати скільки інформації завантажено.

Перший етап розробки дизайну завантаження (Рис. 7.1.1.).



**Рис. 7.1.1. Етапи завантаження.**  
а) – 5%, б) – 30%, в) – 70%, г) - 100%

Далі за допомогою програми Gimp була створена схема перетворень [20] гербу факультету, для стилізації завантаження (Рис. 7.1.2).



**Рис. 7.1.2. схема перетворень.**

За допомогою схеми було написано наступний код (Рис. 7.1.3), який відтворює ці самі перетворення герба факультету.

Рис. 7.1.3. код перетворень

Другий етап розробки дизайну завантаження (Рис. 7.1.4.).



**Рис. 7.1.4. Другий дизайну завантаження**

За допомогою класу THREE.SceneLoader(), що пропонує нам бібліотека ThreeJS, створемо інтерактивну загрузку нашого проекту:

```
var loader = new THREE.SceneLoader();
```

Зробимо на фоні загрузки деяку сцену, нехай це буде велика кількість кубиків розташованих у випадковому порядку на нашій сцені.

```
var object, geometry, material, light, count = 100, range = 200;
material = new THREE.MeshLambertMaterial({color: 0xffffffff});
geometry = new THREE.BoxGeometry(5, 5, 5);
for (var i = 0; i < count; i++) {
    object = new THREE.Mesh(geometry, material);
    object.position.x = (Math.random() - 0.5) * range;
    object.position.y = (Math.random() - 0.5) * range;
    object.position.z = (Math.random() - 0.5) * range;
    object.rotation.x = Math.random() * 6;
    object.rotation.y = Math.random() * 6;
    object.rotation.z = Math.random() * 6;
    object.matrixAutoUpdate = false;
    object.updateMatrix();
    result.scene.add(object);
}
```

Створимо у центрі сцени джерело світла:

```
light = new THREE.DirectionalLight(0x111111);
light.position.x = 1;
result.scene.add(light);
```

Отримали наступне зображення (7.1.5):

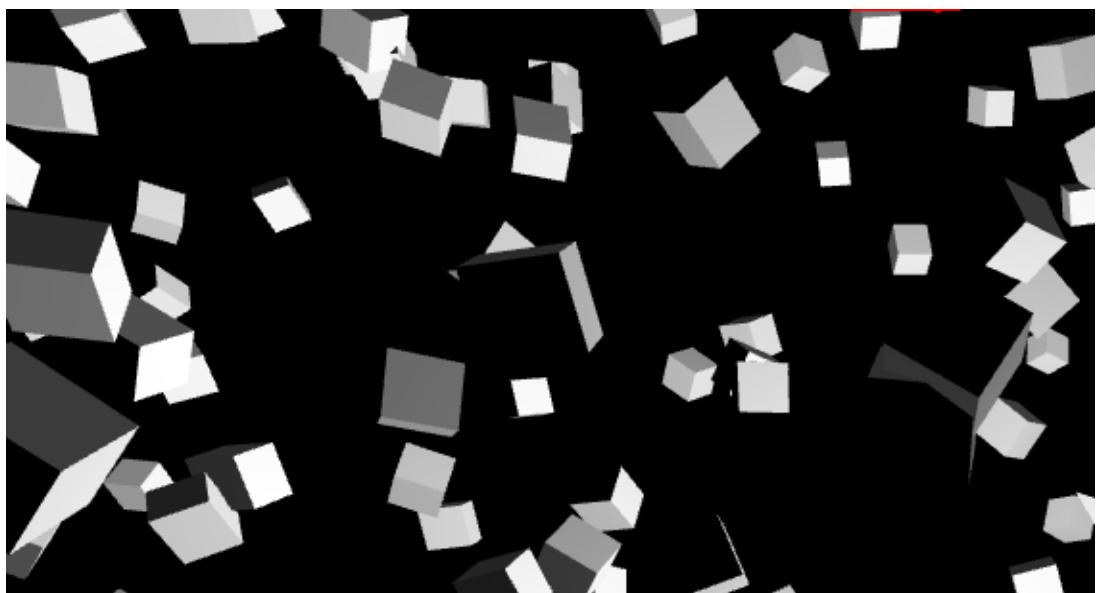


Рис. 7.1.5. Проста сцена

Тепер обернемо отриманий код у функцію і передамо її до завантажувача сцени.

```
loader.callbackProgress = callbackProgress;
loader.load("...", callbackFinished);
```

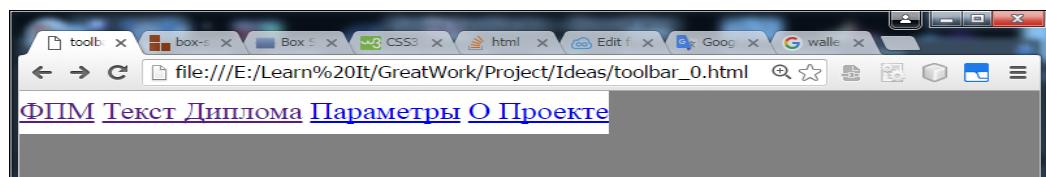
Тепер об'єднаємо отриману сцену з дизайном завантаження і трохи підправимо саму сцену. Отримали наступний результат (Рис. 7.1.6.).



**Рис. 7.1.6. Загрузка та сцена**

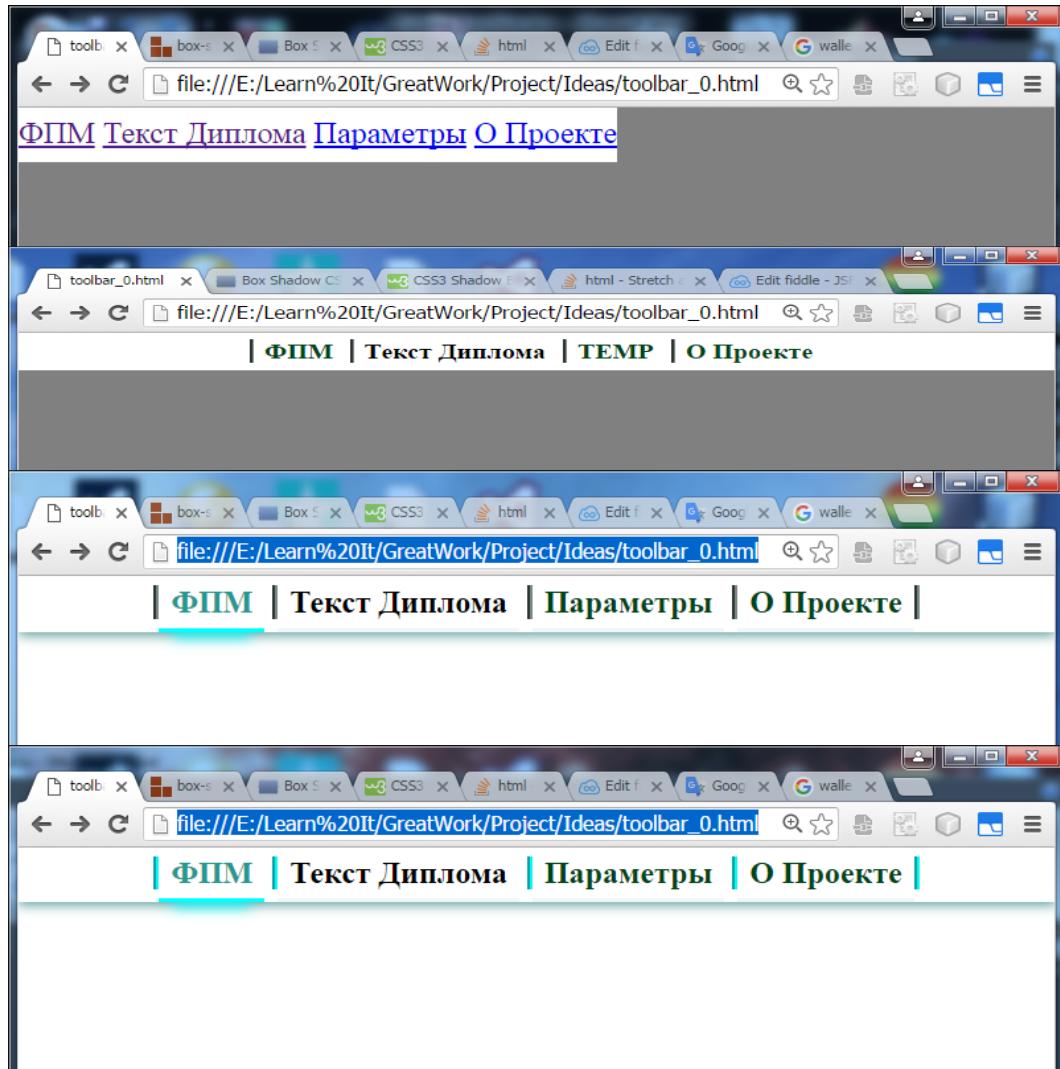
Почнемо розробку верхнього пункту меню.

На першій стадії розробки створимо звичайний шаблон меню, який з часом будемо змінювати та доповнювати (Рис. 7.1.7.)



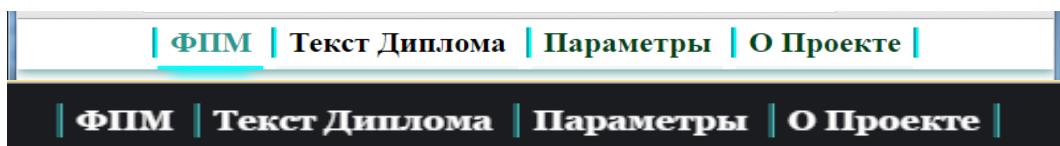
**Рис. 7.1.7. Шаблон меню**

Тепер створимо для нашого шаблону стилі та зробимо його більш цікавим. На (Рис. 7.1.8.) зображене процес створення дизайну для пункту меню.



**Рис. 7.1.8. Процес створення дизайну для пункту меню.**

Даний пункт меню реалізований в двох стилях світлий та темній (рис. 7.1.9), які користувач може вибирати з пункту параметри.



**Рис. 7.1.9. Світливий та темній стилі.**

Тепер об'єднаємо отриманий пункт меню з головним веб-додатком.  
Отримали наступний результат (Рис. 7.1.10.).

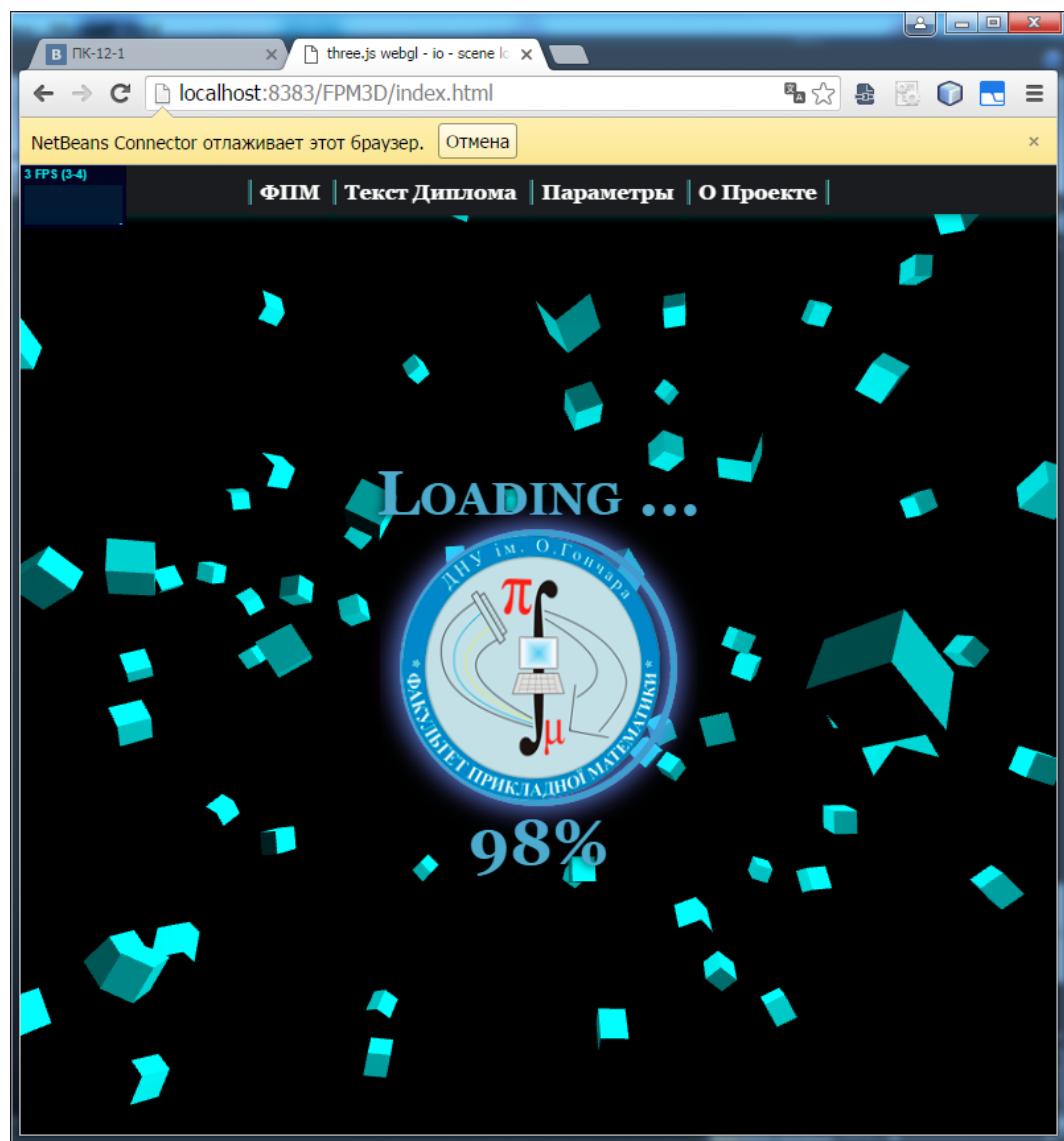


Рис. 7.1.10. Пункт меню та головний веб-додаток.

## 7.2. Завантаження 3D веб-додатку на сервер GitHub pages



Тепер необхідна реєстрація на GitHub. Що таке GitHub? GitHub — один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git і розроблений на Ruby on Rails і Erlang компанією GitHub, Inc. Сервіс безкоштовний для проектів з відкритим вихідним кодом, з наданням користувачам усіх своїх можливостей (включаючи SSL). 21 вересня 2011 року кількість користувачів стала більшою за мільйон.

Крім того що GitHub — Найбільший веб-сервіс для хостингу IT-проектів і їх спільної розробки и заснований на системі контролю версій Git. Цей веб-сервіс можна використовувати як хостинг для веб-додатків. Це повністю безкоштовна технологія. Це стало можливо за допомогою GitHub Pages.

Спочатку створимо аккаунт на сайті GitHub (Рис. 7.2.1.).

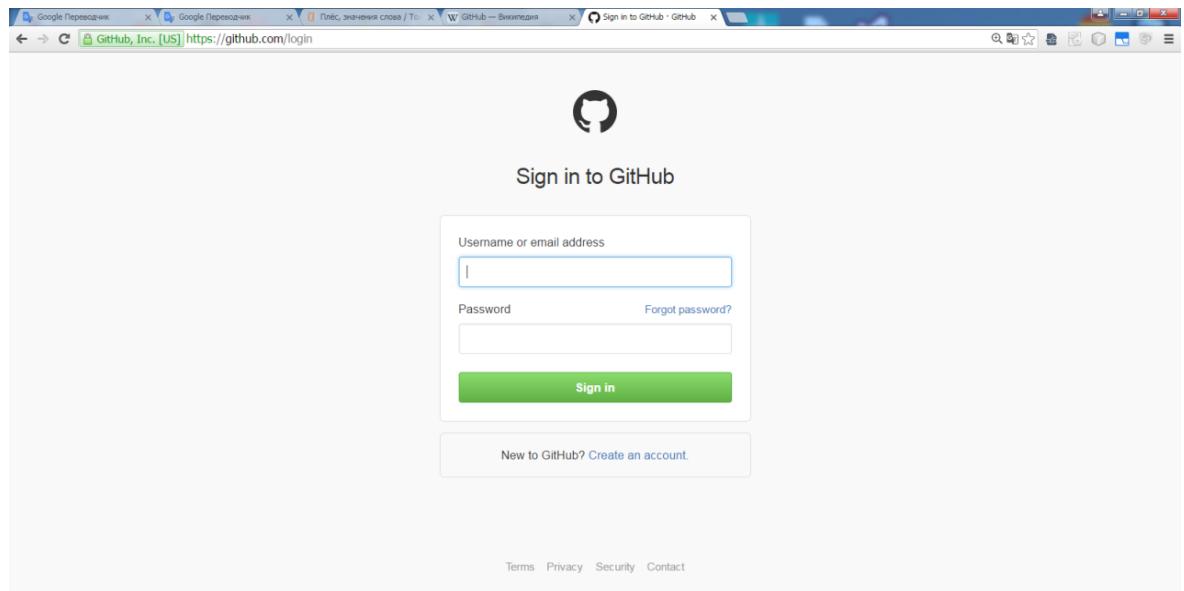


Introducing unlimited

All of our paid plans on GitHub.com now include unlimited private repositories. [Sign up to](#)

**Рис. 7.2.1. Головна сторінка сайту.**

За допомогою наступної форми треба увійти в свій акаунт GitHub (Рис. 4.2.2.).



**Рис. 7.2.2. Сторінка входу в акаунт.**

За допомогою програми Git-bush і команди cd переходимо до каталогу у якому знаходиться проект та командою git init створюємо новий репозиторій для проекту який має назву FPM3D. Тепер проект отримує гілку майстер. Переходимо у папку з проектом командою cd (Рис. 7.2.3.).

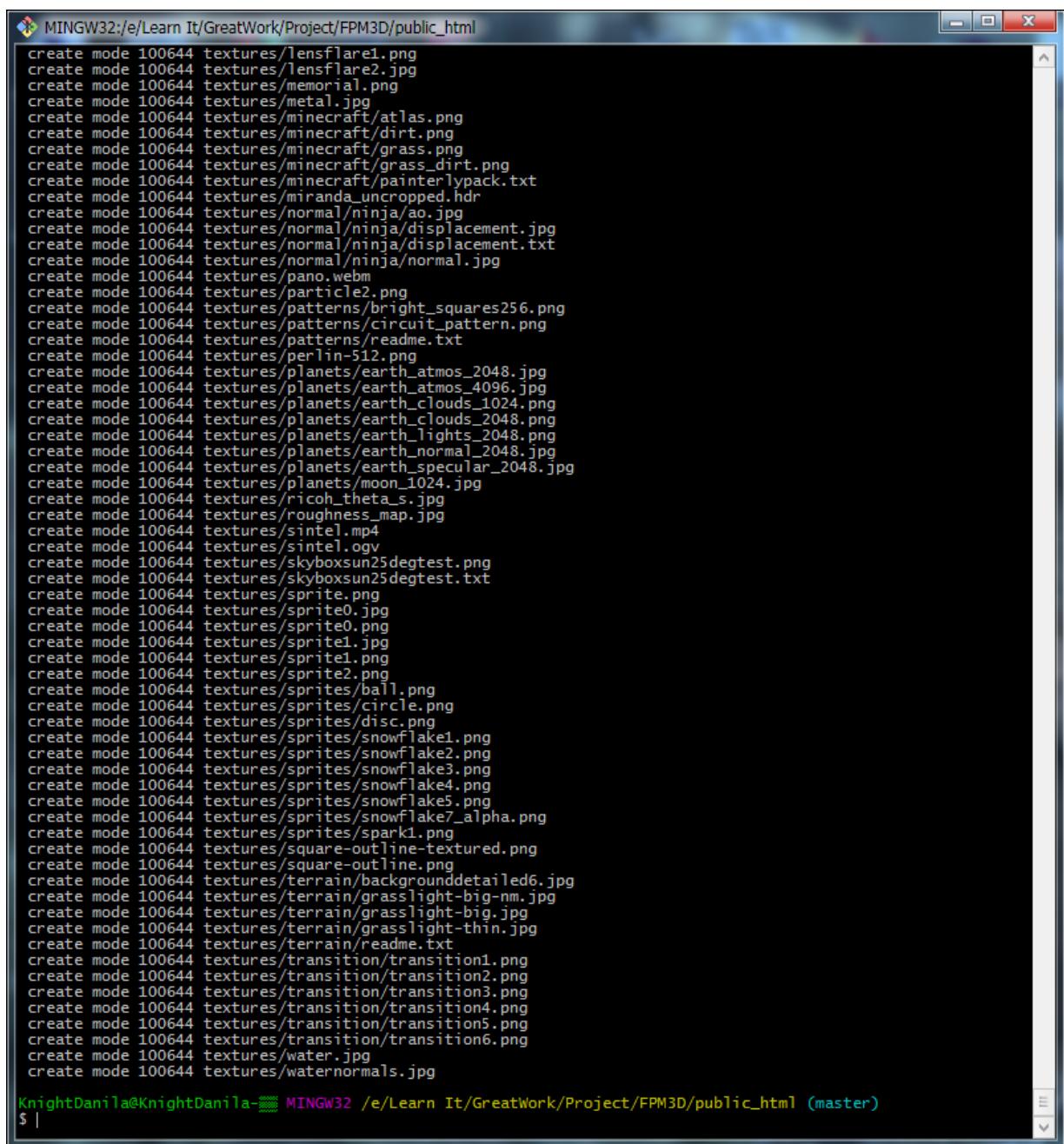
 A screenshot of a terminal window titled "MINGW32:/e/Learn It/GreatWork/Project/FPM3D/public\_html". The window shows a command-line session. The user has run the command "cd "E:\Learn It\GreatWork\Project\FPM3D\public\_html"" and is now in the directory "e/Learn It/GreatWork/Project/FPM3D/public\_html" on the "master" branch. The terminal window has a blue header bar and a black body with white text.
 

```

MINGW32:/e/Learn It/GreatWork/Project/FPM3D/public_html
$ cd "E:\Learn It\GreatWork\Project\FPM3D\public_html"
KnightDanila@KnightDanila-MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)
$ |
  
```

**Рис. 7.2.3. Сторінка входу в аккаунт.**

Тепер за допомогою команди `git status` дізнаємося стан проекту і потім виконавши команди “`git add .`” – добавити файли, “`git commit -m`” “додали нові файли” ” – добавили інформацію про зміну у файлах (Рис. 7.2.4.), “`git remote add origin https://github.com/FPM3D/FPM3D.github.io.git`” – підключилися до гілки віддаленого проекту, потім виконали наступну команду, щоб завантажити проект, “`git push -u original master`” – завантажили проект на сервер.

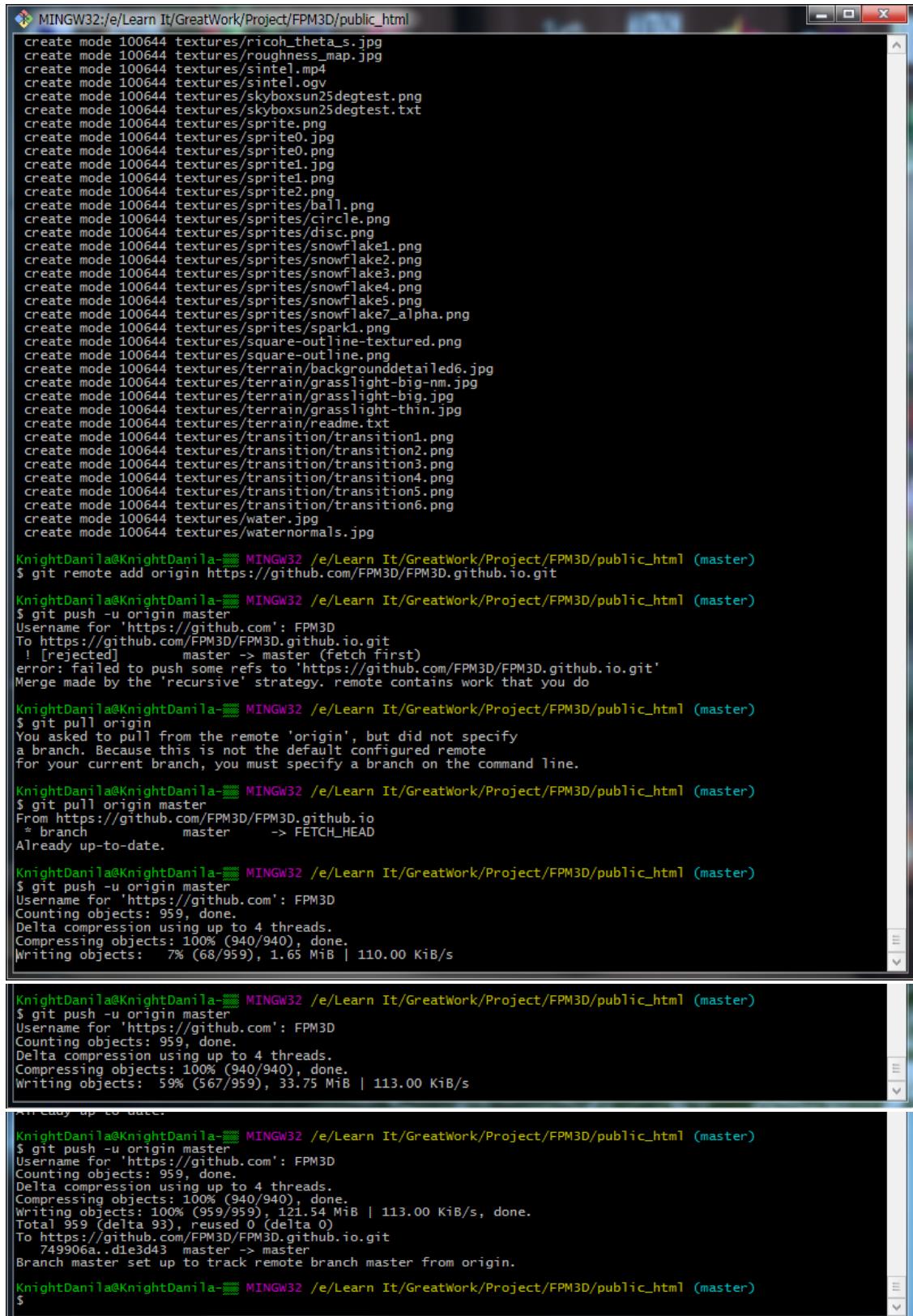


```
MINGW32:/e/Learn It/GreatWork/Project/FPM3D/public_html
create mode 100644 textures/lensflare1.png
create mode 100644 textures/lensflare2.jpg
create mode 100644 textures/memorial.png
create mode 100644 textures/metal.jpg
create mode 100644 textures/minecraft/atlas.png
create mode 100644 textures/minecraft/dirt.png
create mode 100644 textures/minecraft/grass.png
create mode 100644 textures/minecraft/grass_dirt.png
create mode 100644 textures/minecraft/painterlypack.txt
create mode 100644 textures/miranda_uncropped.hdr
create mode 100644 textures/normal/ninja/ao.jpg
create mode 100644 textures/normal/ninja/displacement.jpg
create mode 100644 textures/normal/ninja/displacement.txt
create mode 100644 textures/normal/ninja/normal.jpg
create mode 100644 textures/pano.webm
create mode 100644 textures/particle2.png
create mode 100644 textures/patterns/bright_squares256.png
create mode 100644 textures/patterns/circuit_pattern.png
create mode 100644 textures/patterns/readme.txt
create mode 100644 textures/perlin-512.png
create mode 100644 textures/planets/earth_atmos_2048.jpg
create mode 100644 textures/planets/earth_atmos_4096.jpg
create mode 100644 textures/planets/earth_clouds_1024.png
create mode 100644 textures/planets/earth_clouds_2048.png
create mode 100644 textures/planets/earth_lights_2048.png
create mode 100644 textures/planets/earth_normal_2048.jpg
create mode 100644 textures/planets/earth_specular_2048.jpg
create mode 100644 textures/planets/moon_1024.jpg
create mode 100644 textures/ricoh_theta_s.jpg
create mode 100644 textures/roughness_map.jpg
create mode 100644 textures/sintel.mp4
create mode 100644 textures/sintel.ogv
create mode 100644 textures/skyboxsun25degtest.png
create mode 100644 textures/skyboxsun25degtest.txt
create mode 100644 textures/sprite.png
create mode 100644 textures/sprite0.jpg
create mode 100644 textures/sprite0.png
create mode 100644 textures/sprite1.jpg
create mode 100644 textures/sprite1.png
create mode 100644 textures/sprite2.png
create mode 100644 textures/sprites/ball.png
create mode 100644 textures/sprites/circle.png
create mode 100644 textures/sprites/disc.png
create mode 100644 textures/sprites/snowflake1.png
create mode 100644 textures/sprites/snowflake2.png
create mode 100644 textures/sprites/snowflake3.png
create mode 100644 textures/sprites/snowflake4.png
create mode 100644 textures/sprites/snowflake5.png
create mode 100644 textures/sprites/snowflake7_alpha.png
create mode 100644 textures/sprites/spark1.png
create mode 100644 textures/square-outline-textured.png
create mode 100644 textures/square-outline.png
create mode 100644 textures/terrain/backgrounddetailed6.jpg
create mode 100644 textures/terrain/grasslight-big-nm.jpg
create mode 100644 textures/terrain/grasslight-big.jpg
create mode 100644 textures/terrain/grasslight-thin.jpg
create mode 100644 textures/terrain/readme.txt
create mode 100644 textures/transition/transition1.png
create mode 100644 textures/transition/transition2.png
create mode 100644 textures/transition/transition3.png
create mode 100644 textures/transition/transition4.png
create mode 100644 textures/transition/transition5.png
create mode 100644 textures/transition/transition6.png
create mode 100644 textures/water.jpg
create mode 100644 textures/waternormals.jpg

KnightDanila@KnightDanila-MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)
$ |
```

**Рис. 7.2.4. Додали нові файли.**

Тепер почалося завантаження проекту на сервер GitHub. Ця процедура може зайняти тривалий час, залежатиме від швидкості Інтернету, сервера і типу акаунта (Рис. 7.2.5.).



```

MINGW32:/e/Learn It/GreatWork/Project/FPM3D/public_html
create mode 100644 textures/ricoh_theta_s.jpg
create mode 100644 textures/roughness_map.jpg
create mode 100644 textures/sintel.mp4
create mode 100644 textures/skyboxsun25degtest.png
create mode 100644 textures/skyboxsun25degtest.txt
create mode 100644 textures/sprite.png
create mode 100644 textures/sprite0.jpg
create mode 100644 textures/sprite0.png
create mode 100644 textures/sprite1.jpg
create mode 100644 textures/sprite1.png
create mode 100644 textures/sprite2.png
create mode 100644 textures/sprites/ball.png
create mode 100644 textures/sprites/circle.png
create mode 100644 textures/sprites/disc.png
create mode 100644 textures/sprites/snowflake1.png
create mode 100644 textures/sprites/snowflake2.png
create mode 100644 textures/sprites/snowflake3.png
create mode 100644 textures/sprites/snowflake4.png
create mode 100644 textures/sprites/snowflake5.png
create mode 100644 textures/sprites/snowflake7_alpha.png
create mode 100644 textures/sprites/spark1.png
create mode 100644 textures/square-outline-textured.png
create mode 100644 textures/terrain/backgrounddetailed6.jpg
create mode 100644 textures/terrain/grasslight-big-nm.jpg
create mode 100644 textures/terrain/grasslight-big.jpg
create mode 100644 textures/terrain/grasslight-thin.jpg
create mode 100644 textures/terrain/readme.txt
create mode 100644 textures/transition/transition1.png
create mode 100644 textures/transition/transition2.png
create mode 100644 textures/transition/transition3.png
create mode 100644 textures/transition/transition4.png
create mode 100644 textures/transition/transition5.png
create mode 100644 textures/transition/transition6.png
create mode 100644 textures/water.jpg
create mode 100644 textures/waternormals.jpg

KnightDanila@KnightDanila- [MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)]
$ git remote add origin https://github.com/FPM3D/FPM3D.github.io.git
KnightDanila@KnightDanila- [MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)]
$ git push -u origin master
Username for 'https://github.com': FPM3D
To https://github.com/FPM3D/FPM3D.github.io.git
 ! [rejected]          master -> master (fetch first)
error: failed to push some refs to 'https://github.com/FPM3D/FPM3D.github.io.git'
Merge made by the 'recursive' strategy. remote contains work that you do

KnightDanila@KnightDanila- [MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)]
$ git pull origin
You asked to pull from the remote 'origin', but did not specify
a branch. Because this is not the default configured remote
for your current branch, you must specify a branch on the command line.

KnightDanila@KnightDanila- [MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)]
$ git pull origin master
From https://github.com/FPM3D/FPM3D.github.io
 * branch            master      -> FETCH_HEAD
Already up-to-date.

KnightDanila@KnightDanila- [MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)]
$ git push -u origin master
Username for 'https://github.com': FPM3D
Counting objects: 959, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (940/940), done.
Writing objects:  7% (68/959), 1.65 MiB | 110.00 KiB/s

KnightDanila@KnightDanila- [MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)]
$ git push -u origin master
Username for 'https://github.com': FPM3D
Counting objects: 959, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (940/940), done.
Writing objects: 59% (567/959), 33.75 MiB | 113.00 KiB/s

KnightDanila@KnightDanila- [MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)]
$ git push -u origin master
Username for 'https://github.com': FPM3D
Counting objects: 959, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (940/940), done.
Writing objects: 100% (959/959), 121.54 MiB | 113.00 KiB/s, done.
Total 959 (delta 93), reused 0 (delta 0)
To https://github.com/FPM3D/FPM3D.github.io.git
    749906a..die3d43  master -> master
Branch master set up to track remote branch master from origin.

KnightDanila@KnightDanila- [MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)]
$ 
```

Рис. 7.2.5. Завантаження нових файлів.

Тепер треба увійти на головну сторінку свого акаунту GitHub. Там з'явився новий репозиторій FPM3D.github.io (Рис. 7.2.6.).

The screenshot shows a browser window with several tabs open. The active tab is 'GitHub, Inc. [US] https://github.com/FPM3D'. The page displays the user's profile information, including a purple 'H' logo, a bio placeholder 'FPM3D', and statistics for followers, starred repositories, and following users, all of which are currently at zero. Below this is a 'Popular repositories' section showing 'FPM3D.github.io'. A prominent feature is a 'Contribution graph' titled '0 contributions in the last year', which is currently empty. A summary below it explains how contributions are tracked and encourages creating a first repository.

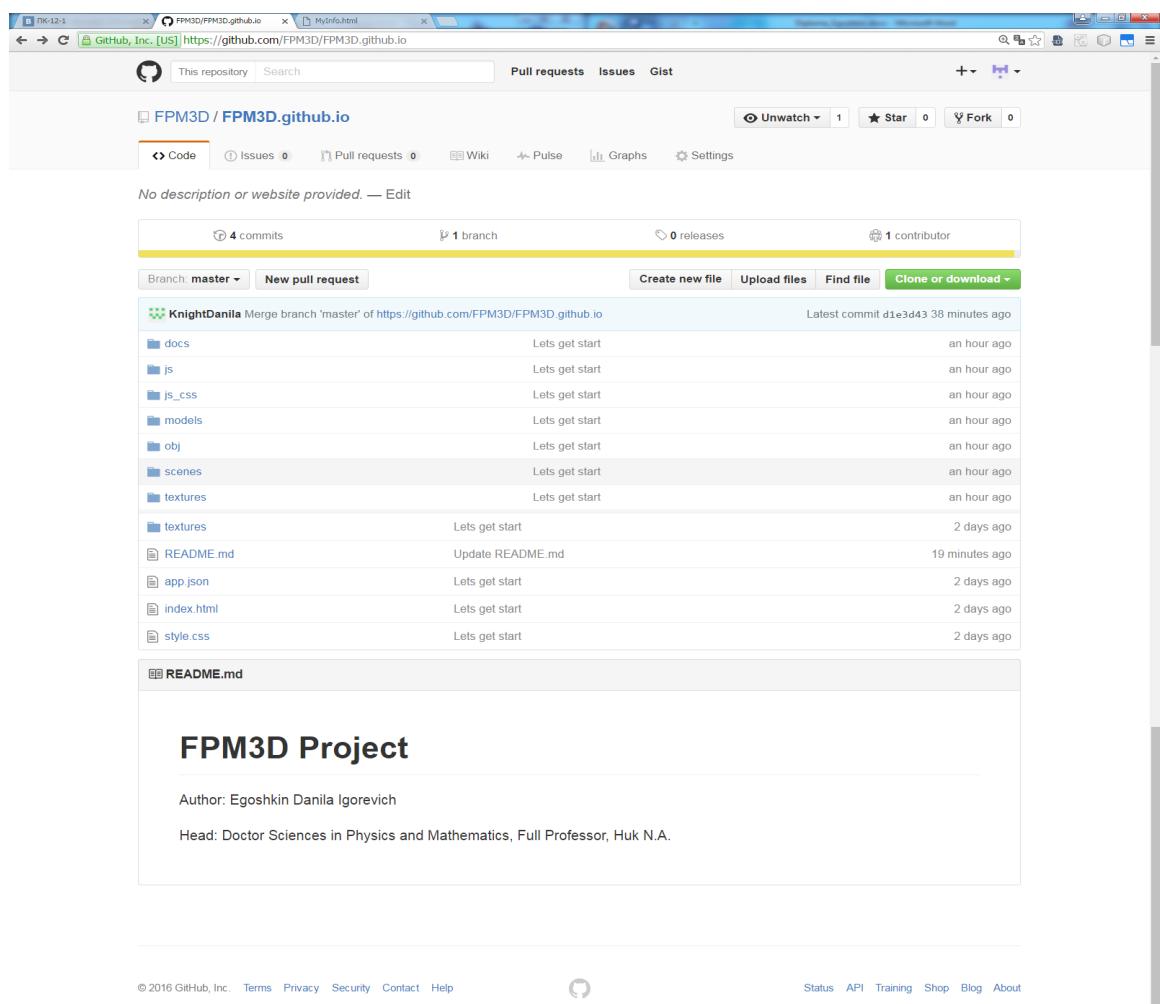
**Рис. 7.2.6. Репозиторій FPM3D.github.io.**

Файли репозиторій FPM3D на комп'ютері (Рис. 7.2.7.).

Имя	Тип	Размер
.git	Папка с файлами	
docs	Папка с файлами	
js	Папка с файлами	
js_css	Папка с файлами	
models	Папка с файлами	
obj	Папка с файлами	
scenes	Папка с файлами	
textures	Папка с файлами	
app.json	Файл "JSON"	4 КБ
favicon.ico	Значок	162 КБ
index.html	Chrome HTML Docu...	13 КБ
README.md	Файл "MD"	1 КБ
style.css	Файл "CSS"	1 КБ

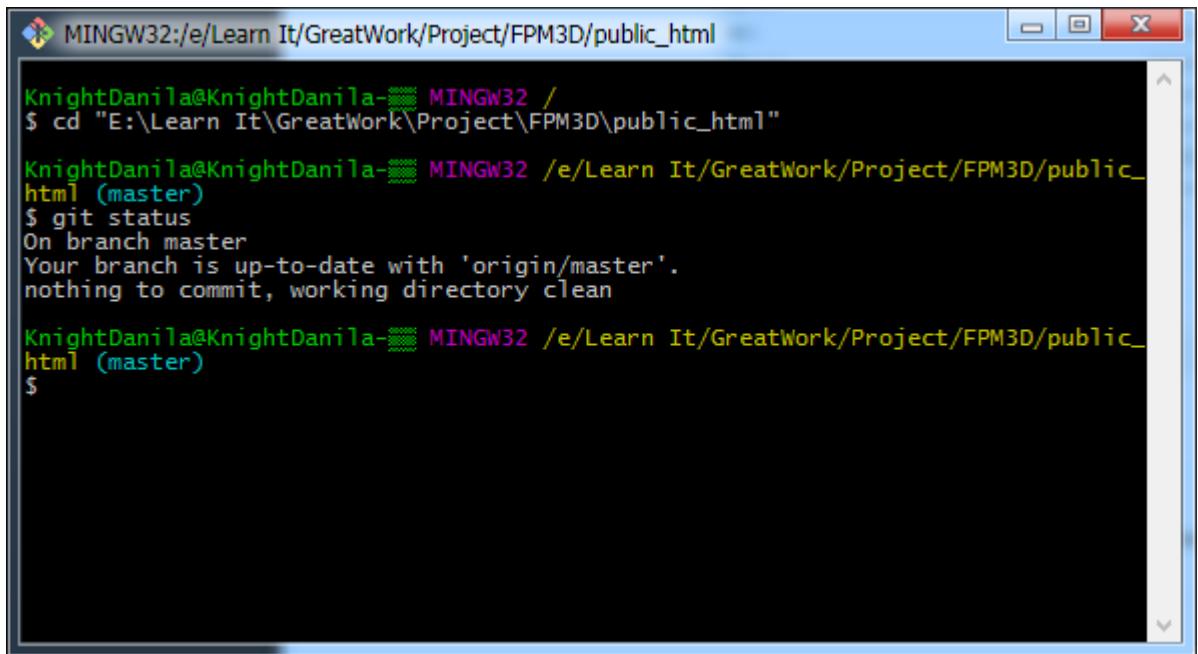
**Рис. 7.2.7. Репозиторій FPM3D.github.io.**

Тепер і усередині репозиторія знаходяться всі файли, які були на комп'ютері. Перейдемо на сторінку репозіторія для того, щоб переконатися у цьому (Рис. 7.2.8.).



**Рис. 7.2.8. Репозиторій FPM3D.github.io.**

За допомогою програми Git та команди “git status” дізнаємося про то, чи всі файли було завантажено до сервера, та чи нема змінених файлів (Рис. 7.2.9.).



```

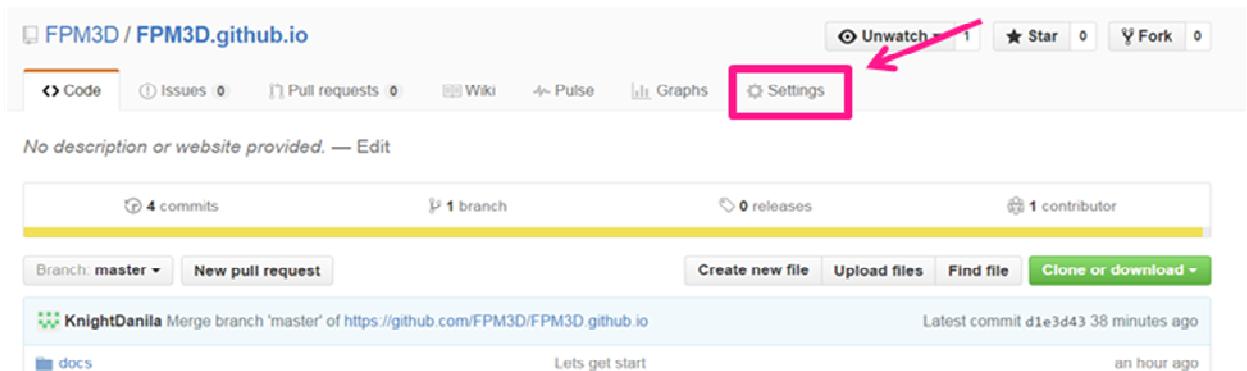
MINGW32:/e/Learn It/GreatWork/Project/FPM3D/public_html
$ cd "E:\Learn It\GreatWork\Project\FPM3D\public_html"
KnightDanila@KnightDanila-MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean

KnightDanila@KnightDanila-MINGW32 /e/Learn It/GreatWork/Project/FPM3D/public_html (master)
$
```

**Рис. 7.2.9. Репозиторій FPM3D.github.io.**

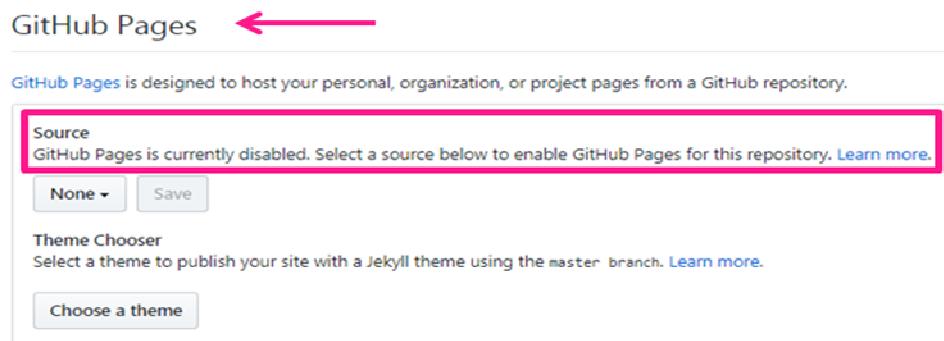
Всі файли було завантажено, тепер кожен зможе створювати гілки змін для проекту, та продовжувати його подальшу розробку. Створення гілок дуже важливо, адже якщо вносити зміни тільки до гілки майстра, то дуже важко буде розробляти сумісний проект та знаходити помилки і відкатувати проект до попередніх версій.

Тепер треба перевірити / налаштувати доступ до сервера за адресою <http://fpm3d.github.io/>. Для цього зайдемо в налаштування репозіторію (Рис. 7.2.10.).



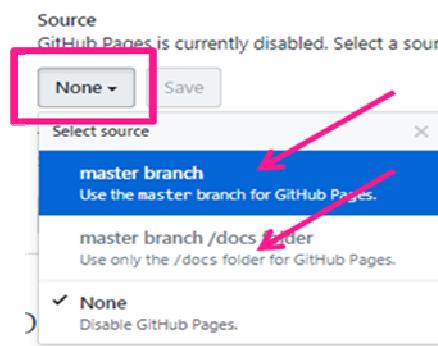
**Рис. 7.2.10. Репозиторій FPM3D.github.io.**

Тепер у налаштуваннях знайдемо пункт GitHub Pages, та перевіримо чи увімкнена функція GitHub Pages у пункті Source, якщо ви бачите наступне повідомлення «GitHub Pages is currently disabled. Select a source below to enable GitHub Pages for this repository.» - це означає, що функція не увімкнена (Рис. 4.2.11.).



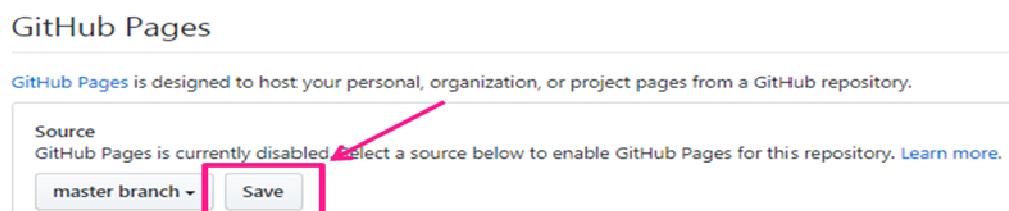
**Рис. 4.2.11. Репозиторій FPM3D.github.io.**

Для того, щоб увімкнути GitHub Pages, натисніть на кнопку «None», та виберіть master branch – щоб використовувати гілку майстра, як корінь сайту, або виберіть master branch /docs folder – щоб використовувати папку /docs гілки майстра, як корінь сайту (Рис. 7.2.12.).



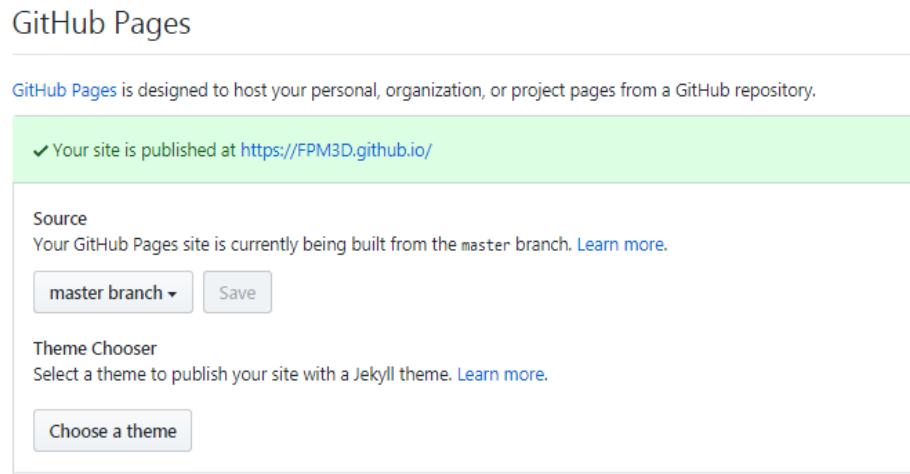
**Рис. 7.2.12. Репозиторій FPM3D.github.io.**

Після цього треба натиснути кнопку «Save» (Рис. 7.2.13.).



**Рис. 7.2.13. Репозиторій FPM3D.github.io.**

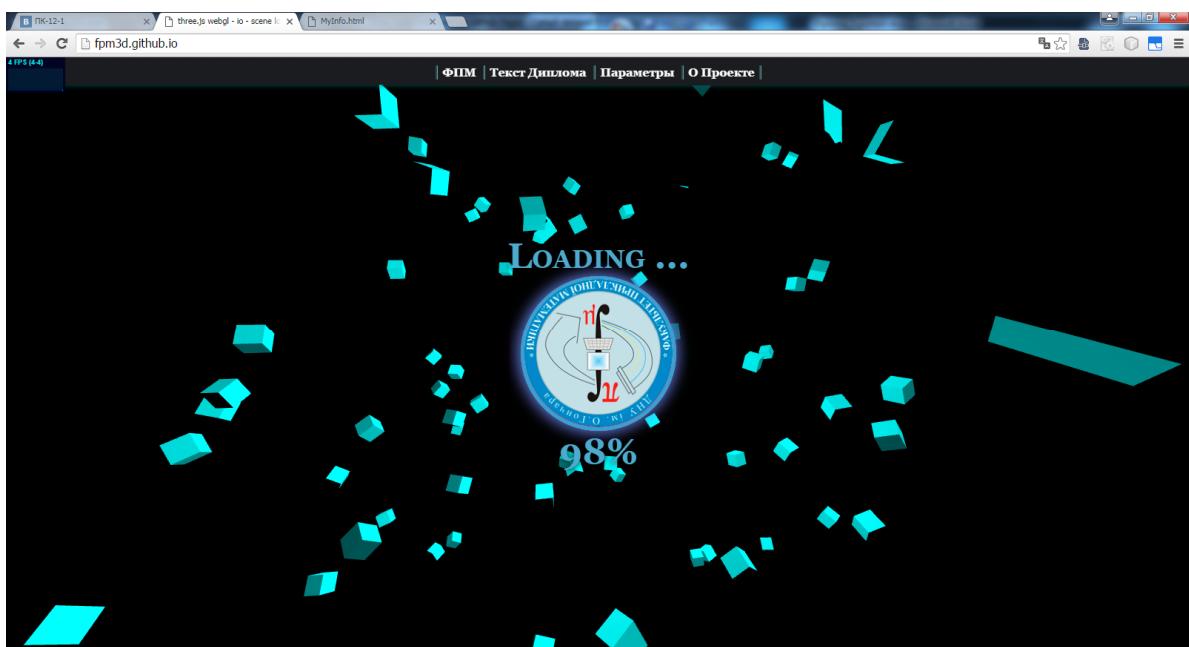
Після цього ми побачимо наступне повідомлення, та адресу публікації сайту (Рис. 7.2.14.).



**Рис. 7.2.14. Репозиторій FPM3D.github.io.**

Також можна створити гілку gh-pages – і GitHub автоматично налаштує її як корінь сайту.

Тепер перейдемо за наступним посиланням <http://fpm3d.github.io/>, щоб перевірити роботу сервера GitHub Pages та веб-додатку, який було загружено на нього (Рис. 7.2.15.).



**Рис. 7.2.15. Проект FPM3D.github.io.**

Тепер за допомогою наступних команд кожен зможе взяти участь в розробці цього проекту, будь-яка зміна має вноситися в певну гілку з ясною назвою і коротким описом того, який саме функціонал було додано, які саме зміни були внесені та в які файли. Після чого посилається запит на злиття гілки майстра з гілкою зі змінами та додатковим функціоналом. Потім гілка зливається в гілку майстра. Кількість гілок, що надається сервером необмежена.

Короткий перелік команд з описом для роботи з репозиторієм:

- cd “Path” – За допомогою команди cd перейдемо у потрібну дерікторію, наприклад до папки “SecretFolder”, що знаходиться на диску X cd “X:\SecretFolder”. Далі все команди будуть виконуватися відносно цієї папки, тобто вона і стане місцем де буде зберігатися копія репозиторія. Якщо в цій папці вже є репозиторій, то програма виведе назву гілки, яка знаходиться в цій папці, назва буде написано в дужках. Наприклад так для гілки майстра:

KnightDanila@KnightDanila- MINGW32 /X/SecretFolder (master).

Чи так для гілки NewSecretDesign:

KnightDanila@KnightDanila- MINGW32 /X/SecretFolder (NewSecretDesign)

- git clone <https://github.com/FPM3D/FPM3D.github.io.git> - Якщо ви бажаєте отримати копію Git репозиторія FPM3D, для того що б взяти участь в розробці проекту, виконайте команду цю команду. Результат: каталог з гілкою master. Тепер можна створювати нові гілки, або використовувати існуючі;
- git status - виводить статус гілки. Основний інструмент, який використовується для визначення, які файли в якому стані знаходяться. Якщо ви виконаете цю команду відразу після клонування, ви побачите щось на зразок цього:

```
$ Git status
```

```
# On branch master
```

```
nothing to commit (working directory clean)
```

Це означає, що у вас чистий робочий каталог, іншими словами-в ньому немає відслідковуються змінених файлів. Команда також повідомляє вам на який гілці (branch) ви зараз перебуваєте. Поки що це гілка master - це гілка за замовчуванням;

- git branch testing - створює в репозиторії нову гілку testing. Після створення гілки, необхідно перейти у неї за допомогою наступної команди;
- git checkout testing – ця команда дозволяє перейти в гілку testing;
- Щоб відправити файли у свою гілку (це називається зробити комміт), треба виконати наступну послідовність команд, ця послідовність команда відправить усі зміни у гілку DogModel, тому треба написати замість DogModel назву вашої гілки:
  - 1) git add .
  - 2) git commit -m "Commit description"
  - 3) git push origin DogModel
- git add . - точка в кінці дуже важлива, це означає, що ми хочимо додати файли всіх типів. Ця команда говорить, що ми хочемо

позначити зміни і додати їх в список змінених файлів для наступної відправки у свою гілку на сервері.

- git commit -m "Commit description" - ця команда виконується відразу після команди add і означає додавання інформації про ті файли, що ми змінили, наприклад, який функціонал додали ці зміни “git commit -m "added new car to game."” ;
  - git push origin DogModel – відправляє всі файли, що ми змінили на сервер у гілку DogModel, якщо ваша гілка має іншу назву напишіть її, наприклад наша гілка має назву CatModel, тоді git push origin CatModel;
  - Завантажити зміни з сервера або з іншої гілки, можна за допомогою наступних команд (Зараз ми завантажуємо файли з гілки майстра, та з гілки DogModel):
    - 1) git pull origin
    - 2) git merge DogModel
  - git pull origin – завантажує в гілку у якій ми знаходимся усі зміни з гілки майстра, тобто з головного проекту. Ми це робимо тому, що в ній завжди має знаходитися нова та робоча версія проекту;
  - git merge DogModel – завантажує в гілку у якій ми знаходимся усі зміни з гілки DogModel;
  - git rm bug.txt – ця команда дозволяє видаляти файли. В нашому випадку вона видалить файл bug.txt;
  - git branch -a – виклик цієї команди дозволяє переглянути всі існуючі гілки репозиторія;
  - git branch -r - дозволяє переглянути список всіх віддалених гілок.
- При виконанні цієї командирівку ви побачите наступне:
- ```
origin/HEAD
origin/master
...

```

- `git branch -d bugBranch` – ця команда дозволяє видалити гілку з репозиторію. В нашому буде віддано гілку `bugBranch`. Але якщо в ході видалення Git виявить помилки, він відмовиться видаляти гілку, в цьому випадку необхідно використовувати наступну команду;
- `git branch -D bugBranch` – за допомогою цієї команди будь-яка гілка видаляється з репозиторію примусово, навіть якщо в ході видалення були виявлені помилки. В нашому випадку, буде примусово віддано гілку `bugBranch`;
- `git log` – виводить список всіх змін (коммітов, файлів, гілок) та їх SHA1 хеші. Після виконння цієї команди, ви побачите щось на зразок цього:

```
commit ee1538f2e7769cfa37541d3e56b74627c530401f
```

Author: KnightDanila [KnightDanila@gmail.com](mailto:KnightDanila@gmail.com)

Date: Sat Jun 11 02:58:03 2016 +0300

add modal window

`ee1538f2e7769cfa37541d3e56b74627c530401f` - це SHA1 хеш код комміту.

KnightDanila – автор цього коміту, що вносив зміни у файли.

Date – час, коли було зроблено ці зміни

`add modal window` – короткий опис комміту. В нашому випадку, було додано модальне вікно.

- `git log --stat --graph` – ця команда аналогічна попередній, але більш наочно описує історію з іменами файлів і псевдографічним зображенням гілок.
- `git show ee1538f2e7769cfa37541d3e56b74627c530401f` - за допомогою даної команди можна переглянути все зміни, зроблені в заданому комміті. В нашому випадку, буде переглянуто комміт, що має хеш код зі значенням `ee1538f2e7769cfa37541d3e56b74627c530401f`;

- `git blame bug.txt` – ця команда показує хто і коли змінив кожен рядок обраного файлу. На відміну від багатьох інших систем контролю версій ця операція відбувається офлайн, тобто дані беруться з локального диска;
- `git reset --hard ee1538f2e7769cfa37541d3e56b74627c530401f` – ця команда дозволяє стерти всі зміни, які відбулися в проекті після цього комміту, тобто повернути проект до конкретного комміту.
- `git reset --soft ee1538f2e7769cfa37541d3e56b74627c530401f` – аналогічно попередній команді, але файли на диску залишаються без змін.
- `git diff` – Відображення змін у файлах у порівнянні з останнім коммітом.
- `git diff <filename>` – Показувати зміни в окремому файлі порівняно з останнім коммітом.
- `git diff <commit id> <commit id 2>` – Показати зміни між двома різними коммітами.

## **ВИСНОВКИ**

1. Дані моделі дозволяють створити реалістичні ефекти навколошнього світу і поведінки об'єктів, що дозволяють перенести користувача в світ інтерактивного додатку і створити атмосферу повного занурення. Моделі є максимально адаптовані для застосування їх в веб-додатках.
2. На підставі результатів дослідження сучасного стану розвитку проблеми створення якісних 2D і 3D інтерактивних додатків та веб-додатків, була сформульована математична модель тривимірного звуку, модель анімації поведінки камери, модель Гатлінга, модель штучного інтелекту NPC птиці.
3. Було зроблено порівняння різних моделей тривимірного звуку на основі яких була вибрана модель ефекту Доплера. Була розроблена і реалізована UML діаграма моделі об'ємного звуку та реалізована модель Гатлінга.
4. Було зроблено порівняння різних моделей анімації, та зроблено висновки до їх застосування. Була обрана модель процедурної анімації. Та розроблена її програмна реалізація.
5. Проект було розроблено за допомогою наступних безкоштовних програм: Git, NetBeans, Gimp, Blender, AwesomeBump, Paint.NET, Google Chrome, VirtualBox, Mozilla Firefox.
6. Було зроблено порівняння різних моделей штучного інтелекту. Та розроблена модель штучного інтелекту NPC птиці.
7. За допомогою бібліотеки ThreeJS та JS було розроблено моделі двовимірних і тривимірних спрайтів.
8. За результатами дослідження зроблено доповідь на конференції «Сучасні науково-технічні дослідження у контексті мовного простору

(іноземними мовами)» 14 квітня 2017, «CLASSIFICATION ALGORITHM ON THE BASIS OF FUZZY LOGIC WITH EXTENDABLE NUMBER OF OUTPUTS».

9. За результатами дослідження зроблено доповідь на конференції «Сучасні науково-технічні дослідження у контексті мовного простору (іноземними мовами)» 14 квітня 2017, «CLASSIFICATION ALGORITHM ON THE BASIS OF FUZZY LOGIC WITH EXTENDABLE NUMBER OF OUTPUTS».

## ***СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ***

1. Richard S.W. Nicholas H. Graham S. - OpenGL SuperBible (6th Edition) – Addison-Wesley, 2013, 1474.
2. Adam Lake Game Programming Gems 8 1st Edition. - Course Technology PTR, 2011, 557.
3. Creative Technology. - OpenAL Programmer's Guide - Creative Technology, 2007, 142 p.
4. Кологризов В. Н. Эффект Доплера в классической физике. — М.: МФТИ, 2012, С. 25—26, 32 с.
5. Лауэ М. История физики. — Москва: ГИТЛ, 1956, 229 с.
6. Вікіпедія: Pulse code modulation [Электронный ресурс]. — Режим доступу: [https://en.wikipedia.org/wiki/Pulse-code\\_modulation](https://en.wikipedia.org/wiki/Pulse-code_modulation).
7. Nicola Valcasara Unreal Engine Game Development Blueprints. - Packt Publishing Ltd, 2015, 352 с.
8. Панфилов И.П., Дырда В.Е. Теория электрической связи. — М.: Радио и связь, 1991, 344 с.
9. Высоцкий М. З. Системы кино и стереозвук. М.: Искусство, 1972, 336 с.
10. Ландсберг Г.С. Элементарный учебник физики. — М.: Физматлит, 1984, Т. 3. Колебания и волны. Оптика. Атомная и ядерная физика, 656 с.
11. Бессон Р. Все о стереофонии. М.-Л, Госэнергоиздат, 1963, 128 стр.
12. Мейсон, МакКаски. Звук в играх: технологии программирования. М. : КУДИЦ-ОБРАЗ, 2004, 370 с.
13. Вікіпедія: Wiki user page KnightDanila [Электронный ресурс]. — Режим доступу: <https://en.wikipedia.org/wiki/User:KnightDanila>
14. Matt Pharr Randima Fernando GPU Gems 2 –Addison Wesley, 2005. – 834 с.

15. Шаньгин В.Ф. Защита информации в компьютерных системах и сетях М.: ДМК Пресс, 2012, 593 с.
16. Blair Preston. Cartoon Animation. –Walter Foster Publishing, 1994, 224 с.
17. Kelly L. Murdock 3ds Max 2010 Bible.-Wiley Publishing, 2009. – 1312 р.
18. Ландсберг Г.С. Элементарный учебник физики. — М.: Физматлит, 1984, Т. 1. Механика. Теплота. Молекулярная физика., 616 с.
19. Вікіпедія: Audio file format [Электронный ресурс]. — Режим доступу: [https://en.wikipedia.org/wiki/Audio\\_file\\_format](https://en.wikipedia.org/wiki/Audio_file_format)
20. Джексон П. Введение в экспертные системы / Питер Джексон // Introduction to Expert Systems. – 3-е изд. — М.: Вильямс, 2001. — С. 624
21. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л. Рутковский; пер. с польск. И. Д. Рудинского. – М.: Горячая линия – Телеком, 2006. – 452 с
22. Rouse, Richard. Game Design: Theory & Practice : Los Rios Boulevard, Plano, Texas, USA:Wordware Publishing, 2004. — 698 р.
23. Заде Л.А. Понятие лингвистической переменной и его применение к принятию приближенных решений. М.:Мир, 1976.
24. Вікіпедія: Гральний рушій [Электронный ресурс]. — Режим доступу: [https://uk.wikipedia.org/wiki/Гральний\\_рушій](https://uk.wikipedia.org/wiki/Гральний_рушій).
25. Jos Dirksen Learning Three.js: The JavaScript 3D Library for WebGL. - Packt Publishing Ltd, 2015, 402 с.
26. Patrick Cozzi WebGL Insights - CRC Press, 2015, 416 с.
27. Леоненков А.В. Нечеткое моделирование в среде MATLAB и fuzzyTECH / А.В. Леоненков. – СПб.: БХВ – Петербург, 2005. – 736 с.
28. Negnevitsky M. Artificial intelligence: a guide to intelligent systems. // Addison-Wesley, 2002. 394 р.
29. Ротштейн А.П. "Интеллектуальные технологии идентификации: нечеткая логика, генетические алгоритмы, нейронные сети". – [Электронный

- ресурс]. – Режим доступа:  
<http://matlab.exponenta.ru/fuzzylogic/book5/index.php>
30. Штовба, С. Д. Проектирование нечетких систем средствами MATLAB / С. Д. Штовба. – М.: Горячая линия – Телеком, 2007. – 288 с.
31. Рыбин В.В. Основы теории нечетких множеств и нечеткой логики. — М.: МАИ, 2007. — С. 96
32. Люгер Дж. Ф. Искусственный интеллект: стратегии и методы решения сложных проблем. – 4-е изд. – М.: Вильямс, 2003. 860 с.
33. Ярушкина Н. Г. Основы теории нечетких и гибридных систем. М.: Финансы и статистика, 2004.- 320 с.
34. Жданов А. А., Караваев М. В. Применение нечеткой логики в имитационной системе автономного адаптивного управления // Труды ИСП РАН. 2002. №. URL: <http://cyberleninka.ru/article/n/primenie-nechetkoy-logiki-v-imitatsionnoy-sisteme-avtonomnogo-adaptivnogo-upravleniya> (дата обращения: 29.11.2017).
35. Матвеев Е. В., Глинчиков В. А. Адаптивная эталонная модель в системе управления беспилотного летательного аппарата // Вестник СибГУ им. М.Ф. Решетнева. 2010. №6. URL: <http://cyberleninka.ru/article/n/adaptivnaya-etalonnayamodel-v-sisteme-upravleniya-bespilotnogo-letatelnogo-apparata> (дата обращения: 29.11.2017).
36. Шишкун Е. В. Боресков А. В. "Компьютерная графика". – М.: Диалог-МИФИ, 1995, 288 с.
37. John M. Blain The Complete Guide to Blender Graphics, Second Edition – Computer Modeling and Animation, Edition 2 – CRC Press, 2014. 515 с.
38. Akkana Peck Beginning GIMP – From Novice to Professional, Edition 2 – Apress, 2009. 551 с.
39. Егошкин Д.И. "knightdanila.github.io ". – [Электронный ресурс]. – Режим доступа: <https://knightdanila.github.io>
40. Егошкин Д.И. "Audio test". – [Электронный ресурс]. – Режим доступа: <https://knightdanila.github.io/WebAudio/GatlingGun/Audio%20test.htm>

41. Егошкин Д.И. "Walking - Test 1". – [Электронный ресурс]. – Режим доступа: [https://knightdanila.github.io/Walking/test\\_1.html](https://knightdanila.github.io/Walking/test_1.html)
42. Егошкин Д.И. "Walking - Test 2". – [Электронный ресурс]. – Режим доступа: [https://knightdanila.github.io/Walking/test\\_2.html](https://knightdanila.github.io/Walking/test_2.html)
43. Егошкин Д.И. " Walking - Test 3". – [Электронный ресурс]. – Режим доступа: [https://knightdanila.github.io/Walking/test\\_3.html](https://knightdanila.github.io/Walking/test_3.html)
44. Егошкин Д.И. " Walking - Test 4". – [Электронный ресурс]. – Режим доступа: [https://knightdanila.github.io/Walking/test\\_4.html](https://knightdanila.github.io/Walking/test_4.html)
45. Егошкин Д.И. "CMD - Combat Marvel DC". – [Электронный ресурс]. – Режим доступа: <https://knightdanila.github.io/CMD/index.html>
46. Scott Chacon Pro Git 2012th Edition – Apress, 2012. 282 с.
47. Егошкин Д.И. "Flying Bird". – [Электронный ресурс]. – Режим доступа: <https://knightdanila.github.io/AI/Bird/index.html>
48. Егошкин Д.И. " Sprite3DViewer ". – [Электронный ресурс]. – Режим доступа: <https://knightdanila.github.io/Sprite3DViewer>
49. Егошкин Д.И. " Sprite 2D ". – [Электронный ресурс]. – Режим доступа: [knightdanila.github.io/Walking/Sprite\\_test\\_1.html](https://knightdanila.github.io/Walking/Sprite_test_1.html)